

1

2 1.1 Scope

3 IEEE Std. 1003.1-200x defines a standard operating system interface and environment, including
4 a command interpreter (or “shell”), and common utility programs to support applications
5 portability at the source code level. It is intended to be used by both applications developers
6 and system implementors.

7 IEEE Std. 1003.1-200x comprises three major components (each in an associated volume):

- 8 1. General terms, concepts, and interfaces common to all volumes of IEEE Std. 1003.1-200x,
9 including utility conventions and C language header definitions, are included in the Base
10 Definitions volume of IEEE Std. 1003.1-200x.
- 11 2. Definitions for system service functions and subroutines, language-specific system
12 services for the C programming language, function issues, including portability, error
13 handling, and error recovery, are included in the System Interfaces volume of
14 IEEE Std. 1003.1-200x.
- 15 3. Definitions for a standard source code-level interface to command interpretation services
16 (a “shell”) and common utility programs for application programs are included in the
17 Shell and Utilities volume of IEEE Std. 1003.1-200x.

18 The following areas are outside of the scope of IEEE Std. 1003.1-200x:

- 19 • Graphics interfaces
- 20 • Database management system interfaces
- 21 • Record I/O considerations
- 22 • Object or binary code portability
- 23 • System configuration and resource availability

24 IEEE Std. 1003.1-200x describes the external characteristics and facilities that are of importance
25 to applications developers, rather than the internal construction techniques employed to achieve
26 these capabilities. Special emphasis is placed on those functions and facilities that are needed in
27 a wide variety of commercial applications.

28 The facilities provided in IEEE Std. 1003.1-200x are drawn from the following base documents:

- 29 • IEEE Std. 1003.1-1996 (POSIX-1) (incorporating IEEE Stds. 1003.1-1990, 1003.1b-1993,
30 1003.1c-1995, and 1003.1i-1995)
- 31 • The following amendments to the POSIX.1-1990 standard:
 - 32 — IEEE P1003.1a draft standard (Additional System Services)
 - 33 — IEEE Std. 1003.1d-1999 (Additional Realtime Extensions)
 - 34 — IEEE Std. 1003.1g-2000 (Protocol-Independent Interfaces (PII))
 - 35 — IEEE Std. 1003.1j-2000 (Advanced Realtime Extensions)
 - 36 — IEEE Std. 1003.1q-2000 (Tracing)

- 37 • IEEE Std. 1003.2-1992 (POSIX-2) (includes IEEE Std. 1003.2a-1992)
- 38 • The following amendment to the ISO POSIX-2: 1993 standard:
- 39 — IEEE P1003.2b draft standard (Additional Utilities)
- 40 — IEEE Std. 1003.2d-1994 (Batch Environment)
- 41 • Open Group Technical Standard, February 1997, System Interface Definitions, Issue 5 (XBD5)
- 42 (ISBN: 1-85912-186-1, C605)
- 43 • Open Group Technical Standard, February 1997, Commands and Utilities, Issue 5 (XCU5)
- 44 (ISBN: 1-85912-191-8, C604)
- 45 • Open Group Technical Standard, February 1997, System Interfaces and Headers, Issue 5
- 46 (XSH5) (in 2 Volumes) (ISBN: 1-85912-181-0, C606)
- 47 **Note:** XBD5, XCU5, and XSH5 are collectively referred to as the *Base Specifications*.
- 48 • Open Group Technical Standard, January 2000, Networking Services, Issue 5.2 (XNS5.2)
- 49 (ISBN: 1-85912-241-8, C808)
- 50 • ISO/IEC 9899: 1999, Programming Languages — C.
- 51 IEEE Std. 1003.1-200x uses the *Base Specifications* as its organizational basis and adds the
- 52 following additional functionality to them drawn from the base documents above:
- 53 • Normative text from the ISO POSIX-1: 1996 standard and the ISO POSIX-2: 1993 standard not
- 54 included in the *Base Specifications*
- 55 • The amendments to the POSIX.1-1990 standard and the ISO POSIX-2: 1993 standard listed
- 56 above, except for parts of IEEE Std. 1003.1g-2000
- 57 • Portability Considerations
- 58 • Additional rationale and notes
- 59 The following features, marked legacy or obsolescent in the base documents, are not carried
- 60 forward into IEEE Std. 1003.1-200x. Other features from the base documents marked legacy or
- 61 obsolescent are carried forward unless otherwise noted.
- 62 From XSH5, the following legacy interfaces, headers, and external variables are not carried
- 63 forward:
- 64 *advance()*, *brk()*, *chroot()*, *compile()*, *cuserid()*, *gamma()*, *getdtablesize()*, *getpagesize()*, *getpass()*,
- 65 *getw()*, *putw()*, *re_comp()*, *re_exec()*, *regcmp()*, *sbrk()*, *sigstack()*, *wait3()*, *<re_comp.h>*,
- 66 *<regexp.h>*, *<varargs.h>*, *loc1*, *__loc1*, *loc2*, *locs*
- 67 From XCU5, the following legacy utilities are not carried forward:
- 68 *calendar*, *cancel*, *cc*, *col*, *cpio*, *cu*, *dircmp*, *dis*, *egrep*, *fgrep*, *line*, *lint*, *lpstat*, *mail*, *pack*, *pcat*, *pg*, *spell*,
- 69 *sum*, *tar*, *unpack*, *uulog*, *uname*, *uupick*, *uuto*
- 70 In addition, legacy features within non-legacy reference pages (for example, headers) are not
- 71 carried forward.
- 72 From the ISO POSIX-1: 1996 standard, the following obsolescent features are not carried
- 73 forward:
- 74 Page 112, CLK_TCK
- 75 Page 197 *tgetattr()* rate returned option
- 76 From the ISO POSIX-2: 1993 standard, obsolescent features within the following pages are not
- 77 carried forward:

78 Page 75 zero-length prefix within PATH
 79 Page 156, 159 *set*,
 80 Page 178, *awk*, use of no argument and no parentheses with length
 81 Page 259, *ed*
 82 Page 272, *env*
 83 Page 282, *find -perm[-]onum*,
 84 Page 295-296, *egrep*
 85 Page 299-300, *head*
 86 Page 305-306, *join*
 87 Page 309-310, *kill*
 88 Page 431-433, 435-436, *sort*
 89 Page 444-445, *tail*
 90 Page 453, 455-456, *touch*
 91 Page 464-465, *tty*
 92 Page 472, *uniq*
 93 Page 515-516, *ex*
 94 Page 542-543, *expand*
 95 Page 563-565, *more*
 96 Page 574-576, *newgrp*
 97 Page 578, *nice*
 98 Page 594-596, *renice*
 99 Page 597-598, *split*
 100 Page 600-601, *strings*
 101 Page 624-625, *vi*
 102 Page 693, *lex*

103 The *c89* utility (which specified a compiler for the C Language specified by the
 104 ISO/IEC 9899:1990 standard) has been replaced by a *c99* utility (which specifies a compiler for
 105 the C Language specified by the ISO/IEC 9899:1999 standard).

106 From XSH5, text marked OH has been reviewed on a case-by-case basis and removed where
 107 appropriate. The XCU5 text marked OF, OP, PI, and UN has been reviewed on a case-by-case
 108 basis and removed where appropriate

109 For the networking interfaces, the base document is the XNS, Issue 5.2 specification. The
 110 following parts of the XNS, Issue 5.2 specification are out of scope and not included in
 111 IEEE Std. 1003.1-200x:

- 112 • Part 3 (XTI)
- 113 • Part 4 (Appendixes)

114 Since there is much duplication between the XNS, Issue 5.2 specification and
 115 IEEE Std. 1003.1g-2000, material only from the following sections of IEEE Std. 1003.1g-2000 has
 116 been considered for inclusion:

- 117 • General terms related to sockets (Clause 2.2.2)
- 118 • Socket concepts (Clauses 5.1 through 5.3, inclusive)
- 119 • The *pselect()* function (Clauses 6.2.2.1 and 6.2.3)
- 120 • The *isfdtype()* function (Clause 5.4.8)
- 121 • The `<sys/select.h>` header (Clause 6.2)

122 Emphasis is placed on standardizing existing practice for existing users, with changes and
 123 additions limited to correcting deficiencies in the following areas:

- 124 • Issues raised by IEEE or ISO/IEC Interpretations against IEEE Std. 1003.1 and IEEE Std. |
125 1003.2 |
- 126 • Issues raised in corrigenda for the *Base Specifications* and working group resolutions from The |
127 Open Group |
- 128 • Corrigenda and resolutions passed by The Open Group for the XNS, Issue 5.2 specification |
- 129 • Changes to make the text self-consistent with the additional material merged |
- 130 • A reorganization of the options in order to facilitate profiling, both for smaller profiles such |
131 as IEEE Std 1003.13, and larger profiles such as the Single UNIX Specification |
- 132 • Alignment with the ISO/IEC 9899:1999 standard |

133 **1.2 Conformance**

134 Conformance requirements for IEEE Std. 1003.1-200x are defined in Chapter 2 (on page 19).

135 1.3 Normative References

136 The following standards contain provisions which, through references in this text, constitute
 137 provisions of this volume of IEEE Std. 1003.1-200x. At the time of publication, the editions
 138 indicated were valid. All standards are subject to revision, and parties to agreements based on
 139 this volume of IEEE Std. 1003.1-200x are encouraged to investigate the possibility of applying
 140 the most recent editions of the standards listed below. Members of IEC and ISO maintain
 141 registers of currently valid International Standards.

142 **Notes to Reviewers**

143 *This section with side shading will not appear in the final copy. - Ed.*

144 The following list will be updated.

145 ANS X3.9-1978

146 (Reaffirmed 1989) American National Standard Programming Language FORTRAN.¹

147 ISO/IEC 646

148 ISO/IEC 646: 1991, Information Processing — ISO 7-bit Coded Character Set for Information
 149 Interchange.²

150 ISO 4217

151 ISO 4217: 1995, Codes for the Representation of Currencies and Funds.

152 ISO/IEC 4873

153 ISO/IEC 4873: 1991, Information Technology — ISO 8-bit Code for Information Interchange
 154 — Structure and Rules for Implementation.

155 ISO 8601

156 ISO 8601: 1988, Data Elements and Interchange Formats — Information Interchange —
 157 Representation of Dates and Times.

158 ISO 8859-1

159 ISO 8859-1: 1988, Information Processing — 8-bit Single-byte Coded Graphic Character Sets
 160 — Part 1: Latin Alphabet No. 1.

161 ISO 8859-2

162 ISO 8859-2: 1987, Information Processing — 8-bit Single-byte Coded Graphic Character Sets
 163 — Part 2: Latin Alphabet No. 2.

164 ISO/IEC 9899

165 ISO/IEC 9899: 1999, Programming Languages — C.

166 ISO/IEC 9945-1

167 ISO/IEC 9945-1: 200x, Information Technology — Portable Operating System Interface
 168 (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to
 169 ANSI/IEEE Std 1003.1-200x).³

170

171 1. ANSI documents can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New
 172 York, NY 10018, U.S.A.

173 2. ISO/IEC documents can be obtained from the ISO office: 1 Rue de Varembe, Case Postale 56, CH-1211, Genève 20,
 174 Switzerland/Suisse

175 3. This standard is available from the IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, U.S.A. Tel:
 176 1 (800) 678-IEEE or +1 (908) 981-1393.

177	ISO/IEC 9945-2
178	ISO/IEC 9945-2:1993, Information Technology — Portable Operating System Interface
179	(POSIX) — Part 2: Shell and Utilities.
180	ISO/IEC 10646-1
181	ISO/IEC 10646-1:1993, Information Technology — Universal Multiple-Octet Coded
182	Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.
183	ISO/IEC 14519:1999
184	ISO/IEC 14519:1999, Information Technology — POSIX Ada Language Interfaces —
185	Binding for System Application Program Interface (API) — Realtime Extensions.

186 1.4 Terminology

187 For the purposes of IEEE Std. 1003.1-200x, the following terminology definitions apply:

188 **can**

189 Describes a permissible optional feature or behavior available to the user or application. The
190 feature or behavior is mandatory for an implementation that conforms to
191 IEEE Std. 1003.1-200x. An application can rely on the existence of the feature or behavior.

192 **implementation-defined**

193 Describes a value or behavior that is not defined by IEEE Std. 1003.1-200x but is selected by
194 an implementor. The value or behavior may vary among implementations that conform to
195 IEEE Std. 1003.1-200x. An application should not rely on the existence of the value or
196 behavior. An application that relies on such a value or behavior cannot be assured to be
197 portable across conforming implementations.

198 The implementor shall document such a value or behavior so that it can be used correctly
199 by an application.

200 **legacy**

201 Describes a feature or behavior that is being retained for compatibility with older
202 applications, but which has limitations which make it inappropriate for developing portable
203 applications. New applications should use alternative means of obtaining equivalent
204 functionality.

205 **may**

206 Describes a feature or behavior that is optional for an implementation that conforms to
207 IEEE Std. 1003.1-200x. An application should not rely on the existence of the feature or
208 behavior. An application that relies on such a feature or behavior cannot be assured to be
209 portable across conforming implementations.

210 To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

211 **shall**

212 For an implementation that conforms to IEEE Std. 1003.1-200x, describes a feature or
213 behavior that is mandatory. An application can rely on the existence of the feature or
214 behavior.

215 For an application or user, describes a behavior that is mandatory.

216 **should**

217 For an implementation that conforms to IEEE Std. 1003.1-200x, describes a feature or
218 behavior that is recommended but not mandatory. An application should not rely on the
219 existence of the feature or behavior. An application that relies on such a feature or behavior
220 cannot be assured to be portable across conforming implementations.

221 For an application, describes a feature or behavior that is recommended programming
222 practice for optimum portability.

223 **undefined**

224 Describes the nature of a value or behavior not defined by IEEE Std. 1003.1-200x which
225 results from use of an invalid program construct or invalid data input.

226 The value or behavior may vary among implementations that conform to
227 IEEE Std. 1003.1-200x. An application should not rely on the existence or validity of the
228 value or behavior. An application that relies on any particular value or behavior cannot be
229 assured to be portable across conforming implementations.

230	unspecified	
231		Describes the nature of a value or behavior not specified by IEEE Std. 1003.1-200x which
232		results from use of a valid program construct or valid data input.
233		The value or behavior may vary among implementations that conform to
234		IEEE Std. 1003.1-200x. An application should not rely on the existence or validity of the
235		value or behavior. An application that relies on any particular value or behavior cannot be
236		assured to be portable across conforming implementations.

237 1.5 Portability

238 Some of the utilities in the Shell and Utilities volume of IEEE Std. 1003.1-200x and functions in
 239 the System Interfaces volume of IEEE Std. 1003.1-200x describe functionality that might not be
 240 fully portable to systems meeting the requirements for POSIX conformance (see the Base
 241 Definitions volume of IEEE Std. 1003.1-200x, Chapter 2, Conformance).

242 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in
 243 the margin identifies the nature of the option, extension, or warning (see Section 1.5.1). For
 244 maximum portability, an application should avoid such functionality.

245 Unless the primary task of a utility is to produce textual material on its standard output,
 246 application developers should not rely on the format or content of any such material that may be
 247 produced. Where the primary task *is* to provide such material, but the output format is
 248 incompletely specified, the description is marked with the OF margin code and shading.
 249 Application developers are warned not to expect that the output of such an interface on one
 250 system is any guide to its behavior on another system.

251 1.5.1 Codes

252 The codes and their meanings are as follows. See also Section 1.5.2 (on page 17).

253 ADV Advisory Information

254 The functionality described is optional. The functionality described is also an extension to the
 255 ISO C standard.

256 Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section.
 257 Where additional semantics apply to a function, the material is identified by use of the ADV
 258 margin legend.

259 AIO Asynchronous Input and Output

260 The functionality described is optional. The functionality described is also an extension to the
 261 ISO C standard.

262 Where applicable, functions are marked with the AIO margin legend in the SYNOPSIS section.
 263 Where additional semantics apply to a function, the material is identified by use of the AIO
 264 margin legend.

265 BAR Barriers

266 The functionality described is optional. The functionality described is also an extension to the
 267 ISO C standard.

268 Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.
 269 Where additional semantics apply to a function, the material is identified by use of the BAR
 270 margin legend.

271 BE Batch Environment Services and Utilities

272 The functionality described is optional.

273 Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.
 274 Where additional semantics apply to a utility, the material is identified by use of the BE margin
 275 legend.

276 CD C-Language Development Utilities

277 The functionality described is optional.

278 Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.
 279 Where additional semantics apply to a utility, the material is identified by use of the CD margin
 280 legend.

281	CPT	Process CPU-Time Clocks
282		The functionality described is optional. The functionality described is also an extension to the
283		ISO C standard.
284		Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.
285		Where additional semantics apply to a function, the material is identified by use of the CPT
286		margin legend.
287	CS	Clock Selection
288		The functionality described is optional. The functionality described is also an extension to the
289		ISO C standard.
290		Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section.
291		Where additional semantics apply to a function, the material is identified by use of the CS
292		margin legend.
293	CX	Extension to the ISO C standard
294		The functionality described is an extension to the ISO C standard. Application writers may
295		make use of an extension as it is supported on all IEEE Std. 1003.1-200x-conforming systems.
296	FD	FORTTRAN Development Utilities
297		The functionality described is optional.
298		Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.
299		Where additional semantics apply to a utility, the material is identified by use of the FD margin
300		legend.
301	FR	FORTTRAN Runtime Utilities
302		The functionality described is optional.
303		Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.
304		Where additional semantics apply to a utility, the material is identified by use of the FR margin
305		legend.
306	FSC	File Synchronization
307		The functionality described is optional. The functionality described is also an extension to the
308		ISO C standard.
309		Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.
310		Where additional semantics apply to a function, the material is identified by use of the FSC
311		margin legend.
312	IP6	IPV6
313		The functionality described is optional. The functionality described is also an extension to the
314		ISO C standard.
315		Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.
316		Where additional semantics apply to a function, the material is identified by use of the IP6
317		margin legend.
318	MAN	Mandatory in the Next Draft
319		This is an interim draft code used to aid reviewers during the development of
320		IEEE Std. 1003.1-200x. It denotes a feature that was previously an option or extension that is
321		being brought into the mandatory base functionality. This margin code will be removed from the
322		final draft.
323	MF	Memory Mapped Files
324		The functionality described is optional. The functionality described is also an extension to the
325		ISO C standard.

326 Where applicable, functions are marked with the MF margin legend in the SYNOPSIS section.
327 Where additional semantics apply to a function, the material is identified by use of the MF
328 margin legend.

329 ML **Process Memory Locking**
330 The functionality described is optional. The functionality described is also an extension to the
331 ISO C standard.

332 Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.
333 Where additional semantics apply to a function, the material is identified by use of the ML
334 margin legend.

335 MLR **Range Memory Locking**
336 The functionality described is optional. The functionality described is also an extension to the
337 ISO C standard.

338 Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
339 Where additional semantics apply to a function, the material is identified by use of the MLR
340 margin legend.

341 MON **Monotonic Clock**
342 The functionality described is optional. The functionality described is also an extension to the
343 ISO C standard.

344 Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.
345 Where additional semantics apply to a function, the material is identified by use of the MON
346 margin legend.

347 MPR **Memory Protection**
348 The functionality described is optional. The functionality described is also an extension to the
349 ISO C standard.

350 Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section.
351 Where additional semantics apply to a function, the material is identified by use of the MPR
352 margin legend.

353 MSG **Message Passing**
354 The functionality described is optional. The functionality described is also an extension to the
355 ISO C standard.

356 Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.
357 Where additional semantics apply to a function, the material is identified by use of the MSG
358 margin legend.

359 OB **Obsolescent**
360 The functionality described may be withdrawn in a future version of this volume of
361 IEEE Std. 1003.1-200x. Strictly Conforming POSIX Applications and Strictly Conforming XSI
362 Applications shall not use obsolescent features.

363 OF **Output Format Incompletely Specified**
364 The functionality described is an XSI extension. The format of the output produced by the utility
365 is not fully specified. It is therefore not possible to post-process this output in a consistent
366 fashion. Typical problems include unknown length of strings and unspecified field delimiters.

367 OH **Optional Header**
368 In the SYNOPSIS section of some interfaces in the System Interfaces volume of
369 IEEE Std. 1003.1-200x an included header is marked as in the following example:


```

370 OH      #include <sys/types.h>
371         #include <grp.h>
372         struct group *getgrnam(const char *name);

```

373 This indicates that the marked header is not required on XSI-conformant systems.

374 PIO **Prioritized Input and Output**
375 The functionality described is optional. The functionality described is also an extension to the
376 ISO C standard.

377 Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.
378 Where additional semantics apply to a function, the material is identified by use of the PIO
379 margin legend.

380 PS **Process Scheduling**
381 The functionality described is optional. The functionality described is also an extension to the
382 ISO C standard.

383 Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.
384 Where additional semantics apply to a function, the material is identified by use of the PS
385 margin legend.

386 RTS **Realtime Signals Extension**
387 The functionality described is optional. The functionality described is also an extension to the
388 ISO C standard.

389 Where applicable, functions are marked with the RTS margin legend in the SYNOPSIS section.
390 Where additional semantics apply to a function, the material is identified by use of the RTS
391 margin legend.

392 SD **Software Development Utilities**
393 The functionality described is optional.

394 Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.
395 Where additional semantics apply to a utility, the material is identified by use of the SD margin
396 legend.

397 SEM **Semaphores**
398 The functionality described is optional. The functionality described is also an extension to the
399 ISO C standard.

400 Where applicable, functions are marked with the SEM margin legend in the SYNOPSIS section.
401 Where additional semantics apply to a function, the material is identified by use of the SEM
402 margin legend.

403 SHM **Shared Memory Objects**
404 The functionality described is optional. The functionality described is also an extension to the
405 ISO C standard.

406 Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.
407 Where additional semantics apply to a function, the material is identified by use of the SHM
408 margin legend.

409 SIO **Synchronized Input and Output**
410 The functionality described is optional. The functionality described is also an extension to the
411 ISO C standard.

412 Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.
413 Where additional semantics apply to a function, the material is identified by use of the SIO
414 margin legend.

415	SPI	Spin Locks
416		The functionality described is optional. The functionality described is also an extension to the
417		ISO C standard.
418		Where applicable, functions are marked with the SPI margin legend in the SYNOPSIS section.
419		Where additional semantics apply to a function, the material is identified by use of the SPI
420		margin legend.
421	SPN	Spawn
422		The functionality described is optional. The functionality described is also an extension to the
423		ISO C standard.
424		Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.
425		Where additional semantics apply to a function, the material is identified by use of the SPN
426		margin legend.
427	SS	Process Sporadic Server
428		The functionality described is optional. The functionality described is also an extension to the
429		ISO C standard.
430		Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.
431		Where additional semantics apply to a function, the material is identified by use of the SS
432		margin legend.
433	TCT	Thread CPU-Time Clocks
434		The functionality described is optional. The functionality described is also an extension to the
435		ISO C standard.
436		Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.
437		Where additional semantics apply to a function, the material is identified by use of the TCT
438		margin legend.
439	THR	Threads
440		The functionality described is optional. The functionality described is also an extension to the
441		ISO C standard.
442		Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section.
443		Where additional semantics apply to a function, the material is identified by use of the THR
444		margin legend.
445	TMO	Timeouts
446		The functionality described is optional. The functionality described is also an extension to the
447		ISO C standard.
448		Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section.
449		Where additional semantics apply to a function, the material is identified by use of the TMO
450		margin legend.
451	TMR	Timers
452		The functionality described is optional. The functionality described is also an extension to the
453		ISO C standard.
454		Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section.
455		Where additional semantics apply to a function, the material is identified by use of the TMR
456		margin legend.
457	TPI	Threads Priority Inheritance
458		The functionality described is optional. The functionality described is also an extension to the
459		ISO C standard.

460 Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
461 Where additional semantics apply to a function, the material is identified by use of the TPI
462 margin legend.

463 TPP **Thread Priority Protection**

464 The functionality described is optional. The functionality described is also an extension to the
465 ISO C standard.

466 Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
467 Where additional semantics apply to a function, the material is identified by use of the TPP
468 margin legend.

469 TPS **Thread Execution Scheduling**

470 The functionality described is optional. The functionality described is also an extension to the
471 ISO C standard.

472 Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
473 Where additional semantics apply to a function, the material is identified by use of the TPS
474 margin legend.

475 TRC **Trace**

476 The functionality described is optional. The functionality described is also an extension to the
477 ISO C standard.

478 Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
479 Where additional semantics apply to a function, the material is identified by use of the TRC
480 margin legend.

481 TEF **Trace Event Filter**

482 The functionality described is optional. The functionality described is also an extension to the
483 ISO C standard.

484 Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.
485 Where additional semantics apply to a function, the material is identified by use of the TEF
486 margin legend.

487 TRL **Trace Log**

488 The functionality described is optional. The functionality described is also an extension to the
489 ISO C standard.

490 Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
491 Where additional semantics apply to a function, the material is identified by use of the TRL
492 margin legend.

493 TRI **Trace Inherit**

494 The functionality described is optional. The functionality described is also an extension to the
495 ISO C standard.

496 Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
497 Where additional semantics apply to a function, the material is identified by use of the TRI
498 margin legend.

499 TSA **Thread Stack Address Attribute**

500 The functionality described is optional. The functionality described is also an extension to the
501 ISO C standard.

502 Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
503 Where additional semantics apply to a function, the material is identified by use of the TSA
504 margin legend.

505	TSF	Thread-Safe Functions
506		The functionality described is optional. The functionality described is also an extension to the
507		ISO C standard.
508		Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section.
509		Where additional semantics apply to a function, the material is identified by use of the TSF
510		margin legend.
511	TSH	Thread Process-Shared Synchronization
512		The functionality described is optional. The functionality described is also an extension to the
513		ISO C standard.
514		Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.
515		Where additional semantics apply to a function, the material is identified by use of the TSH
516		margin legend.
517	TSP	Thread Sporadic Server
518		The functionality described is optional. The functionality described is also an extension to the
519		ISO C standard.
520		Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.
521		Where additional semantics apply to a function, the material is identified by use of the TSP
522		margin legend.
523	TSS	Thread Stack Address Size
524		The functionality described is optional. The functionality described is also an extension to the
525		ISO C standard.
526		Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.
527		Where additional semantics apply to a function, the material is identified by use of the TSS
528		margin legend.
529	TYM	Typed Memory Objects
530		The functionality described is optional. The functionality described is also an extension to the
531		ISO C standard.
532		Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.
533		Where additional semantics apply to a function, the material is identified by use of the TYM
534		margin legend.
535	UN	Possibly Unsupportable Feature
536		The functionality described is an XSI extension. It need not be possible to implement the
537		required functionality (as defined) on all conformant systems and the functionality need not be
538		present. This may, for example, be the case where the conformant system is hosted and the
539		underlying system provides the service in an alternative way.
540	UP	User Portability Utilities
541		The functionality described is optional.
542		Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.
543		Where additional semantics apply to a utility, the material is identified by use of the UP margin
544		legend.
545	XSI	Extension
546		The functionality described is an XSI extension. Functionality marked XSI is also an extension to
547		the ISO C standard. Application writers may confidently make use of an extension on all
548		systems supporting the X/Open System Interfaces Extension.

549 If an entire SYNOPSIS section is shaded and marked with one XSI, all the functionality described
 550 in that reference page is an extension. See Section 3.441 (on page 117).

551 XSR **XSI STREAMS**
 552 The functionality described is optional. The functionality described is also an extension to the
 553 ISO C standard.

554 Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section.
 555 Where additional semantics apply to a function, the material is identified by use of the XSR
 556 margin legend.

557 1.5.2 Margin Code Notation

558 Some of the functionality described in IEEE Std. 1003.1-200x depends on support of more than
 559 one option, or independently may depend on several options. The following notation for margin
 560 codes is used to denote the following cases:

- 561 • A feature dependent on one or two options.

562 In this case, margin codes have a <space> separator; for example:

563 MF This feature requires support for only the Memory Mapped Files option.

564 MF SHM This feature requires support for both the Memory Mapped Files and the Shared Memory
 565 Objects options; that is, an application which uses this feature is portable only between
 566 implementations that provide both options.

- 567 • A feature dependent on either of the options denoted.

568 In this case, margin codes have a ' | ' separator to denote the logical OR; for example:

569 MF|SHM This feature is dependent on support for either the Memory Mapped Files option or the
 570 Shared Memory Objects option; that is, an application which uses this feature is portable
 571 between implementations that provide any (or all) of the options.

- 572 • A feature dependent on more than two options.

573 The following special notations are used:

574 code1 ADV (MF | SHM) This feature requires support of the Advisory Informtion option and either
 575 the Memory Mapped Files or Shared Memory Objects option.

576 code2 MF|SHM|MPR This feature requires support of either the Memory Mapped Files, Shared
 577 Memory Objects, or Memory Protection options.

578 Where large sections of text are dependent on support for an option, a lead-in text block is
 579 provided and shaded accordingly; for example:

580 TRC This section describes extensions to support tracing of user applications. This functionality is
 581 dependent on support of the Trace option (and the rest of this section is not further shaded for
 582 this option).

584 2.1 Implementation Conformance

585 2.1.1 Requirements

586 A *conforming implementation* shall meet all of the following criteria:

- 587 1. The system shall support all utilities, functions, and facilities defined within
588 IEEE Std. 1003.1-200x that are required for POSIX conformance (see Section 2.1.3 (on page
589 20)). These interfaces shall support the functional behavior described herein.
- 590 2. The system may support one or more options as described under Section 2.1.5 (on page
591 25). When an implementation claims that an option is supported, all of its constituent
592 parts shall be provided.
- 593 3. The system may support the X/Open System Interface Extension (XSI) as described under
594 Section 2.1.4 (on page 23).
- 595 4. The system may provide additional utilities, functions, or facilities not required by
596 IEEE Std. 1003.1-200x. Non-standard extensions of the utilities, functions, or facilities
597 specified in IEEE Std. 1003.1-200x should be identified as such in the system
598 documentation. Non-standard extensions, when used, may change the behavior of utilities,
599 functions, or facilities defined by IEEE Std. 1003.1-200x. The conformance document shall
600 define an environment in which an application can be run with the behavior specified by
601 IEEE Std. 1003.1-200x. In no case shall such an environment require modification of a
602 Strictly Conforming POSIX Application (see Section 2.2.1 (on page 38)).

603 2.1.2 Documentation

604 A conformance document with the following information shall be available for an
605 implementation claiming conformance to IEEE Std. 1003.1-200x. The conformance document
606 shall have the same structure as IEEE Std. 1003.1-200x, with the information presented in the
607 appropriate sections and subsections. Sections and subsections that consist solely of subordinate
608 section titles, with no other information, are not required. The conformance document shall not
609 contain information about extended facilities or capabilities outside the scope of
610 IEEE Std. 1003.1-200x.

611 The conformance document shall contain a statement that indicates the full name, number, and
612 date of the standard that applies. The conformance document may also list international
613 software standards that are available for use by a Conforming POSIX Application. Applicable
614 characteristics where documentation is required by one of these standards, or by standards of
615 government bodies, may also be included.

616 The conformance document shall describe the limit values found in the headers `<limits.h>` (on
617 page 281) and `<unistd.h>` (on page 437), stating values, the conditions under which those values
618 may change, and the limits of such variations, if any.

619 The conformance document shall describe the behavior of the implementation for all
620 implementation-defined features defined in IEEE Std. 1003.1-200x. This requirement shall be met
621 by listing these features and providing either a specific reference to the system documentation or
622 providing full syntax and semantics of these features. When the value or behavior in the

623 implementation is designed to be variable or customized on each instantiation of the system, the
624 implementation provider shall document the nature and permissible ranges of this variation.

625 The conformance document may specify the behavior of the implementation for those features
626 where IEEE Std. 1003.1-200x states that implementations may vary or where features are
627 identified as undefined or unspecified.

628 The conformance document shall not contain documentation other than that specified in the
629 preceding paragraph except where such documentation is specifically allowed or required by
630 other provisions of IEEE Std. 1003.1-200x.

631 The phrases “shall document” or “shall be documented” in IEEE Std. 1003.1-200x mean that
632 documentation of the feature shall appear in the conformance document, as described
633 previously, unless there is an explicit reference in the conformance document to show where the
634 information can be found in the system documentation.

635 The system documentation should also contain the information found in the conformance
636 document.

637 **2.1.3 POSIX Conformance**

638 A conforming implementation shall meet the following criteria for POSIX conformance. |

639 *2.1.3.1 POSIX System Interfaces*

- 640 • The system shall set the symbolic constant `_POSIX_BASE` to a value other than `-1`. |
- 641 • The system shall support the following symbolic constants, reflecting mandatory Profiling
642 Option Groups for IEEE Std. 1003.1-200x (see Section 2.1.5 (on page 25)):
 - 643 — `_POSIX_C_LANG_SUPPORT`
 - 644 — `_POSIX_DEVICE_IO`
 - 645 — `_POSIX_DEVICE_SPECIFIC`
 - 646 — `_POSIX_FD_MGMT`
 - 647 — `_POSIX_FIFO`
 - 648 — `_POSIX_FILE_ATTRIBUTES`
 - 649 — `_POSIX_FILE_SYSTEM`
 - 650 — `_POSIX_JOB_CONTROL`
 - 651 — `_POSIX_MULTIPLE_PROCESS`
 - 652 — `_POSIX_PIPE`
 - 653 — `_POSIX_SIGNALS`
 - 654 — `_POSIX_SINGLE_PROCESS`
 - 655 — `_POSIX_SYSTEM_DATABASE`
 - 656 — `_POSIX_USER_GROUPS`
 - 657 — `_POSIX_NETWORKING`
- 658 • The system may support one or more Profiling Option Groups (see Section 2.1.5.1 (on page
659 25)) denoted by the following symbolic constants:

- 660 — `_POSIX_C_LANG_SUPPORT_R`
- 661 — `_POSIX_FILE_LOCKING`
- 662 — `_POSIX_SYSTEM_DATABASE_R`
- 663 — `_POSIX_USER_GROUPS_R`
- 664 • Although all implementations conforming to IEEE Std. 1003.1-200x support all the features
 665 described below, there may be system-dependent or file system-dependent configuration
 666 procedures that can remove or modify any or all of these features. Such configurations
 667 should not be made if strict compliance is required.
- 668 The following symbolic constants shall either be undefined or defined with a value other
 669 than `-1`. If a constant is undefined, an application should use the `sysconf()`, `pathconf()`, or
 670 `fpathconf()` functions, or the `getconf` utility, to determine which features are present on the
 671 system at that time or for the particular path name in question.
- 672 — `_POSIX_CHOWN_RESTRICTED`
- 673 The use of `chown()` is restricted to a process with appropriate privileges, and to changing
 674 the group ID of a file only to the effective group ID of the process or to one of its
 675 supplementary group IDs.
- 676 — `_POSIX_NO_TRUNC`
- 677 Path name components longer than `{NAME_MAX}` generate an error.
- 678 • The following symbolic constants shall be defined with a value other than `-1`:
- 679 — `_POSIX_JOB_CONTROL` |
- 680 — `_POSIX_SAVED_IDS` |
- 681 — `_POSIX_VDISABLE` |
- 682 **Note:** The symbols above represent historical options that are no longer allowed as |
 683 options, but are retained here for backwards-compatibility of applications. |
- 684 • The system may support one or more options (see Section 2.1.6 (on page 32)) denoted by the |
 685 following symbolic constants: |
- 686 — `_POSIX_ADVISORY_INFO`
- 687 — `_POSIX_ASYNCHRONOUS_IO`
- 688 — `_POSIX_BARRIERS`
- 689 — `_POSIX_CLOCK_SELECTION`
- 690 — `_POSIX_CPUTIME`
- 691 — `_POSIX_FSYNC`
- 692 — `_POSIX_IPV6`
- 693 — `_POSIX_MAPPED_FILES`
- 694 — `_POSIX_MEMLOCK`
- 695 — `_POSIX_MEMLOCK_RANGE`
- 696 — `_POSIX_MEMORY_PROTECTION`
- 697 — `_POSIX_MESSAGE_PASSING`

698	—	_POSIX_MONOTONIC_CLOCK	
699	—	_POSIX_PRIORITIZED_IO	
700	—	_POSIX_PRIORITY_SCHEDULING	
701	—	_POSIX_RAW_SOCKETS	
702	—	_POSIX_REALTIME_SIGNALS	
703	—	_POSIX_SEMAPHORES	
704	—	_POSIX_SHARED_MEMORY_OBJECTS	
705	—	_POSIX_SPAWN	
706	—	_POSIX_SPIN_LOCKS	
707	—	_POSIX_SPORADIC_SERVER	
708	—	_POSIX_SYNCHRONIZED_IO	
709	—	_POSIX_THREAD_ATTR_STACKADDR	
710	—	_POSIX_THREAD_CPUTIME	
711	—	_POSIX_THREAD_ATTR_STACKSIZE	
712	—	_POSIX_THREAD_PRIO_INHERIT	
713	—	_POSIX_THREAD_PRIO_PROTECT	
714	—	_POSIX_THREAD_PRIORITY_SCHEDULING	
715	—	_POSIX_THREAD_PROCESS_SHARED	
716	—	_POSIX_THREAD_SAFE_FUNCTIONS	
717	—	_POSIX_THREAD_SPARADIC_SERVER	
718	—	_POSIX_THREADS	
719	—	_POSIX_TIMEOUTS	
720	—	_POSIX_TIMERS	
721	—	_POSIX_TRACE	
722	—	_POSIX_TRACE_EVENT_FILTER	
723	—	_POSIX_TRACE_INHERIT	
724	—	_POSIX_TRACE_LOG	
725	—	_POSIX_TYPED_MEMORY_OBJECTS	
726		If any of the symbolic constants <code>_POSIX_TRACE_EVENT_FILTER</code> , <code>_POSIX_TRACE_LOG</code> , or	
727		<code>_POSIX_TRACE_INHERIT</code> is defined to have a value other than <code>-1</code> , then the symbolic	
728		constant <code>_POSIX_TRACE</code> shall also be defined to have a value other than <code>-1</code> .	
729	XSI	• The system may support the XSI extensions (see Section 2.1.5.2 (on page 27)) denoted by the	
730		following symbolic constants:	
731		— <code>_XOPEN_CRYPT</code>	
732		— <code>_XOPEN_LEGACY</code>	

733 — `_XOPEN_REALTIME`

734 — `_XOPEN_REALTIME_THREADS`

735 — `_XOPEN_UNIX`

736 2.1.3.2 *POSIX Shell and Utilities*

737 • The system shall provide all the mandatory utilities in the Shell and Utilities volume of
738 IEEE Std. 1003.1-200x with all the functional behavior described therein.

739 • The system shall support the Large File capabilities described in the Shell and Utilities
740 volume of IEEE Std. 1003.1-200x.

741 • The system may support one or more options (see Section 2.1.6 (on page 32)) denoted by the
742 following symbolic constants. (The literal names below apply to the *getconf* utility.)

743 — `POSIX2_C_DEV`

744 — `POSIX2_CHAR_TERM`

745 — `POSIX2_FORT_DEV`

746 — `POSIX2_FORT_RUN`

747 — `POSIX2_LOCALEDEF`

748 — `POSIX2_PBS`

749 — `POSIX2_PBS_ACCOUNTING`

750 — `POSIX2_PBS_LOCATE`

751 — `POSIX2_PBS_MESSAGE`

752 — `POSIX2_PBS_TRACK`

753 — `POSIX2_SW_DEV`

754 — `POSIX2_UPE`

755 • The system may support the XSI extensions (see Section 2.1.4).

756 Additional language bindings and development utility options may be provided in other related
757 standards or in a future version of IEEE Std. 1003.1-200x. In the former case, additional symbolic
758 constants of the same general form as shown in this subsection should be defined by the related
759 standard document and made available to the application without requiring
760 IEEE Std. 1003.1-200x to be updated.

761 2.1.4 **XSI Conformance**

762 XSI IEEE Std. 1003.1-200x describes utilities, functions, and facilities offered to application programs
763 by the X/Open System Interface (XSI). An XSI-conforming implementation shall meet the
764 criteria for POSIX conformance and the following requirements.

765 2.1.4.1 *XSI System Interfaces*

766 • The system shall support all the functions and headers defined in IEEE Std. 1003.1-200x as
767 part of the XSI extension denoted by the symbolic constant `_XOPEN_UNIX` and any
768 extensions marked with the XSI extension marking (see Section 1.5.1 (on page 10)).

769 • The system shall support the *mmap()*, *munmap()*, and *msync()* functions.

- 770 • The system shall support the following options defined within IEEE Std. 1003.1-200x (see
771 Section 2.1.6 (on page 32)):
- 772 — `_POSIX_FSYNC`
- 773 — `_POSIX_MAPPED_FILES`
- 774 — `_POSIX_MEMORY_PROTECTION`
- 775 — `_POSIX_THREAD_ATTR_STACKADDR`
- 776 — `_POSIX_THREAD_ATTR_STACKSIZE`
- 777 — `_POSIX_THREAD_PROCESS_SHARED`
- 778 — `_POSIX_THREAD_SAFE_FUNCTIONS`
- 779 — `_POSIX_THREADS`
- 780 • The system shall support the following Profiling Option Groups (see Section 2.1.5.1 (on page
781 25)) defined within IEEE Std. 1003.1-200x:
- 782 — `_POSIX_C_LANG_SUPPORT_R`
- 783 — `_POSIX_FILE_LOCKING`
- 784 — `_POSIX_SYSTEM_DATABASE_R`
- 785 — `_POSIX_USER_GROUPS_R`
- 786 • The system may support the following XSI Option Groups (see Section 2.1.5.2 (on page 27))
787 defined within IEEE Std. 1003.1-200x:
- 788 — `_XOPEN_CRYPT`
- 789 — `_XOPEN_LEGACY`
- 790 — `_XOPEN_REALTIME`
- 791 — `_XOPEN_REALTIME_THREADS`

792 2.1.4.2 XSI Shell and Utilities Conformance

- 793 • The system shall support all the utilities defined in the Shell and Utilities volume of
794 IEEE Std. 1003.1-200x as part of the XSI extension denoted by the XSI marking in the
795 SYNOPSIS section, and any extensions marked with the XSI extension marking (see Section
796 1.5.1 (on page 10)) within the text.
- 797 • The system shall support the User Portability Utilities option.
- 798 • The system shall support creation of locales (see Chapter 7 (on page 143)).
- 799 • The C-language Development utility *c99* shall be supported.
- 800 • The XSI Development Utilities option may be supported. It consists of the following software
801 development utilities:
- | | | | | | | | |
|-----|--------------|--------------|--------------|-------------|--|--|--|
| 802 | <i>admin</i> | <i>delta</i> | <i>rmdel</i> | <i>val</i> | | | |
| 803 | <i>cflow</i> | <i>get</i> | <i>sact</i> | <i>what</i> | | | |
| 804 | <i>ctags</i> | <i>m4</i> | <i>sccs</i> | | | | |
| 805 | <i>cxref</i> | <i>prs</i> | <i>unget</i> | | | | |
- 806 • Within the utilities that are provided, functionality marked by the codes OF, OP, PI, or UN
807 (see Section 1.5.1 (on page 10)) need not be provided.
808

809 **2.1.5 Option Groups**

810 An Option Group is a group of related functions or options defined within the System Interfaces
811 volume of IEEE Std. 1003.1-200x.

812 If an implementation supports an Option Group, then the system shall support the functional
813 behavior described herein.

814 If an implementation does not support an Option Group, then the system need not support the
815 functional behavior described herein.

816 **2.1.5.1 Profiling Option Groups**

817 The following Option Groups are defined to support profiling. These Option Groups allow
818 profiles to subset the System Interfaces volume of IEEE Std. 1003.1-200x by defining sets of
819 functions, denoted by the following symbolic constants:

820 `_POSIX_C_LANG_SUPPORT`: General C Library Support

821 `abs()`, `acos()`, `asctime()`, `asin()`, `atan()`, `atan2()`, `atof()`, `atoi()`, `atol()`, `bsearch()`, `calloc()`, `ceil()`,
822 `cos()`, `cosh()`, `ctime()`, `exp()`, `fabs()`, `floor()`, `fmod()`, `free()`, `frexp()`, `gmtime()`, `idexp()`, `isalnum()`,
823 `isalpha()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `ispunct()`, `isspace()`, `isupper()`,
824 `isxdigit()`, `localtime()`, `log()`, `log10()`, `longjmp()`, `malloc()`, `mktime()`, `modf()`, `pow()`, `qsort()`,
825 `rand()`, `realloc()`, `setjmp()`, `sin()`, `sinh()`, `sqrt()`, `srand()`, `strcat()`, `strchr()`, `strcmp()`, `strcpy()`,
826 `strcspn()`, `strlen()`, `strncat()`, `strncmp()`, `strncpy()`, `strpbkr()`, `strrchr()`, `strspn()`, `strstr()`,
827 `strtok()`, `tan()`, `tanh()`, `tolower()`, `toupper()`

828 `_POSIX_C_LANG_SUPPORT_R`: Thread-Safe C-Language Support

829 `asctime_r()`, `ctime_r()`, `gmtime_r()`, `localtime_r()`, `readdir_r()`, `rand_r()`, `strtok_r()`

830 `_POSIX_DEVICE_IO`: Device Input and Output

831 `close()`, `clearerr()`, `getc()`, `getchar()`, `gets()`, `fclose()`, `fdopen()`, `feof()`, `ferror()`, `fflush()`, `fgetc()`,
832 `fgets()`, `fileno()`, `fopen()`, `fprintf()`, `fputc()`, `fputs()`, `fread()`, `freopen()`, `fscanf()`, `fwrite()`, `open()`,
833 `perror()`, `printf()`, `putc()`, `putchar()`, `puts()`, `read()`, `sprintf()`, `scanf()`, `sscanf()`, `setbuf()`,
834 `ungetc()`, `write()`

835 `_POSIX_DEVICE_SPECIFIC`: General Terminal Interface

836 `cfgetospeed()`, `cfsetispeed()`, `cfsetospeed()`, `ctermid()`, `isatty()`, `tcgetattr()`, `tcsetattr()`,
837 `tcsendbreak()`, `tcdrain()`, `tclflush()`, `tclflow()`, `ttyname()`

838 `_POSIX_DEVICE_SPECIFIC_R`: Thread-Safe General Terminal Interface

839 `ttyname_r()`

840 `_POSIX_FD_MGMT`: File Descriptor

841 `dup()`, `dup2()`, `fcntl()`, `fseek()`, `ftell()`, `lseek()`, `rewind()`

842 `_POSIX_FIFO`: FIFO

843 `mkfifo()`

844 `_POSIX_FILE_ATTRIBUTES`: File Attributes

845 `chmod()`, `chown()`, `umask()`

846 `_POSIX_FILE_LOCKING`: Thread-Safe Stdio Locking

847 `flockfile()`, `ftrylockfile()`, `funlockfile()`, `getc_unlocked()`, `getchar_unlocked()`, `putc_unlocked()`,
848 `putchar_unlocked()`

849 `_POSIX_FILE_SYSTEM`: File System

850 `access()`, `chdir()`, `closedir()`, `creat()`, `fpathconf()`, `fstat()`, `getcwd()`, `link()`, `mkdir()`, `opendir()`,
851 `pathconf()`, `readdir()`, `remove()`, `rename()`, `rewinddir()`, `rmdir()`, `stat()`, `tmpfile()`, `tmpnam()`,
852 `unlink()`, `utime()`

853 _POSIX_JOB_CONTROL: Job Control
854 *setpgid()*, *tcgetpgrp()*, *tcsetpgrp()*

855 _POSIX_MULTIPLE_PROCESS: Multiple Process
856 *_exit()*, *assert()*, *exit()*, *execl()*, *execle()*, *execlp()*, *execv()*, *execve()*, *execvp()*, *fork()*, *getenv()*,
857 *getpid()*, *getppid()*, *setlocale()*, *sleep()*, *times()*, *wait()*, *waitpid()*

858 _POSIX_NETWORKING: Networking
859 *accept()*, *bind()*, *connect()*, *endhostent()*, *endnetent()*, *endprotoent()*, *endservent()*, *getaddrinfo()*,
860 *gethostbyaddr()*, *gethostbyname()*, *gethostent()*, *gethostname()*, *getipnodebyaddr()*,
861 *getipnodebyname()*, *getnameinfo()*, *getnetbyaddr()*, *getnetbyname()*, *getnetent()*, *getpeername()*,
862 *getprotobyname()*, *getprotobynumber()*, *getprotoent()*, *getservbyname()*, *getservbyport()*,
863 *getservent()*, *getsockname()*, *getsockopt()*, *htonl()*, *htons()*, *if_freenameindex()*, *if_indextoname()*,
864 *if_nameindex()*, *if_nametoindex()*, *inet_addr()*, *inet_lnaof()*, *inet_makeaddr()*, *inet_netof()*,
865 *inet_network()*, *inet_ntoa()*, *listen()*, *ntohl()*, *ntohs()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*,
866 *sendmsg()*, *sendto()*, *sethostent()*, *setnetent()*, *setprotoent()*, *setservent()*, *setsockopt()*,
867 *shutdown()*, *socket()*, *socketpair()*

868 _POSIX_PIPE: Pipe
869 *pipe()*

870 _POSIX_SIGNALS: Signal
871 *abort()*, *alarm()*, *kill()*, *pause()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*,
872 *sigismember()*, *siglongjmp()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigsetjmp()*

873 _POSIX_SINGLE_PROCESS: Single Process
874 *sysconf()*, *time()*, *uname()*

875 _POSIX_SYSTEM_DATABASE: System Database
876 *getgrgid()*, *getgrnam()*, *getpwnam()*, *getpwuid()*

877 _POSIX_SYSTEM_DATABASE_R: Thread-Safe System Database
878 *getgrgid_r()*, *getgrnam_r()*, *getpwuid_r()*, *getpwnam_r()*

879 _POSIX_USER_GROUPS: User and Group
880 *geteuid()*, *getegid()*, *getgid()*, *getgroups()*, *getlogin()*, *getpgrp()*, *getuid()*, *setgid()*, *setuid()*,
881 *setuid()*

882 _POSIX_USER_GROUPS_R: Thread-Safe User and Group
883 *getlogin_r()*

884 Many of these profiling option groups provide basic system functionality that other profiling
885 option groups and options depend upon.⁴ All of the mandatory profiling option groups (listed in
886 Section 2.1.3.1 (on page 20)) shall be supported by an implementation conforming to
887 IEEE Std. 1003.1-200x. If a profile of IEEE Std. 1003.1-200x does not require an implementation to
888 provide all of the mandatory profiling option groups or does not require an implementation to
889 provide an option or profiling option group that provides features required by another option or
890 profiling option group,⁵ the profile shall specify⁶ all of the following:

- 891
- 892 4. As an example, the File System profiling option group provides underlying support for path name resolution and file creation
893 which are needed by any interface in IEEE Std. 1003.1-200x that parses a *path* argument. If a profile requires support for the
894 Device Input and Output profiling option group but does not require support for the File System profiling option group, the
895 profile must specify how path name resolution is to behave in that profile, how the *O_CREAT* flag to *open()* is to be handled (and
896 the use of the character 'a' in the *mode* argument of *fopen()* when a file name argument names a file that does not exist), and
897 specify lots of other details.
- 898 5. As an example, IEEE Std. 1003.1-200x requires that implementations claiming to support the Range Memory Locking option also
899 support the Process Memory Locking option. A profile could require that the Range Memory Locking option had to be supplied
900 without requiring that the Process Memory Locking option be supplied as long as the profile specifies everything an application
901 writer or system implementor would have to know to build an application or implementation conforming to the profile.

- 902 • Restricted or altered behavior of interfaces defined by IEEE Std. 1003.1-200x that may differ
903 on an implementation of the profile
 - 904 • Additional behaviors that may produce undefined or unspecified results
 - 905 • Additional implementation-defined behavior that implementations shall be required to
906 document in the profile’s conformance document
- 907 if any of the above is a result of the profile not providing an interface required by
908 IEEE Std. 1003.1-200x.

909 **2.1.5.2 XSI Option Groups**

910 XSI The following Option Groups are defined to support the definition of XSI conformance within
911 the System Interfaces volume of IEEE Std. 1003.1-200x:

912 **Encryption**

913 The Encryption Option Group is denoted by the symbolic constant `_XOPEN_CRYPT`. It includes
914 the following functions:

915 `crypt()`, `encrypt()`, `setkey()`

916 These functions are marked `CRYPT`.

917 Due to U.S. Government export restrictions on the decoding algorithm, implementations are
918 restricted in making these functions available. All the functions in the Encryption Option Group
919 may therefore return `[ENOSYS]` or, alternatively, `encrypt()` shall return `[ENOSYS]` for the
920 decryption operation.

921 An implementation that claims conformance to this Option Group shall set the symbolic
922 constant `_XOPEN_CRYPT` to a value other than `-1`.

923 **Realtime**

924 The Realtime Option Group is denoted by the symbolic constant `_XOPEN_REALTIME`.

925 This Option Group includes a set of realtime functions drawn from options within
926 IEEE Std. 1003.1-200x (see Section 2.1.6 (on page 32)).

927 Where entire functions are included in the Option Group, the NAME section is marked with
928 `REALTIME`. Where additional semantics have been added to existing pages, the new material is
929 identified by use of the appropriate margin legend for the underlying option defined within
930 IEEE Std. 1003.1-200x.

931 An implementation that claims conformance to this Option Group shall set the symbolic
932 constant `_XOPEN_REALTIME` to a value other than `-1`.

933 This Option Group consists of the set of the following options from within IEEE Std. 1003.1-200x
934 (see Section 2.1.6 (on page 32)):

935 _____

936 6. Note that the profile could just specify that any use of the features not specified by the profile would produce undefined or
937 unspecified results.

938 _POSIX_ASYNCHRONOUS_IO
 939 _POSIX_FSYNC
 940 _POSIX_MAPPED_FILES
 941 _POSIX_MEMLOCK
 942 _POSIX_MEMLOCK_RANGE
 943 _POSIX_MEMORY_PROTECTION
 944 _POSIX_MESSAGE_PASSING
 945 _POSIX_PRIORITIZED_IO
 946 _POSIX_PRIORITY_SCHEDULING
 947 _POSIX_REALTIME_SIGNALS
 948 _POSIX_SEMAPHORES
 949 _POSIX_SHARED_MEMORY_OBJECTS
 950 _POSIX_SYNCHRONIZED_IO
 951 _POSIX_TIMERS

952 If the symbolic constant `_XOPEN_REALTIME` is defined to have a value other than `-1`, then the
 953 following symbolic constants shall be defined by the implementation to have the value
 954 `200ymmL`, the date of approval of IEEE Std. 1003.1-200x:

955 _POSIX_ASYNCHRONOUS_IO
 956 _POSIX_MEMLOCK
 957 _POSIX_MEMLOCK_RANGE
 958 _POSIX_MESSAGE_PASSING
 959 _POSIX_PRIORITY_SCHEDULING
 960 _POSIX_REALTIME_SIGNALS
 961 _POSIX_SEMAPHORES
 962 _POSIX_SHARED_MEMORY_OBJECTS
 963 _POSIX_SYNCHRONIZED_IO
 964 _POSIX_TIMERS

965 The functionality associated with `_POSIX_MAPPED_FILES`, `_POSIX_MEMORY_PROTECTION`,
 966 and `_POSIX_FSYNC` is always supported on XSI-conformant systems.

967 Support of `_POSIX_PRIORITIZED_IO` on XSI-conformant systems is optional. If this
 968 functionality is supported, then `_POSIX_PRIORITIZED_IO` shall be set to a value other than `-1`.
 969 Otherwise, it shall be undefined.

970 If `_POSIX_PRIORITIZED_IO` is supported, then asynchronous I/O operations performed by
 971 `aio_read()`, `aio_write()`, and `lio_listio()` shall be submitted at a priority equal to the scheduling
 972 priority of the process minus `aiocbp->aio_reqprio`. The implementation shall also document for
 973 which files I/O prioritization is supported.

974 **Advanced Realtime**

975 An implementation that claims conformance to this Option Group shall also support the
 976 Realtime Option Group.

977 Where entire functions are included in the Option Group, the NAME section is marked with
 978 ADVANCED REALTIME. Where additional semantics have been added to existing pages, the
 979 new material is identified by use of the appropriate margin legend for the underlying option
 980 defined within IEEE Std. 1003.1-200x.

981 This Option Group consists of the set of the following options from within IEEE Std. 1003.1-200x
 982 (see Section 2.1.6 (on page 32)):

983 _POSIX_ADVISORY_INFO
 984 _POSIX_CLOCK_SELECTION
 985 _POSIX_CPUTIME
 986 _POSIX_MONOTONIC_CLOCK
 987 _POSIX_SPAWN
 988 _POSIX_SPORADIC_SERVER
 989 _POSIX_TIMEOUTS
 990 _POSIX_TYPED_MEMORY_OBJECTS

991 If the implementation supports the Advanced Realtime Option Group, then the following
 992 symbolic constants shall be defined by the implementation to have the value 200ymmL, the date
 993 of approval of IEEE Std. 1003.1-200x:

994 _POSIX_ADVISORY_INFO
 995 _POSIX_CLOCK_SELECTION
 996 _POSIX_CPUTIME
 997 _POSIX_MONOTONIC_CLOCK
 998 _POSIX_SPAWN
 999 _POSIX_SPORADIC_SERVER
 1000 _POSIX_TIMEOUTS
 1001 _POSIX_TYPED_MEMORY_OBJECTS

1002 If the symbolic constant `_POSIX_SPORADIC_SERVER` is defined, then the symbolic constant
 1003 `_POSIX_PRIORITY_SCHEDULING` shall also be defined by the implementation to have the
 1004 value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.

1005 If the symbolic constant `_POSIX_CPUTIME` is defined, then the symbolic constant
 1006 `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200ymmL, the
 1007 date of approval of IEEE Std. 1003.1-200x.

1008 If the symbolic constant `_POSIX_MONOTONIC_CLOCK` is defined, then the symbolic constant
 1009 `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200ymmL, the
 1010 date of approval of IEEE Std. 1003.1-200x.

1011 If the symbolic constant `_POSIX_CLOCK_SELECTION` is defined, then the symbolic constant
 1012 `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200ymmL, the
 1013 date of approval of IEEE Std. 1003.1-200x.

1014 **Realtime Threads**

1015 The Realtime Threads Option Group is denoted by the symbolic constant
 1016 `_XOPEN_REALTIME_THREADS`.

1017 This Option Group consists of the set of the following options from within IEEE Std. 1003.1-200x
 1018 (see Section 2.1.6 (on page 32)):

1019 _POSIX_THREAD_PRIO_INHERIT
 1020 _POSIX_THREAD_PRIO_PROTECT
 1021 _POSIX_THREAD_PRIORITY_SCHEDULING

1022 Where applicable, whole pages are marked `REALTIME THREADS`, together with the
 1023 appropriate option margin legend for the SYNOPSIS section (see Section 1.5.1 (on page 10)).

1024 An implementation that claims conformance to this Option Group shall set
 1025 `_XOPEN_REALTIME_THREADS` to a value other than `-1`.

1026 If the symbol `_XOPEN_REALTIME_THREADS` is defined to have a value other than `-1`, then the
 1027 symbols shall also be defined by the implementation to have the value 200ymmL, the date of

1028	approval of IEEE Std. 1003.1-200x:	
1029	_POSIX_THREAD_PRIO_INHERIT	
1030	_POSIX_THREAD_PRIO_PROTECT	
1031	_POSIX_THREAD_PRIORITY_SCHEDULING	
1032	Advanced Realtime Threads	
1033	An implementation that claims conformance to this Option Group shall also support the	
1034	Realtime Threads Option Group.	
1035	Where entire functions are included in the Option Group, the NAME section is marked with	
1036	ADVANCED REALTIME THREADS. Where additional semantics have been added to existing	
1037	pages, the new material is identified by use of the appropriate margin legend for the underlying	
1038	option defined within IEEE Std. 1003.1-200x.	
1039	This Option Group consists of the set of the following options from within IEEE Std. 1003.1-200x	
1040	(see Section 2.1.6 (on page 32)):	
1041	_POSIX_BARRIERS	
1042	_POSIX_SPIN_LOCKS	
1043	_POSIX_THREAD_CPUTIME	
1044	_POSIX_THREAD_SPORADIC_SERVER	
1045	If the symbolic constant _POSIX_THREAD_SPORADIC_SERVER is defined, then the symbolic	
1046	constant _POSIX_THREAD_PRIORITY_SCHEDULING shall also be defined by the	
1047	implementation to have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.	
1048	If the symbolic constant _POSIX_THREAD_CPUTIME is defined, then the symbolic constant	
1049	_POSIX_TIMERS shall also be defined by the implementation to have the value 200ymmL, the	
1050	date of approval of IEEE Std. 1003.1-200x.	
1051	If the symbolic constant _POSIX_BARRIERS is defined, then the symbolic constants	
1052	_POSIX_THREADS and _POSIX_THREAD_SAFE_FUNCTIONS shall also be defined by the	
1053	implementation to have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.	
1054	If the symbolic constant _POSIX_SPIN_LOCKS is defined, then the symbolic constants	
1055	_POSIX_THREADS and _POSIX_THREAD_SAFE_FUNCTIONS shall also be defined by the	
1056	implementation to have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.	
1057	If the implementation supports the Advanced Realtime Threads Option Group, then the	
1058	following symbolic constants shall be defined by the implementation to have the value	
1059	200ymmL, the date of approval of IEEE Std. 1003.1-200x:	
1060	_POSIX_BARRIERS	
1061	_POSIX_SPIN_LOCKS	
1062	_POSIX_THREAD_CPUTIME	
1063	_POSIX_THREAD_SPORADIC_SERVER	

1064	Tracing
1065 1066	This Option Group includes a set of tracing functions drawn from options within IEEE Std. 1003.1-200x (see Section 2.1.6 (on page 32)).
1067 1068 1069 1070	Where entire functions are included in the Option Group, the NAME section is marked with TRACING. Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within IEEE Std. 1003.1-200x.
1071 1072	This Option Group consists of the set of the following options from within IEEE Std. 1003.1-200x (see Section 2.1.6 (on page 32)):
1073 1074 1075 1076	_POSIX_TRACE _POSIX_TRACE_EVENT_FILTER _POSIX_TRACE_LOG _POSIX_TRACE_INHERIT
1077 1078 1079	If the implementation supports the Tracing Option Group, then the following symbolic constants shall be defined by the implementation to have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x:
1080 1081 1082 1083	_POSIX_TRACE _POSIX_TRACE_EVENT_FILTER _POSIX_TRACE_LOG _POSIX_TRACE_INHERIT
1084	XSI STREAMS
1085	The XSI STREAMS Option Group is denoted by the symbolic constant <code>_XOPEN_STREAMS</code> .
1086 1087 1088	This Option Group includes functionality related to STREAMS, a uniform mechanism for implementing networking services and other character-based I/O as described in the System Interfaces volume of IEEE Std. 1003.1-200x, Section 2.6, STREAMS.
1089	It includes the following functions:
1090	<i>fattach()</i> , <i>fdetach()</i> , <i>getmsg()</i> , <i>ioctl()</i> , <i>isastream()</i> , <i>putmsg()</i> , <i>putpmsg()</i>
1091	and the <code><stropts.h></code> header.
1092 1093 1094 1095	Where applicable, whole pages are marked STREAMS, together with the appropriate option margin legend for the SYNOPSIS section (see Section 1.5.1 (on page 10)). Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within IEEE Std. 1003.1-200x.
1096	Legacy
1097	The Legacy Option Group is denoted by the symbolic constant <code>_XOPEN_LEGACY</code> .
1098 1099	The Legacy Option Group includes the functions and headers which were mandatory in previous versions of IEEE Std. 1003.1-200x but are optional in this version.
1100 1101 1102 1103	These functions and headers are retained in IEEE Std. 1003.1-200x because of their widespread use. Application writers should not rely on the existence of these functions or headers in new applications, but should follow the migration path detailed in the APPLICATION USAGE sections of the relevant pages.
1104 1105	Various factors may have contributed to the decision to mark a function or header LEGACY. In all cases, the specific reasons for the withdrawal of a function or header are documented on the

1106 relevant pages.

1107 Once a function or header is marked LEGACY, no modifications are made to the specifications
 1108 of such functions or headers other than to the APPLICATION USAGE sections of the relevant
 1109 pages.

1110 The functions and headers which form this Option Group are as follows:

1111 *bcmp()*, *bcopy()*, *bzero()*, *ecvt()*, *fcvt()*, *ftime()*, *gcvt()*, *getwd()*, *index()*, *mktemp()*, *rindex()*,
 1112 *utimes()*

1113 An implementation that claims conformance to this Option Group shall set the macro
 1114 `_XOPEN_LEGACY` to a value other than `-1`.

1115 2.1.6 Options

1116 The following symbolic constants reflect implementation options for IEEE Std. 1003.1-200x.
 1117 These symbols can be used by the application to determine which optional facilities are present
 1118 on the implementation. The *sysconf()* function defined in the System Interfaces volume of
 1119 IEEE Std. 1003.1-200x or the *getconf* utility defined in the Shell and Utilities volume of
 1120 IEEE Std. 1003.1-200x can be used to retrieve the value of each symbol on each specific
 1121 implementation.

1122 Where an option is not supported, the associated utilities, functions, or facilities need not be
 1123 present.

1124 Margin codes are defined for each option (see Section 1.5.1 (on page 10)).

1125 2.1.6.1 System Interfaces

1126 ADV `_POSIX_ADVISORY_INFO`
 1127 If this symbolic constant is defined, then the implementation supports the functions and
 1128 additional semantics in the Advisory Information option.

1129 AIO `_POSIX_ASYNCHRONOUS_IO`
 1130 If this symbolic constant is defined, then the implementation supports the functions and
 1131 additional semantics in the Asynchronous Input and Output option.

1132 BAR `_POSIX_BARRIERS`
 1133 If this symbolic constant is defined, then the implementation supports the functions and
 1134 additional semantics in the Barriers option.

1135 CS `_POSIX_CLOCK_SELECTION`
 1136 If this symbolic constant is defined, then the implementation supports the functions and
 1137 additional semantics in the Clock Selection option.

1138 CPT `_POSIX_CPUTIME`
 1139 If this symbolic constant is defined, then the implementation supports the functions and
 1140 additional semantics in the Process CPU-Time Clocks option.

1141 FSC `_POSIX_FSYNC`
 1142 If this symbolic constant is defined, then the implementation supports the functions and
 1143 additional semantics in the File Synchronization option.

1144 IP6 `_POSIX_IPV6`
 1145 If this symbol is defined, then the implementation supports the functions and additional
 1146 semantics in the IPV6 option.

1147 MF `_POSIX_MAPPED_FILES`
 1148 If this symbolic constant is defined, then the implementation supports the functions and

1149		additional semantics in the Memory Mapped Files option.
1150	ML	_POSIX_MEMLOCK
1151		If this symbolic constant is defined, then the implementation supports the functions and
1152		additional semantics in the Process Memory Locking option.
1153	MLR	_POSIX_MEMLOCK_RANGE
1154		If this symbolic constant is defined, then the implementation supports the functions and
1155		additional semantics in the Range Memory Locking option.
1156	MPR	_POSIX_MEMORY_PROTECTION
1157		If this symbolic constant is defined, then the implementation supports the functions and
1158		additional semantics in the Memory Protection option.
1159	MSG	_POSIX_MESSAGE_PASSING
1160		If this symbolic constant is defined, then the implementation supports the functions and
1161		additional semantics in the Message Passing option.
1162	MON	_POSIX_MONOTONIC_CLOCK
1163		If this symbolic constant is defined, then the implementation supports the functions and
1164		additional semantics in the Monotonic Clock option.
1165	PIO	_POSIX_PRIORITIZED_IO
1166		If this symbolic constant is defined, then the implementation supports the functions and
1167		additional semantics in the Prioritized Input and Output option.
1168	PS	_POSIX_PRIORITY_SCHEDULING
1169		If this symbolic constant is defined, then the implementation supports the functions and
1170		additional semantics in the Process Scheduling option.
1171	RTS	_POSIX_REALTIME_SIGNALS
1172		If this symbolic constant is defined, then the implementation supports the functions and
1173		additional semantics in the Realtime Signals Extension option.
1174	SEM	_POSIX_SEMAPHORES
1175		If this symbolic constant is defined, then the implementation supports the functions and
1176		additional semantics in the Semaphores option.
1177	SHM	_POSIX_SHARED_MEMORY_OBJECTS
1178		If this symbolic constant is defined, then the implementation supports the functions and
1179		additional semantics in the Shared Memory Objects option.
1180	SH	_POSIX_SHELL
1181		If this symbolic constant is defined, then the implementation supports the <i>sh</i> utility
1182		command line interpreter specified by the Shell and Utilities volume of
1183		IEEE Std. 1003.1-200x.
1184	SPN	_POSIX_SPAWN
1185		If this symbolic constant is defined, then the implementation supports the functions and
1186		additional semantics in the Spawn option.
1187	SPI	_POSIX_SPIN_LOCKS
1188		If this symbolic constant is defined, then the implementation supports the functions and
1189		additional semantics in the Spin Locks option.
1190	SS	_POSIX_SPORADIC_SERVER
1191		If this symbolic constant is defined, then the implementation supports the functions and
1192		additional semantics in the Process Sporadic Server option.

1193	SIO	_POSIX_SYNCHRONIZED_IO
1194		If this symbolic constant is defined, then the implementation supports the functions and
1195		additional semantics in the Synchronized Input and Output option.
1196	TSA	_POSIX_THREAD_ATTR_STACKADDR
1197		If this symbolic constant is defined, then the implementation supports the additional
1198		semantics in the Thread Stack Address Attribute option.
1199	TSS	_POSIX_THREAD_ATTR_STACKSIZE
1200		If this symbolic constant is defined, then the implementation supports the additional
1201		semantics in the Thread Stack Address Size option.
1202	TCT	_POSIX_THREAD_CPUTIME
1203		If this symbolic constant is defined, then the implementation supports the functions and
1204		additional semantics in the Thread CPU-Time Clocks option.
1205	TPI	_POSIX_THREAD_PRIO_INHERIT
1206		If this symbolic constant is defined, then the implementation supports the functions and
1207		additional semantics in the Threads Priority Inheritance option.
1208	TPP	_POSIX_THREAD_PRIO_PROTECT
1209		If this symbolic constant is defined, then the implementation supports the additional
1210		semantics in the Thread Priority Protection option.
1211	TPS	_POSIX_THREAD_PRIORITY_SCHEDULING
1212		If this symbolic constant is defined, then the implementation supports the functions and
1213		additional semantics in the Thread Execution Scheduling option.
1214	TSH	_POSIX_THREAD_PROCESS_SHARED
1215		If this symbolic constant is defined, then the implementation supports the additional
1216		semantics in the Thread Process-Shared Synchronization option.
1217	TSF	_POSIX_THREAD_SAFE_FUNCTIONS
1218		If this symbolic constant is defined, then the implementation supports the functions and
1219		additional semantics in the Thread-Safe Functions option.
1220	TSP	_POSIX_THREAD_SPORADIC_SERVER
1221		If this symbolic constant is defined, then the implementation supports the functions and
1222		additional semantics in the Thread Sporadic Server option.
1223	THR	_POSIX_THREADS
1224		If this symbolic constant is defined, then the implementation supports the functions and
1225		additional semantics in the Threads option.
1226	TMO	_POSIX_TIMEOUTS
1227		If this symbolic constant is defined, then the implementation supports the functions and
1228		additional semantics in the Timeouts option.
1229	TMR	_POSIX_TIMERS
1230		If this symbolic constant is defined, then the implementation supports the functions and
1231		additional semantics in the Timers option.
1232	TRC	_POSIX_TRACE
1233		If this symbolic constant is defined, then the implementation supports the functions and
1234		additional semantics in the Trace option.
1235	TEF	_POSIX_TRACE_EVENT_FILTER
1236		If this symbolic constant is defined, then the implementation supports the functions and
1237		additional semantics in the Trace Event Filter option.

1238	TRL	_POSIX_TRACE_LOG
1239		If this symbolic constant is defined, then the implementation supports the functions and
1240		additional semantics in the Trace Log option.
1241	TRI	_POSIX_TRACE_INHERIT
1242		If this symbolic constant is defined, then the implementation supports the functions and
1243		additional semantics in the Trace Inherit option.
1244	TYM	_POSIX_TYPED_MEMORY_OBJECTS
1245		If this symbolic constant is defined, then the implementation supports the functions and
1246		additional semantics in the Typed Memory Objects option.
1247	2.1.6.2	<i>Shell and Utilities</i>
1248		Each of these symbols shall be considered valid names by the implementation. Each shall be
1249		defined on the system with a value of 1 if the corresponding option is supported; otherwise, the
1250		symbol shall be undefined.
1251		The literal names shown below apply only to the <i>getconf</i> utility.
1252	CD	POSIX2_C_DEV
1253		The system supports the C-Language Development Utilities option.
1254		The utilities in the C-Language Development Utilities option are used for the development
1255		of C-language applications, including compilation or translation of C source code and
1256		complex program generators for simple lexical tasks and processing of context-free
1257		grammars.
1258		The utilities listed below may be provided by a conforming system; however, any system
1259		claiming conformance to the C-Language Development Utilities option shall provide all of
1260		the utilities listed.
1261		<i>c99</i>
1262		<i>lex</i>
1263		<i>yacc</i>
1264		POSIX2_CHAR_TERM
1265		The system supports the Terminal Characteristics option. This value need not be present on
1266		a system not supporting the User Portability Utilities option.
1267		Where applicable, the dependency is noted within the description of the utility.
1268		This option applies only to systems supporting the User Portability Utilities option. If
1269		supported, then the system supports at least one terminal type capable of all operations
1270		described in IEEE Std. 1003.1-200x; see Section 10.2 (on page 211).
1271	FD	POSIX2_FORT_DEV
1272		The system supports the FORTRAN Development Utilities option.
1273		The <i>fort77</i> FORTRAN compiler is the only utility in the FORTRAN Development Utilities
1274		option. This is used for the development of FORTRAN language applications, including
1275		compilation or translation of FORTRAN source code.
1276		The <i>fort77</i> utility may be provided by a conforming system; however, any system claiming
1277		conformance to the FORTRAN Development Utilities option shall provide the <i>fort77</i> utility.
1278	FR	POSIX2_FORT_RUN
1279		The system supports the FORTRAN Runtime Utilities option.

- 1280 The *asa* utility is the only utility in the FORTRAN Runtime Utilities option.
- 1281 The *asa* utility may be provided by a conforming system; however, any system claiming
1282 conformance to the FORTRAN Runtime Utilities option shall provide the *asa* utility.
- 1283 POSIX2_LOCALEDEF
1284 The system supports the Locale Creation Utilities option.
- 1285 If supported, the system supports the creation of locales as described in the *localedef* utility.
- 1286 The *localedef* utility may be provided by a conforming system; however, any system
1287 claiming conformance to the Locale Creation Utilities option shall provide the *localedef*
1288 utility.
- 1289 BE POSIX2_PBS
1290 The system supports the Batch Environment Services and Utilities option (see the Shell and
1291 Utilities volume of IEEE Std. 1003.1-200x, Chapter 3, Batch Environment Services).
- 1292 **Note:** The Batch Environment Services and Utilities option is a combination of
1293 mandatory and optional batch services and utilities. The POSIX_PBS symbolic
1294 constant implies the system supports all the mandatory batch services and
1295 utilities.
- 1296 POSIX2_PBS_ACCOUNTING
1297 The system supports the Batch Accounting option.
- 1298 POSIX2_PBS_CHECKPOINT
1299 The system supports the Batch Checkpoint/Restart option.
- 1300 POSIX2_PBS_LOCATE
1301 The system supports the Locate Batch Job Request option.
- 1302 POSIX2_PBS_MESSAGE
1303 The system supports the Batch Job Message Request option.
- 1304 POSIX2_PBS_TRACK
1305 The system supports the Track Batch Job Request option.
- 1306 SD POSIX2_SW_DEV
1307 The system supports the Software Development Utilities option.
- 1308 The utilities in the Software Development Utilities option are used for the development of
1309 applications, including compilation or translation of source code, the creation and
1310 maintenance of library archives, and the maintenance of groups of inter-dependent
1311 programs.
- 1312 The utilities listed below may be provided by the conforming system; however, any system
1313 claiming conformance to the Software Development Utilities option shall provide all of the
1314 utilities listed here.
- 1315 *ar*
1316 *make*
1317 *nm*
1318 *strip*
- 1319 UP POSIX2_UPE
1320 The system supports the User Portability Utilities option.
- 1321 The utilities in the User Portability Utilities option shall be implemented on all systems that
1322 claim conformance to this option. Certain utilities are noted as having features that cannot
1323 be implemented on all terminal types; if the POSIX2_CHAR_TERM option is supported, the

1324 system shall support all such features on at least one terminal type; see Section 10.2 (on
1325 page 211).

1326 Some of the utilities are required only on systems that also support the Software
1327 Development Utilities option, or the character-at-a-time terminal option (see Section 10.2
1328 (on page 211)); such utilities have this noted in their DESCRIPTION sections. All of the
1329 other utilities listed are required only on systems that claim conformance to the User
1330 Portability Utilities option.

1331	<i>alias</i>	<i>expand</i>	<i>nm</i>	<i>unalias</i>		
1332	<i>at</i>	<i>fc</i>	<i>patch</i>	<i>unexpand</i>		
1333	<i>batch</i>	<i>fg</i>	<i>ps</i>	<i>udecode</i>		
1334	<i>bg</i>	<i>file</i>	<i>renice</i>	<i>uencode</i>		
1335	<i>crontab</i>	<i>jobs</i>	<i>split</i>	<i>vi</i>		
1336	<i>split</i>	<i>man</i>	<i>strings</i>	<i>who</i>		
1337	<i>ctags</i>	<i>mesg</i>	<i>tabs</i>	<i>write</i>		
1338	<i>df</i>	<i>more</i>	<i>talk</i>			
1339	<i>du</i>	<i>newgrp</i>	<i>time</i>			
1340	<i>ex</i>	<i>nice</i>	<i>tput</i>			

1341 2.2 Application Conformance

1342 All applications claiming conformance to IEEE Std. 1003.1-200x shall use only language-
1343 dependent services for the C programming language described in Section 2.3 (on page 40), shall
1344 use only the utilities and facilities defined in the Shell and Utilities volume of
1345 IEEE Std. 1003.1-200x, and shall fall within one of the following categories.

1346 2.2.1 Strictly Conforming POSIX Application

1347 A Strictly Conforming POSIX Application is an application that requires only the facilities
1348 described in IEEE Std. 1003.1-200x. Such an application:

- 1349 1. Shall accept any implementation behavior that results from actions it takes in areas
1350 described in IEEE Std. 1003.1-200x as *implementation-defined* or *unspecified*, or where
1351 IEEE Std. 1003.1-200x indicates that implementations may vary
- 1352 2. Shall not perform any actions that are described as producing *undefined* results
- 1353 3. For symbolic constants, shall accept any value in the range permitted by
1354 IEEE Std. 1003.1-200x, but shall not rely on any value in the range being greater than the
1355 minimums listed or being less than the maximums listed in IEEE Std. 1003.1-200x
- 1356 4. Shall not use facilities designated as *obsolescent*
- 1357 5. Is required to tolerate and permitted to adapt to the presence or absence of optional
1358 facilities whose availability is indicated by Section 2.1.3 (on page 20)
- 1359 6. For the C programming language, shall not produce any output dependent on any
1360 behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-*
1361 *defined*, unless the System Interfaces volume of IEEE Std. 1003.1-200x specifies the behavior
- 1362 7. For the C programming language, shall not exceed any minimum implementation limit
1363 defined in the ISO C standard, unless the System Interfaces volume of
1364 IEEE Std. 1003.1-200x specifies a higher minimum implementation limit
- 1365 8. For the C programming language, shall define `_POSIX_C_SOURCE` to be 200xxxL before
1366 any header is included

1367 Within IEEE Std. 1003.1-200x, any restrictions placed upon a Conforming POSIX Application
1368 shall restrict a Strictly Conforming POSIX Application.

1369 2.2.2 Conforming POSIX Application

1370 2.2.2.1 ISO/IEC Conforming POSIX Application

1371 An ISO/IEC Conforming POSIX Application is an application that uses only the facilities
1372 described in IEEE Std. 1003.1-200x and approved Conforming Language bindings for any ISO or
1373 IEC standard. Such an application shall include a statement of conformance that documents all
1374 options and limit dependencies, and all other ISO or IEC standards used.

1375 2.2.2.2 <National Body> Conforming POSIX Application

1376 A <National Body> Conforming POSIX Application differs from an ISO/IEC Conforming
1377 POSIX Application in that it also may use specific standards of a single ISO/IEC member body
1378 referred to here as <National Body>. Such an application shall include a statement of
1379 conformance that documents all options and limit dependencies, and all other <National Body>
1380 standards used.

1381 2.2.3 Conforming POSIX Application Using Extensions

1382 A Conforming POSIX Application Using Extensions is an application that differs from a
 1383 Conforming POSIX Application only in that it uses non-standard facilities that are consistent
 1384 with IEEE Std. 1003.1-200x. Such an application shall fully document its requirements for these
 1385 extended facilities, in addition to the documentation required of a Conforming POSIX
 1386 Application. A Conforming POSIX Application Using Extensions shall be either an ISO/IEC
 1387 Conforming POSIX Application Using Extensions or a <National Body> Conforming POSIX
 1388 Application Using Extensions (see Section 2.2.2.1 (on page 38) and Section 2.2.2.2 (on page 38)).

1389 2.2.4 Strictly Conforming XSI Application

1390 A Strictly Conforming XSI Application is an application that requires only the facilities described
 1391 in IEEE Std. 1003.1-200x. Such an application:

- 1392 1. Shall accept any implementation behavior that results from actions it takes in areas
 1393 described in IEEE Std. 1003.1-200x as *implementation-defined* or *unspecified*, or where
 1394 IEEE Std. 1003.1-200x indicates that implementations may vary
- 1395 2. Shall not perform any actions that are described as producing *undefined* results
- 1396 3. For symbolic constants, shall accept any value in the range permitted by
 1397 IEEE Std. 1003.1-200x, but shall not rely on any value in the range being greater than the
 1398 minimums listed or being less than the maximums listed
- 1399 4. Shall not use facilities designated as *obsolescent*
- 1400 5. Is required to tolerate and permitted to adapt to the presence or absence of optional
 1401 facilities whose availability is indicated by Section 2.1.4 (on page 23)
- 1402 6. For the C programming language, shall not produce any output dependent on any
 1403 behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-*
 1404 *defined*, unless the System Interfaces volume of IEEE Std. 1003.1-200x specifies the behavior
- 1405 7. For the C programming language, shall not exceed any minimum implementation limit
 1406 defined in the ISO C standard, unless the System Interfaces volume of
 1407 IEEE Std. 1003.1-200x specifies a higher minimum implementation limit
- 1408 8. For the C programming language, shall define `_XOPEN_SOURCE` to be 600 before any
 1409 header is included

1410 Within IEEE Std. 1003.1-200x, any restrictions placed upon a Conforming POSIX Application
 1411 shall restrict a Strictly Conforming XSI Application.

1412 2.2.5 Conforming XSI Application Using Extensions

1413 A Conforming XSI Application Using Extensions is an application that differs from a Strictly
 1414 Conforming XSI Application only in that it uses non-standard facilities that are consistent with
 1415 IEEE Std. 1003.1-200x. Such an application shall fully document its requirements for these
 1416 extended facilities, in addition to the documentation required of a Strictly Conforming XSI
 1417 Application.

1418 2.3 Language-Dependent Services for the C Programming Language

1419 Implementors seeking to claim conformance using the ISO C standard shall claim POSIX
1420 conformance as described in Section 2.1.3 (on page 20), C Language Binding (C Standard
1421 Language-Dependent System Support).

1422 2.4 Other Language-Related Specifications

1423 IEEE Std. 1003.1-200x is currently specified in terms of the shell command language and ISO C.
1424 Bindings to other programming languages are being developed.

1425 If conformance to IEEE Std. 1003.1-200x is claimed for implementation of any programming
1426 language, the implementation of that language shall support the use of external symbols distinct
1427 to at least 31 bytes in length in the source program text. (That is, identifiers that differ at or
1428 before the thirty-first byte shall be distinct.) If a national or international standard governing a
1429 language defines a maximum length that is less than this value, the language-defined maximum
1430 shall be supported. External symbols that differ only by case shall be distinct when the character
1431 set in use distinguishes uppercase and lowercase characters and the language permits (or
1432 requires) uppercase and lowercase characters to be distinct in external symbols.

1433

1434 3.1 Abortive Release

1435 An abrupt termination of a network connection that may result in the loss of data.

1436 3.2 Absolute Path Name

1437 A path name beginning with a single or more than two slashes; see also Section 3.268 (on page
1438 86).

1439 **Note:** Path Name Resolution is defined in detail in Section 4.9 (on page 123).

1440 3.3 Access Mode

1441 A particular form of access permitted to a file.

1442 3.4 Additional File Access Control Mechanism

1443 An implementation-defined mechanism that is layered upon the access control mechanisms
1444 defined here, but which do not grant permissions beyond those defined herein, although they
1445 may further restrict them.

1446 **Note:** File Access Permissions are defined in detail in Section 4.3 (on page 121).

1447 3.5 Address Space

1448 The memory locations that can be referenced by a process or the threads of a process.

1449 3.6 Advisory Information

1450 An interface that advises the implementation on (portable) application behavior so that it can
1451 optimize the system.

1452 **3.7 Affirmative Response**

1453 An input string that matches one of the responses acceptable to the *LC_MESSAGES* category
1454 keyword **yesexpr**, matching an extended regular expression in the current locale.

1455 **Note:** The *LC_MESSAGES* category is defined in detail in Section 7.3.6 (on page 174).

1456 **3.8 Alert**

1457 To cause the user's terminal to give some audible or visual indication that an error or some other
1458 event has occurred. When the standard output is directed to a terminal device, the method for
1459 alerting the terminal user is unspecified. When the standard output is not directed to a terminal
1460 device, the alert is accomplished by writing the <alert> character to standard output (unless the
1461 utility description indicates that the use of standard output produces undefined results in this
1462 case).

1463 **3.9 Alert Character (<alert>)**

1464 A character that in the output stream should cause a terminal to alert its user via a visual or
1465 audible notification. The <alert> character is the character designated by '\a' in the C
1466 language. It is unspecified whether this character is the exact sequence transmitted to an output
1467 device by the system to accomplish the alert function.

1468 **3.10 Alias Name**

1469 In the shell command language, a word consisting solely of underscores, digits, and alphabets
1470 from the portable character set and any of the following characters: '!', '%', ', ', '@'.

1471 Implementations may allow other characters within alias names as an extension.

1472 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 133).

1473 **3.11 Alignment**

1474 A requirement that objects of a particular type be located on storage boundaries with addresses
1475 that are particular multiples of a byte address

1476 **Note:** See also the ISO C standard, § B3.

1477 3.12 Alternate File Access Control Mechanism

1478 An implementation-defined mechanism that is independent of the access control mechanisms
1479 defined herein, and which if enabled on a file may either restrict or extend the permissions of a
1480 given user. IEEE Std. 1003.1-200x defines when such mechanisms can be enabled and when they
1481 are disabled.

1482 **Note:** File Access Permissions are defined in detail in Section 4.3 (on page 121).

1483 3.13 Alternate Signal Stack

1484 Memory associated with a thread, established upon request by the implementation for a thread,
1485 separate from the thread signal stack, in which signal handlers responding to signals sent to that
1486 thread may be executed.

1487 3.14 Ancillary Data

1488 Protocol-specific, local system-specific, or optional information. The information can be both
1489 local or end-to-end significant, header information, part of a data portion, protocol-specific, and
1490 implementation or system-specific.

1491 3.15 Angle Brackets

1492 The characters '`<`' (left-angle-bracket) and '`>`' (right-angle-bracket). When used in the phrase
1493 "enclosed in angle brackets", the symbol '`<`' immediately precedes the object to be enclosed,
1494 and '`>`' immediately follows it. When describing these characters in the portable character set,
1495 the names `<less-than-sign>` and `<greater-than-sign>` are used.

1496 3.16 Application

1497 A computer program that performs some desired function.

1498 3.17 Application Address

1499 Endpoint address of a specific application.

1500 3.18 Application Program Interface (API)

1501 The definition of syntax and semantics for providing computer system services.

1502 3.19 Appropriate Privileges

1503 An implementation-defined means of associating privileges with a process with regard to the
1504 function calls, function call options, and the commands that need special privileges. There may
1505 be zero or more such means. These means (or lack thereof) are described in the conformance
1506 document.

1507 **Note:** Function calls are defined in the System Interfaces volume of IEEE Std. 1003.1-200x,
1508 and commands are defined in the Shell and Utilities volume of IEEE Std. 1003.1-200x.

1509 3.20 Argument

1510 In the shell command language, a parameter passed to a utility as the equivalent of a single
1511 string in the *argv* array created by one of the *exec* functions. An argument is one of the options,
1512 option-arguments, or operands following the command name.

1513 **Note:** The Utility Argument Syntax is defined in detail in Section 12.1 (on page 227) and the
1514 Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.9.1.1, Command Search
1515 and Execution.

1516 In the C language, an expression in a function call expression or a sequence of preprocessing
1517 tokens in a function-like macro invocation.

1518 3.21 Arm (a Timer)

1519 To start a timer measuring the passage of time, enabling notifying a process when the specified
1520 time or time interval has passed.

1521 3.22 Assignment

1522 NEW DEF REQUIRED.

1523 **Note:** Variable Assignment is defined in detail in Section 4.16 (on page 127).

1524 3.23 Asterisk

1525 The character ' * '.

1526 3.24 Async-Cancel-Safe Function

1527 A function that may be safely invoked by an application while the asynchronous form of
1528 cancelation is enabled. No function is async-cancel-safe unless explicitly described as such.

1529 3.25 Asynchronous Events

1530 Events that occur independently of the execution of the application.

1531 3.26 Asynchronous Input and Output

1532 A functionality enhancement to allow an application process to queue data input and output
1533 commands with asynchronous notification of completion. This facility includes in its scope the
1534 requirements of supercomputer applications.

1535 3.27 Async-Signal-Safe Function

1536 A function that may be invoked, without restriction, from signal-catching functions. No function
1537 is async-signal-safe unless explicitly described as such.

1538 3.28 Asynchronously-Generated Signal

1539 A signal that is not attributable to a specific thread. Examples are signals sent via *kill()*, signals
1540 sent from the keyboard, and signals delivered to process groups. Being asynchronous is a
1541 property of how the signal was generated and not a property of the signal number. All signals
1542 may be generated asynchronously.

1543 **Note:** The *kill()* function is defined in detail in the System Interfaces volume of
1544 IEEE Std. 1003.1-200x.

1545 3.29 Asynchronous I/O Operation

1546 An I/O operation that does not of itself cause the thread requesting the I/O to be blocked from
1547 further use of the processor. |

1548 This implies that the process and the I/O operation may be running concurrently. |

1549 3.30 Asynchronous I/O Completion

1550 For an asynchronous read or write operation, when a corresponding synchronous read or write
1551 would have completed and when any associated status fields have been updated. |

1552 3.31 Authentication

1553 The process of validating a user or process to verify that the user or process is not a counterfeit.

1554 3.32 Authorization

1555 The process of verifying that a user or process has permission to use a resource in the manner
1556 requested.

1557 To ensure security, the user or process would also need to be authenticated before granting
1558 access.

1559 3.33 Background Job

1560 See *Background Process Group* in Section 3.35.

1561 3.34 Background Process

1562 A process that is a member of a background process group.

1563 3.35 Background Process Group (or Background Job)

1564 Any process group, other than a foreground process group, that is a member of a session that
1565 has established a connection with a controlling terminal.

1566 3.36 Backquote

1567 The character ' ` ', also known as a *grave accent*.

1568 3.37 Backslash

1569 The character ' \ ', also known as a *reverse solidus*.

1570 3.38 Backspace Character (<backspace>)

1571 A character that, in the output stream, should cause printing (or displaying) to occur one column
1572 position previous to the position about to be printed. If the position about to be printed is at the
1573 beginning of the current line, the behavior is unspecified. The <backspace> character is the
1574 character designated by ' \b ' in the C language. It is unspecified whether this character is the
1575 exact sequence transmitted to an output device by the system to accomplish the backspace
1576 function. The <backspace> character defined here is not necessarily the ERASE special character.

1577 **Note:** Special Characters are defined in detail in Section 11.1.9 (on page 217).

1578 3.39 Barrier

1579 A synchronization object that allows multiple threads to synchronize at a particular point in
1580 their execution.

1581 3.40 Base Character

1582 One of the set of characters defined in the Latin alphabet. In Western European languages other
1583 than English, these characters are commonly used with diacritical marks (accents, cedilla, and so
1584 on) to extend the range of characters in an alphabet.

1585 3.41 Basename

1586 The final, or only, file name in a path name.

1587 3.42 Basic Regular Expression (BRE)

1588 A regular expression (see Section 3.318 (on page 95)) used by the majority of utilities that select
1589 strings from a set of character strings.

1590 **Note:** Basic Regular Expressions are described in detail in Section 9.3 (on page 198).

1591 3.43 Batch Access List

1592 A list of user IDs and group IDs of those users and groups authorized to place batch jobs in a
1593 batch queue.

1594 A batch access list is associated with a batch queue. A batch server uses the batch access list of a
1595 batch queue as one of the criteria in deciding to put a batch job in a batch queue.

1596 3.44 Batch Administrator

1597 A person who is authorized to use all restricted batch services.

1598 3.45 Batch Client

1599 A computational entity that utilizes batch services by making requests of batch servers.

1600 Batch clients often provide the means by which users access batch services, although a batch
1601 server may act as a batch client by virtue of making requests of another batch server.

1602 3.46 Batch Destination

1603 The batch server in a batch system to which a batch job should be sent for processing.

1604 Acceptance of a batch job at a batch destination is the responsibility of a receiving batch server.
1605 A batch destination may consist of a batch server-specific portion, a network-wide portion, or
1606 both. The batch server-specific portion is referred to as the *batch queue*. The network-wide
1607 portion is referred to as a *batch server name*.

1608 3.47 Batch Destination Identifier

1609 A string that identifies a specific batch destination.

1610 A string of characters in the portable character set used to specify a particular batch destination.

1611 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 133).

1612 3.48 Batch Directive

1613 A line from a file that is interpreted by the batch server. The line is usually in the form of a
1614 comment and is an additional means of passing options to the *qsub* utility.

1615 **Note:** The *qsub* utility is defined in detail in the Shell and Utilities volume of
1616 IEEE Std. 1003.1-200x.

1617 3.49 Batch Job

1618 A set of computational tasks for a computing system.

1619 Batch jobs are managed by batch servers.

1620 Once created, a batch job may be executing or pending execution. A batch job that is executing
1621 has an associated session leader (a process) that initiates and monitors the computational tasks
1622 of the batch job.

1623 3.50 Batch Job Attribute

1624 A named data type whose value affects the processing of a batch job.

1625 The values of the attributes of a batch job affect the processing of that job by the batch server
1626 that manages the batch job.

1627 The attributes defined for a batch job are called the batch job attributes.

1628 3.51 Batch Job Identifier

1629 A unique name for a batch job. A name that is unique among all other batch job identifiers in a
1630 batch system and that identifies the batch server to which the batch job was originally
1631 submitted.

1632 3.52 Batch Job Name

1633 A label that is an attribute of a batch job. The batch job name is not necessarily unique.

1634 3.53 Batch Job Owner

1635 The *username@hostname* of the user submitting the batch job, where *username* is a user name (see
1636 also Section 3.428 (on page 115)) and *hostname* is a network host name.

1637 3.54 Batch Job Priority

1638 An attribute used in selecting a batch job for execution.

1639 A value specified by the user that may be used by an implementation to determine the order in
1640 which batch jobs are selected to be executed. Job priority has a numeric value in the range -1 024
1641 to 1 023.

1642 **Note:** The batch job priority is not the execution priority (nice value) of the batch job.

1643 3.55 Batch Job State

1644 An attribute of a batch job.

1645 The state of a batch job determines the types of requests that the batch server that manages the
1646 batch job can accept for the batch job. Valid states include QUEUED, RUNNING, HELD,
1647 WAITING, EXITING, and TRANSITING.

1648 3.56 Batch Name Service

1649 A service that assigns batch names that are unique within the batch name space, and that can
1650 translate a unique batch name into the location of the named batch entity.

1651 3.57 Batch Name Space

1652 The environment within which a batch name is known to be unique.

1653 3.58 Batch Node

1654 A host containing part or all of a batch system.

1655 A batch node is a host meeting at least one of the following conditions:

- 1656 • Capable of executing a batch client
- 1657 • Contains a routing batch queue
- 1658 • Contains an execution batch queue

1659 3.59 Batch Operator

1660 A person who is authorized to use some, but not all, restricted batch services. |

1661 3.60 Batch Queue

1662 A manageable object that represents a set of batch jobs and is managed by a single batch server. |

1663 **Note:** Each batch job managed by a batch server is a member of a single batch queue
1664 managed by that server.

1665 Such a set of batch jobs is called a batch queue largely for historical reasons. Jobs are
1666 selected from the batch queue for execution based on attributes such as priority,
1667 resource requirements, and hold conditions.

1668 Two types of batch queue are described in IEEE Std. 1003.1-200x: *routing batch queues*
1669 and *execution batch queues*.

1670 3.61 Batch Queue Attribute

1671 A named data type whose value affects the processing of all batch jobs that are members of the
1672 batch queue. |

1673 A batch queue has attributes that affect the processing of batch jobs that are members of the
1674 batch queue.

1675 The attributes defined for a batch queue are called the batch batch queue attributes. |

1676 3.62 Batch Queue Position

1677 The place a batch job occupies in a batch queue. |

1678 This place is relative to other batch jobs in the batch queue and defined in part by submission
1679 time and its priority; see also Section 3.63. |

1680 3.63 Batch Queue Priority

1681 The maximum job priority allowed for any batch job in a given batch queue. |

1682 The batch queue priority is set and may be changed by users with appropriate privilege. The
1683 priority is bounded in an implementation-defined manner. |

1684 3.64 Batch Rerunability

1685 An attribute of a batch job.

1686 If a batch job may be rerun from the beginning after an abnormal termination without affecting
1687 the validity of the results, the batch job is said to be rerunable.

1688 3.65 Batch Restart

1689 Resume the processing of a batch job from the point of the last checkpoint. Typically, this is done
1690 if the batch job has been interrupted because of a system failure.

1691 3.66 Batch Server

1692 A computational entity that provides batch services.

1693 3.67 Batch Server Name

1694 A string that identifies a specific server in a network.

1695 A string of characters in the portable character set used to specify a particular server in a
1696 network.

1697 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 133).

1698 3.68 Batch Service

1699 Computational and organizational services performed by a batch system on behalf of batch jobs.

1700 Batch services are of two types: *requested* and *deferred*.

1701 **Note:** Batch Services are listed in the Shell and Utilities volume of IEEE Std. 1003.1-200x,
1702 Table 3-5, Batch Services Summary.

1703 3.69 Batch Service Request

1704 A solicitation of services from a batch client to a batch server.

1705 A batch service request may entail the exchange of any number of messages between the batch
1706 client and the batch server.

1707 When naming specific types of service requests, the term request is qualified by the type of
1708 request, as in *Queue Batch Job Request* and *Delete Batch Job Request*.

1709 3.70 Batch Submission

1710 The process by which a batch client requests that a batch server create a batch job via a *Queue Job*
1711 *Request* to perform a specified computational task.

1712 3.71 Batch System

1713 A collection of one or more batch servers.

1714 3.72 Batch Target User

1715 The name of a user on the batch destination batch server.

1716 The target user is the user name under whose account the batch job is to execute on the
1717 destination batch server.

1718 3.73 Batch User

1719 A person who is authorized to make use of batch services.

1720 3.74 Bind

1721 Assign a network address to an endpoint.

1722 3.75 Blank Character (<blank>)

1723 One of the characters that belong to the **blank** character class as defined via the *LC_CTYPE*
1724 category in the current locale. In the POSIX locale, a <blank> character is either a <tab> or a
1725 <space> character.

1726 3.76 Blank Line

1727 A line consisting solely of zero or more <blank> characters terminated by a <newline> character;
1728 see also Section 3.146 (on page 66).

1729 3.77 Blocked Process (or Thread)

1730 A process (or thread) that is waiting for some condition (other than the availability of a
1731 processor) to be satisfied before it can continue execution.

1732 3.78 Blocking

1733 Executing with O_NONBLOCK not set; see also Section 3.242 (on page 82).

1734 3.79 Block-Mode Terminal

1735 A terminal device operating in a mode incapable of the character-at-a-time input and output
1736 operations described by some of the standard utilities.

1737 **Note:** Output Devices and Terminal Types are defined in detail in Section 10.2 (on page
1738 211).

1739 3.80 Block Special File

1740 A file that refers to a device. A block special file is normally distinguished from a character
1741 special file by providing access to the device in a manner such that the hardware characteristics
1742 of the device are not visible.

1743 3.81 Braces

1744 The characters '{' (left brace) and '}' (right brace), also known as *curly braces*. When used in
1745 the phrase “enclosed in (curly) braces” the symbol '{' immediately precedes the object to be
1746 enclosed, and '}' immediately follows it. When describing these characters in the portable
1747 character set, the names <left-brace> and <right-brace> are used.

1748 3.82 Brackets

1749 The characters '[' (left-bracket) and ']' (right-bracket), also known as *square brackets*. When
1750 used in the phrase “enclosed in (square) brackets” the symbol '[' immediately precedes the
1751 object to be enclosed, and ']' immediately follows it. When describing these characters in the
1752 portable character set, the names <left-square-bracket> and <right-square-bracket> are used.

1753 **3.83 Break Value**

1754 The address at which dynamic memory allocation starts.

1755 **3.84 Broadcast**

1756 The transfer of data from one endpoint to several endpoints, as described in RFC 919 and
1757 RFC 922.

1758 **3.85 Built-In Utility (or Built-In)**

1759 A utility implemented within a shell. The utilities referred to as *special built-ins* have special
1760 qualities. Unless qualified, the term built-in includes the special built-in utilities. *Regular built-ins*
1761 are not required to be actually built into the shell on the implementation, but they do have
1762 special command-search qualities.

1763 **Note:** Special Built-In Utilities are defined in detail in the Shell and Utilities volume of
1764 IEEE Std. 1003.1-200x, Section 2.15, Special Built-In Utilities.

1765 Regular Built-In Utilities are defined in detail in the Shell and Utilities volume of
1766 IEEE Std. 1003.1-200x, Section 2.9.1.1, Command Search and Execution.

1767 **3.86 Byte**

1768 An individually addressable unit of data storage that is equal to or larger than an octet, used to
1769 store a character or a portion of a character; see also Section 3.89 (on page 56). A byte is
1770 composed of a contiguous sequence of bits, the number of which is implementation-defined. The
1771 least significant bit is called the *low-order* bit; the most significant is called the *high-order* bit.

1772 **Note:** The definition of *byte* is actually from the ISO C standard. It has been reworded
1773 slightly to clarify its intent without introducing the ISO C standard terminology
1774 “basic execution character set”, which is inapplicable to IEEE Std. 1003.1-200x. It
1775 deviates intentionally from the usage of *byte* in some international standards, where
1776 it is used as a synonym for *octet* (always eight bits). A byte may be larger than eight
1777 bits so that it can be an integral portion of larger data objects that are not evenly
1778 divisible by eight bits (such as a 36-bit word that contains four 9-bit bytes).

1779 3.87 Byte Input/Output Functions

1780 The functions that perform byte-oriented input from streams or byte-oriented output to streams:
1781 *fgetc()*, *fgets()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*, *fscanf()*, *fwrite()*, *getc()*, *getchar()*, *gets()*, *printf()*,
1782 *putc()*, *putchar()*, *puts()*, *scanf()*, *ungetc()*, *vfprintf()*, and *vprintf()*.

1783 **Note:** Functions are defined in detail in the System Interfaces volume of
1784 IEEE Std. 1003.1-200x.

1785 3.88 Carriage-Return Character (<carriage-return>)

1786 A character that in the output stream indicates that printing should start at the beginning of the
1787 same physical line in which the <carriage-return> character occurred. The <carriage-return>
1788 character is the character designated by '`\r`' in the C language. It is unspecified whether this
1789 character is the exact sequence transmitted to an output device by the system to accomplish the
1790 movement to the beginning of the line.

1791 3.89 Character

1792 A sequence of one or more bytes representing a single graphic symbol or control code.

1793 **Note:** This term corresponds to the ISO C standard term multi-byte character, where a
1794 single-byte character is a special case of a multi-byte character. Unlike the usage in
1795 the ISO C standard, *character* here has no necessary relationship with storage space,
1796 and *byte* is used when storage space is discussed.

1797 See the definition of the Portable Character Set in Section 6.1 (on page 133) for a
1798 further explanation of the graphical representations of characters, or *glyphs*, as
1799 opposed to character encodings.

1800 3.90 Character Array

1801 An array of elements of type **char**.

1802 3.91 Character Class

1803 A named set of characters sharing an attribute associated with the name of the class. The classes
1804 and the characters that they contain are dependent on the value of the *LC_CTYPE* category in the
1805 current locale.

1806 **Note:** The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 147).

1807 3.92 Character Set

1808 A finite set of different characters used for the representation, organization, or control of data.

1809 3.93 Character Special File

1810 A file that refers to a device. One specific type of character special file is a terminal device file.

1811 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 213).

1812 3.94 Character String

1813 A contiguous sequence of characters terminated by and including the first null byte.

1814 3.95 Child Process

1815 A new process created (by *fork()* or *spawn()*) by a given process. A child process remains the
1816 child of the creating process as long as both processes continue to exist.

1817 **Note:** The *fork()* and *spawn()* functions are defined in detail in the System Interfaces
1818 volume of IEEE Std. 1003.1-200x.

1819 3.96 Circumflex

1820 The character '^'.

1821 3.97 Clock

1822 A software or hardware object that can be used to measure the apparent or actual passage of
1823 time.

1824 The current value of the time measured by a clock can be queried and, possibly, set to a value
1825 within the legal range of the clock.

1826 3.98 Clock Jump

1827 The difference between two successive distinct values of a clock, as observed from the
1828 application via one of the "get time" operations.

1829 3.99 Clock Tick

1830 An interval of time; an implementation-defined number of these occur each second. Clock ticks
1831 are one of the units that may be used to express a value found in type `clock_t`.

1832 3.100 Coded Character Set

1833 A set of unambiguous rules that establishes a character set and the one-to-one relationship
1834 between each character of the set and its bit representation.

1835 3.101 Codeset

1836 The result of applying rules that map a numeric code value to each element of a character set. An
1837 element of a character set may be related to more than one numeric code value but the reverse is
1838 not true. However, for state-dependent encodings the relationship between numeric code values
1839 to elements of a character set may be further controlled by state information. The character set
1840 may contain fewer elements than the total number of possible numeric code values; that is, some
1841 code values may be unassigned.

1842 **Note:** Character Encoding is defined in detail in Section 6.2 (on page 136).

1843 3.102 Collating Element

1844 The smallest entity used to determine the logical ordering of character or wide-character strings;
1845 see also Section 3.105 (on page 59). A collating element consists of either a single character, or
1846 two or more characters collating as a single entity. The value of the `LC_COLLATE` category in the
1847 current locale determines the current set of collating elements.

1848 3.103 Collating Element Order

1849 The relative order of collating elements as determined by the setting of the `LC_COLLATE`
1850 category in the current locale.

1851 The collating element order is used in range expressions in REs and is determined by the order in
1852 which collating elements are specified between `order_start` and `order_end` keywords in the
1853 `LC_COLLATE` category.

1854 3.104 Collation

1855 The logical ordering of character or wide-character strings according to defined precedence
1856 rules. These rules identify a collation sequence between the collating elements, and such
1857 additional rules that can be used to order strings consisting of multiple collating elements.

1858 3.105 Collation Sequence

1859 The relative order of collating elements as determined by the setting of the *LC_COLLATE*
1860 category in the current locale. The collation sequence is used for sorting and is determined from
1861 the collating weights assigned to each collating element. In the absence of weights, the collation
1862 sequence is also the collating element order.

1863 Multi-level sorting is accomplished by assigning elements one or more collation weights, up to
1864 the limit {*COLL_WEIGHTS_MAX*}. On each level, elements may be given the same weight (at
1865 the primary level, called an equivalence class; see also Section 3.152 (on page 66)) or be omitted
1866 from the sequence. Strings that collate equal using the first assigned weight (primary ordering)
1867 are then compared using the next assigned weight (secondary ordering), and so on.

1868 **Note:** {*COLL_WEIGHTS_MAX*} is defined in detail in `<limits.h>`.

1869 3.106 Column Position

1870 A unit of horizontal measure related to characters in a line.

1871 It is assumed that each character in a character set has an intrinsic column width independent of
1872 any output device. Each printable character in the portable character set has a column width of
1873 one. The standard utilities, when used as described in IEEE Std. 1003.1-200x, assume that all
1874 characters have integral column widths. The column width of a character is not necessarily
1875 related to the internal representation of the character (numbers of bits or bytes).

1876 The column position of a character in a line is defined as one plus the sum of the column widths
1877 of the preceding characters in the line. Column positions are numbered starting from 1.

1878 3.107 Command

1879 A directive to the shell to perform a particular task.

1880 **Note:** Shell Commands are defined in detail in the Shell and Utilities volume of
1881 IEEE Std. 1003.1-200x, Section 2.9, Shell Commands. |

1882 3.108 Command Language Interpreter

1883 An interface that interprets sequences of text input as commands. It may operate on an input
1884 stream or it may interactively prompt and read commands from a terminal. It is possible for
1885 applications to invoke utilities through a number of interfaces, which are collectively considered
1886 to act as command interpreters. The most obvious of these are the *sh* utility and the *system()*
1887 function, although *popen()* and the various forms of *exec* may also be considered to behave as
1888 interpreters.

1889 **Note:** The *sh* utility is defined in detail in the Shell and Utilities volume of
1890 IEEE Std. 1003.1-200x.

1891 The *system()*, *popen()*, and *exec* functions are defined in detail in the System Interfaces
1892 volume of IEEE Std. 1003.1-200x.

1893 3.109 Composite Graphic Symbol

1894 A graphic symbol consisting of a combination of two or more other graphic symbols in a single
1895 character position, such as a diacritical mark and a base character.

1896 3.110 Condition Variable

1897 A synchronization object which allows a thread to suspend execution, repeatedly, until some
1898 associated predicate becomes true. A thread whose execution is suspended on a condition
1899 variable is said to be blocked on the condition variable.

1900 3.111 Connection

1901 An association established between two or more endpoints for the transfer of data

1902 3.112 Connection Mode

1903 The transfer of data in the context of a connection; see also Section 3.113.

1904 3.113 Connectionless Mode

1905 The transfer of data other than in the context of a connection; see also Section 3.112 and Section
1906 3.126 (on page 62).

1907 3.114 Control Character

1908 A character, other than a graphic character, that affects the recording, processing, transmission,
1909 or interpretation of text.

1910 3.115 Control Operator

1911 In the shell command language, a token that performs a control function. It is one of the
1912 following symbols:

1913 & && () ; ;; newline | ||

1914 The end-of-input indicator used internally by the shell is also considered a control operator.

1915 **Note:** Token Recognition is defined in detail in the Shell and Utilities volume of
1916 IEEE Std. 1003.1-200x, Section 2.3, Token Recognition.

1917 3.116 Controlling Process

1918 The session leader that established the connection to the controlling terminal. If the terminal
1919 subsequently ceases to be a controlling terminal for this session, the session leader ceases to be
1920 the controlling process.

1921 3.117 Controlling Terminal

1922 A terminal that is associated with a session. Each session may have at most one controlling
1923 terminal associated with it, and a controlling terminal is associated with exactly one session.
1924 Certain input sequences from the controlling terminal cause signals to be sent to all processes in
1925 the process group associated with the controlling terminal.

1926 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 213).

1927 3.118 Conversion Descriptor

1928 A per-process unique value used to identify an open codeset conversion.

1929 3.119 Core File

1930 A file of unspecified format that may be generated when a process terminates abnormally.

1931 3.120 CPU Time (Execution Time)

1932 The time spent executing a process or thread, including the time spent executing system services
1933 on behalf of that process or thread. If the Threads option is supported, then the value of the
1934 CPU-time clock for a process is implementation-defined. With this definition the sum of all the
1935 execution times of all the threads in a process might not equal the process execution time, even
1936 in a single-threaded process, because implementations may differ in how they account for time
1937 during context switches or for other reasons.

1938 3.121 CPU-Time Clock

1939 A clock that measures the execution time of a particular process or thread.

1940 3.122 CPU-Time Timer

1941 A timer attached to a CPU-time clock.

1942 3.123 Current Job

1943 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities. There
1944 is at most one current job; see also Section 3.205 (on page 76).

1945 3.124 Current Working Directory

1946 See *Working Directory* in Section 3.438 (on page 117).

1947 3.125 Cursor Position

1948 The line and column position on the screen denoted by the terminal's cursor.

1949 3.126 Datagram

1950 A unit of data transferred from one endpoint to another in connectionless mode service.

1951 **3.127 Data Segment**

1952 Memory associated with a process, that can contain dynamically allocated data. |

1953 **3.128 Deferred Batch Service**

1954 A service that is performed as a result of events that are asynchronous with respect to requests. |

1955 **Note:** Once a batch job has been created, it is subject to deferred services.

1956 **3.129 Device**

1957 A computer peripheral or an object that appears to the application as such.

1958 **3.130 Device ID**

1959 A non-negative integer used to identify a device. |

1960 **3.131 Directory**

1961 A file that contains directory entries. No two directory entries in the same directory have the |
1962 same name. |

1963 **3.132 Directory Entry (or Link)**

1964 An object that associates a file name with a file. Several directory entries can associate names
1965 with the same file.

1966 **3.133 Directory Stream**

1967 A sequence of all the directory entries in a particular directory. An open directory stream may be
1968 implemented using a file descriptor.

1969 **3.134 Disarm (a Timer)**

1970 To stop a timer from measuring the passage of time, disabling any future process notifications
1971 (until the timer is armed again).

1972 **3.135 Display**

1973 To output to the user's terminal. If the output is not directed to a terminal, the results are
1974 undefined.

1975 **3.136 Dollar Sign**

1976 The character '\$'.

1977 **3.137 Dot**

1978 In the context of naming files, the file name consisting of a single dot character ('.').

1979 **Note:** In the context of shell special built-in utilities, see *dot* in the Shell and Utilities volume
1980 of IEEE Std. 1003.1-200x, Section 2.15, Special Built-In Utilities.

1981 Path Name Resolution is defined in detail in Section 4.9 (on page 123).

1982 **3.138 Dot-Dot**

1983 The file name consisting solely of two dot characters ("..").

1984 **Note:** Path Name Resolution is defined in detail in Section 4.9 (on page 123).

1985 **3.139 Double-Quote**

1986 The character '"', also known as *quotation-mark*.

1987 **Note:** The *double* adjective in this term refers to the two strokes in the character glyph. |
1988 IEEE Std. 1003.1-200x never uses the term double-quote to refer to two apostrophes |
1989 or quotation marks.

1990 3.140 Downshifting

1991 The conversion of an uppercase character that has a single-character lowercase representation
1992 into this lowercase representation.

1993 3.141 Driver

1994 A module that controls data transferred to and received from devices.

1995 **Note:** Drivers are traditionally written to be a part of the system implementation, although
1996 they are frequently written separately from the writing of the implementation. A
1997 driver may contain processor-specific code, and therefore be non-portable.

1998 3.142 Effective Group ID

1999 An attribute of a process that is used in determining various permissions, including file access
2000 permissions; see also Section 3.190 (on page 73).

2001 3.143 Effective User ID

2002 An attribute of a process that is used in determining various permissions, including file access
2003 permissions; see also Section 3.427 (on page 115).

2004 3.144 Eight-Bit Transparency

2005 The ability of a software component to process 8-bit characters without modifying or utilizing
2006 any part of the character in a way that is inconsistent with the rules of the current coded
2007 character set.

2008 3.145 Empty Directory

2009 A directory that contains, at most, directory entries for dot and dot-dot, and has exactly one link
2010 to it in dot-dot. No other links to the directory may exist. It is unspecified whether an
2011 implementation can ever consider the root directory to be empty.

2012 3.146 Empty Line

2013 A line consisting of only a <newline> character; see also Section 3.76 (on page 53).

2014 3.147 Empty String (or Null String)

2015 A string whose first byte is a null byte.

2016 3.148 Empty Wide-Character String

2017 A wide-character string whose first element is a null wide-character code.

2018 3.149 Encoding Rule

2019 The rules used to convert between wide-character codes and multi-byte character codes.

2020 **Note:** Stream Orientation and Encoding Rules are defined in detail in the System Interfaces
2021 volume of IEEE Std. 1003.1-200x, Section 2.5.2, Stream Orientation and Encoding
2022 Rules.

2023 3.150 Entire Regular Expression

2024 The concatenated set of one or more BREs or EREs that make up the pattern specified for string
2025 selection.

2026 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 195).

2027 3.151 Epoch

2028 The time zero hours, zero minutes, zero seconds, on January 1, 1970 Coordinated Universal
2029 Time.

2030 **Note:** See also Seconds Since the Epoch defined in Section 4.12 (on page 125).

2031 3.152 Equivalence Class

2032 A set of collating elements with the same primary collation weight.

2033 Elements in an equivalence class are typically elements that naturally group together, such as all
2034 accented letters based on the same base letter.

2035 The collation order of elements within an equivalence class is determined by the weights
2036 assigned on any subsequent levels after the primary weight.

2037 3.153 Era

2038 An alternative method for counting and displaying years.

2039 **Note:** The *LC_TIME* category is defined in detail in Section 7.3.5 (on page 168).

2040 3.154 Event Management

2041 The mechanism that enables applications to register for and be made aware of external events
2042 such as data becoming available for reading.

2043 3.155 Executable File

2044 A regular file acceptable as a new process image file by the equivalent of the *exec* family of
2045 functions, and thus usable as one form of a utility. The standard utilities described as compilers
2046 can produce executable files, but other unspecified methods of producing executable files may
2047 also be provided. The internal format of an executable file is unspecified, but a conforming
2048 application cannot assume an executable file is a text file.

2049 3.156 Execute

2050 In the Shell and Utilities volume of IEEE Std. 1003.1-200x, to perform command search and
2051 execution actions; see also Section 3.202 (on page 75).

2052 **Note:** Command Search and Execution is defined in detail in the Shell and Utilities volume
2053 of IEEE Std. 1003.1-200x, Section 2.9.1.1, Command Search and Execution.

2054 3.157 Execution Time

2055 See *CPU Time* in Section 3.120 (on page 62).

2056 3.158 Execution Time Monitoring

2057 A set of execution time monitoring primitives that allow online measuring of thread and process
2058 execution times.

2059 3.159 Expand

2060 In the shell command language, when not qualified, the act of applying word expansions.

2061 **Note:** Work Expansions are defined in detail in the Shell and Utilities volume of
2062 IEEE Std. 1003.1-200x, Section 2.6, Word Expansions.

2063 3.160 Extended Regular Expression (ERE)

2064 A regular expression (see also Section 3.318 (on page 95)) that is an alternative to the Basic
2065 Regular Expression using a more extensive syntax, occasionally used by some utilities.

2066 **Note:** Extended Regular Expressions are described in detail in Section 9.4 (on page 203).

2067 3.161 Extended Security Controls

2068 Implementation-defined security controls allowed by the file access permission and appropriate
2069 privilege (see also Section 3.19 (on page 44)) mechanisms, through which an implementation can
2070 support different security policies from those described in IEEE Std. 1003.1-200x.

2071 **Note:** See also Extended Security Controls defined in Section 4.2 (on page 121).

2072 File Access Permissions are defined in detail in Section 4.3 (on page 121).

2073 3.162 Feature Test Macro

2074 A macro used to determine whether a particular set of features is included from a header.

2075 **Note:** See also the System Interfaces volume of IEEE Std. 1003.1-200x, Section 2.2, The
2076 Compilation Environment.

2077 3.163 Field

2078 In the shell command language, a unit of text that is the result of parameter expansion,
2079 arithmetic expansion, command substitution, or field splitting. During command processing, the
2080 resulting fields are used as the command name and its arguments.

2081 **Note:** Parameter Expansion is defined in detail in the Shell and Utilities volume of
2082 IEEE Std. 1003.1-200x, Section 2.6.2, Parameter Expansion.

2083 Arithmetic Expansion is defined in detail in the Shell and Utilities volume of
2084 IEEE Std. 1003.1-200x, Section 2.6.4, Arithmetic Expansion.

2085 Command Substitution is defined in detail in the Shell and Utilities volume of
2086 IEEE Std. 1003.1-200x, Section 2.6.3, Command Substitution.

2087 Field Splitting is defined in detail in the Shell and Utilities volume of
2088 IEEE Std. 1003.1-200x, Section 2.6.5, Field Splitting.

2089 For further information on command processing, see the Shell and Utilities volume of
2090 IEEE Std. 1003.1-200x, Section 2.9.1, Simple Commands.

2091 3.164 FIFO Special File (or FIFO)

2092 A type of file with the property that data written to such a file is read on a first-in-first-out basis.

2093 **Note:** Other characteristics of FIFOs are described in the System Interfaces volume of
2094 IEEE Std. 1003.1-200x, *lseek()*, *open()*, *read()*, and *write()*.

2095 3.165 File

2096 An object that can be written to, or read from, or both. A file has certain attributes, including
2097 access permissions and type. File types include regular file, character special file, block special
2098 file, FIFO special file, symbolic link, socket, and directory. Other types of files may be supported
2099 by the implementation.

2100 3.166 File Description

2101 See *Open File Description* in Section 3.255 (on page 84).

2102 3.167 File Descriptor

2103 A per-process unique, non-negative integer used to identify an open file for the purpose of file
2104 access. The value of a file descriptor is from zero to {OPEN_MAX}. A process can have no more
2105 than {OPEN_MAX} file descriptors open simultaneously. File descriptors may also be used to
2106 implement message catalog descriptors and directory streams; see also Section 3.255 (on page
2107 84).

2108 **Note:** {OPEN_MAX} is defined in detail in <limits.h>.

2109 3.168 File Group Class

2110 The property of a file indicating access permissions for a process related to the group
2111 identification of a process. A process is in the file group class of a file if the process is not in the
2112 file owner class and if the effective group ID or one of the supplementary group IDs of the
2113 process matches the group ID associated with the file. Other members of the class may be
2114 implementation-defined.

2115 3.169 File Mode

2116 An object containing the *file mode bits* and file type of a file.

2117 **Note:** File mode bits and file types are defined in detail in `<sys/stat.h>`.

2118 3.170 File Mode Bits

2119 A file's file permission bits, set-user-ID-on-execution bit (S_ISUID), and set-group-ID-on-
2120 execution bit (S_ISGID).

2121 **Note:** File Mode Bits are defined in detail in `<sys/stat.h>`.

2122 3.171 File Name

2123 A name consisting of 1 to {NAME_MAX} bytes used to name a file. The characters composing
2124 the name may be selected from the set of all character values excluding the slash character and
2125 the null byte. The file names dot and dot-dot have special meaning. A file name is sometimes
2126 referred to as a *path name component*.

2127 **Note:** Path Name Resolution is defined in detail in Section 4.9 (on page 123).

2128 3.172 File Name Portability

2129 File names should be constructed from the portable file name character set because the use of
2130 other characters can be confusing or ambiguous in certain contexts. (For example, the use of a
2131 colon (':') in a path name could cause ambiguity if that path name were included in a *PATH*
2132 definition.)

2133 3.173 File Offset

2134 The byte position in the file where the next I/O operation begins. Each open file description
2135 associated with a regular file, block special file, or directory has a file offset. A character special
2136 file that does not refer to a terminal device may have a file offset. There is no file offset specified
2137 for a pipe or FIFO.

2138 3.174 File Other Class

2139 The property of a file indicating access permissions for a process related to the user and group
2140 identification of a process. A process is in the file other class of a file if the process is not in the
2141 file owner class or file group class.

2142 3.175 File Owner Class

2143 The property of a file indicating access permissions for a process related to the user
2144 identification of a process. A process is in the file owner class of a file if the effective user ID of
2145 the process matches the user ID of the file.

2146 3.176 File Permission Bits

2147 Information about a file that is used, along with other information, to determine whether a
2148 process has read, write, or execute/search permission to a file. The bits are divided into three
2149 parts: owner, group, and other. Each part is used with the corresponding file class of processes.
2150 These bits are contained in the file mode.

2151 **Note:** File modes are defined in detail in `<sys/stat.h>`.

2152 File Access Permissions are defined in detail in Section 4.3 (on page 121).

2153 3.177 File Serial Number

2154 A per-file system unique identifier for a file.

2155 3.178 File System

2156 A collection of files and certain of their attributes. It provides a name space for file serial
2157 numbers referring to those files.

2158 3.179 File Type

2159 See *File* in Section 3.165 (on page 69).

2160 3.180 Filter

2161 A command whose operation consists of reading data from standard input or a list of input files
2162 and writing data to standard output. Typically, its function is to perform some transformation
2163 on the data stream.

2164 3.181 First Open (of a File)

2165 When a process opens a file that is not currently an open file within any process.

2166 3.182 Flow Control

2167 The mechanism employed by a communications provider that constrains a sending entity to
2168 wait until the receiving entities can safely receive additional data without loss.

2169 3.183 Foreground Job

2170 See *Foreground Process Group* in Section 3.185.

2171 3.184 Foreground Process

2172 A process that is a member of a foreground process group.

2173 3.185 Foreground Process Group (or Foreground Job)

2174 A process group whose member processes have certain privileges, denied to processes in
2175 background process groups, when accessing their controlling terminal. Each session that has
2176 established a connection with a controlling terminal has at most one process group of the session
2177 as the foreground process group of that controlling terminal.

2178 **Note:** The General Terminal Interface is defined in detail in Chapter 11.

2179 3.186 Foreground Process Group ID

2180 The process group ID of the foreground process group.

2181 3.187 Form-Feed Character (<form-feed>)

2182 A character that in the output stream indicates that printing should start on the next page of an
2183 output device. The <form-feed> character is the character designated by '\f' in the C language.
2184 If the <form-feed> character is not the first character of an output line, the result is unspecified.
2185 It is unspecified whether this character is the exact sequence transmitted to an output device by
2186 the system to accomplish the movement to the next page.

2187 3.188 Graphic Character

2188 A member of the **graph** character class of the current locale.

2189 **Note:** The **graph** character class is defined in detail in Section 7.3.1 (on page 147).

2190 3.189 Group Database

2191 A system database of implementation-defined format that contains at least the following
2192 information for each group ID:

- 2193 • Group name
- 2194 • Numerical group ID
- 2195 • List of users allowed in the group

2196 The list of users allowed in the group is used by the *newgrp* utility.

2197 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of
2198 IEEE Std. 1003.1-200x.

2199 3.190 Group ID

2200 A non-negative integer, which can be contained in an object of type **gid_t**, that is used to identify
2201 a group of system users. Each system user is a member of at least one group. When the identity
2202 of a group is associated with a process, a group ID value is referred to as a real group ID, an
2203 effective group ID, one of the supplementary group IDs, or a saved set-group-ID.

2204 3.191 Group Name

2205 A string that is used to identify a group; see also Section 3.189. To be portable across conforming
2206 systems, the value is composed of characters from the portable file name character set. The
2207 hyphen should not be used as the first character of a portable group name.

2208 3.192 Hard Limit

2209 A system resource limitation that may be reset to a lesser or greater limit by a privileged process.
2210 A non-privileged process is restricted to only lowering its hard limit.

2211 3.193 Hard Link

2212 The relationship between two directory entries that represent the same file; see also Section 3.132
2213 (on page 63). The result of an execution of the *ln* utility (without the *-s* option) or the *link()*
2214 function. This term is contrasted against symbolic link; see also Section 3.374 (on page 104).

2215 3.194 Home Directory

2216 The directory specified by the *HOME* environment variable.

2217 3.195 Host Byte Order

2218 The arrangement of bytes in any **int** type when using a specific machine architecture.

2219 **Note:** Two common methods of byte ordering are big-endian and little-endian. Big-endian
2220 is a format for storage of binary data in which the most significant byte is placed first,
2221 with the rest in descending order. Little-endian is a format for storage or
2222 transmission of binary data in which the least significant byte is placed first, with the
2223 rest in ascending order.

2224 3.196 Incomplete Line

2225 A sequence of one or more non-`<newline>` characters at the end of the file.

2226 3.197 Inf

2227 A value representing infinity that can be stored in a floating type. Not all systems support the
2228 Inf value.

2229 3.198 Instrumented Application

2230 An application that contains at least one call to the trace point function *posix_trace_event()*. Each
2231 process of an instrumented application has a mapping of trace event names to trace event type
2232 identifiers. This mapping is used by the trace stream that is created for that process.

2233 3.199 Interactive Shell

2234 A processing mode of the shell that is suitable for direct user interaction.

2235 3.200 Internationalization

2236 The provision within a computer program of the capability of making itself adaptable to the
2237 requirements of different native languages, local customs, and coded character sets.

2238 3.201 Interprocess Communication

2239 A functionality enhancement to add a high-performance, deterministic interprocess
2240 communication facility for local communication.

2241 3.202 Invoke

2242 To perform command search and execution actions, except that searching for shell functions and
2243 special built-in utilities is suppressed; see also Section 3.156 (on page 67).

2244 **Note:** Command Search and Execution is defined in detail in the Shell and Utilities volume
2245 of IEEE Std. 1003.1-200x, Section 2.9.1.1, Command Search and Execution.

2246 3.203 Job

2247 A set of processes, comprising a shell pipeline, and any processes descended from it, that are all
2248 in the same process group.

2249 **Note:** See also the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.9.2,
2250 Pipelines.

2251 **3.204 Job Control**

2252 A facility that allows users selectively to stop (suspend) the execution of processes and continue
 2253 (resume) their execution at a later point. The user typically employs this facility via the
 2254 interactive interface jointly supplied by the terminal I/O driver and a command interpreter.

2255 **3.205 Job Control Job ID**

2256 A handle that is used to refer to a job. The job control job ID can be any of the forms shown in the
 2257 following table:

2258 **Table 3-1** Job Control Job ID Formats

2259 2260	Job Control Job ID	Meaning
2261	%%	Current job.
2262	%+	Current job.
2263	%-	Previous job.
2264	% <i>n</i>	Job number <i>n</i> .
2265	% <i>string</i>	Job whose command begins with <i>string</i> .
2266	%? <i>string</i>	Job whose command contains <i>string</i> .

2267 **3.206 Last Close (of a File)**

2268 When a process closes a file, resulting in the file not being an open file within any process.

2269 **3.207 Line**

2270 A sequence of zero or more non-<newline> characters plus a terminating <newline> character.

2271 **3.208 Linger**

2272 Wait for a period of time before terminating a connection, to allow outstanding data to be
 2273 transferred.

2274 **3.209 Link**

2275 See *Directory Entry* in Section 3.132 (on page 63).

2276 **3.210 Link Count**

2277 The number of directory entries that refer to a particular file.

2278 **3.211 Local Customs**

2279 The conventions of a geographical area or territory for such things as date, time, and currency
2280 formats.

2281 **3.212 Local Interprocess Communication (Local IPC)**

2282 The transfer of data between processes in the same system.

2283 **3.213 Locale**

2284 The definition of the subset of a user's environment that depends on language and cultural
2285 conventions.

2286 **Note:** Locales are defined in detail in Chapter 7 (on page 143).

2287 **3.214 Localization**

2288 The process of establishing information within a computer system specific to the operation of
2289 particular native languages, local customs, and coded character sets.

2290 **3.215 Login**

2291 The unspecified activity by which a user gains access to the system. Each login is associated
2292 with exactly one login name.

2293 3.216 Login Name

2294 A user name that is associated with a login.

2295 3.217 Map

2296 To create an association between a page-aligned range of the address space of a process and
2297 some memory object, such that a reference to an address in that range of the address space
2298 results in a reference to the associated memory object. The mapped memory object is not
2299 necessarily memory-resident.

2300 3.218 Marked Message

2301 A STREAMS message on which a certain flag is set. Marking a message gives the application
2302 protocol-specific information. An application can use *ioctl()* to determine whether a given
2303 message is marked.

2304 **Note:** The *ioctl()* function is defined in detail in the System Interfaces volume of
2305 IEEE Std. 1003.1-200x.

2306 3.219 Matched

2307 A state applying to a sequence of zero or more characters when the characters in the sequence
2308 correspond to a sequence of characters defined by a BRE or ERE pattern.

2309 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 195).

2310 3.220 Memory Mapped Files and Shared Memory Objects

2311 A performance improvement facility to allow for programs to access files as part of the address
2312 space and for separate application programs to have portions of their address space commonly
2313 accessible.

2314 3.221 Memory Object

2315 One of:

- 2316 • A file
- 2317 • A shared memory object
- 2318 • A typed memory object

2319 When used in conjunction with *mmap()*, a memory object appears in the address space of the
2320 calling process.

2321 **Note:** The *mmap()* function is defined in detail in the System Interfaces volume of
2322 IEEE Std. 1003.1-200x.

2323 3.222 Memory-Resident

2324 Managed by the implementation in such a way as to provide an upper bound on memory access
2325 times.

2326 3.223 Message

2327 In the context of programmatic message passing, information that can be transferred between
2328 processes or threads by being added to and removed from a message queue. A message consists
2329 of a fixed-size message buffer.

2330 3.224 Message Catalog

2331 In the context of providing natural language messages to the user, a file or storage area
2332 containing program messages, command prompts, and responses to prompts for a particular
2333 native language, territory, and codeset.

2334 3.225 Message Catalog Descriptor

2335 In the context of providing natural language messages to the user, a per-process unique value
2336 used to identify an open message catalog. A message catalog descriptor may be implemented
2337 using a file descriptor.

2338 3.226 Message Queue

2339 In the context of programmatic message passing, an object to which messages can be added and
2340 removed. Messages may be removed in the order in which they were added or in priority order.

2341 3.227 Mode

2342 A collection of attributes that specifies a file's type and its access permissions.

2343 **Note:** File Access Permissions are defined in detail in Section 4.3 (on page 121).

2344 3.228 Monotonic Clock

2345 A clock whose value cannot be set via `clock_settime()` and which cannot have negative clock
2346 jumps.

2347 3.229 Mount Point

2348 Either the system root directory or a directory for which the `st_dev` field of structure `stat` differs
2349 from that of its parent directory.

2350 **Note:** The `stat` structure is defined in detail in `<sys/stat.h>`.

2351 3.230 Multi-Character Collating Element

2352 A sequence of two or more characters that collate as an entity. For example, in some coded
2353 character sets, an accented character is represented by a non-spacing accent, followed by the
2354 letter. Other examples are the Spanish elements *ch* and *ll*.

2355 3.231 Mutex

2356 A synchronization object used to allow multiple threads to serialize their access to shared data.
2357 The name derives from the capability it provides; namely, mutual-exclusion. The thread that has
2358 locked a mutex becomes its owner and remains the owner until that same thread unlocks the
2359 mutex.

2360 3.232 Name

2361 In the shell command language, a word consisting solely of underscores, digits, and alphabetic
2362 from the portable character set. The first character of a name is not a digit.

2363 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 133).

2364 3.233 Named STREAM

2365 A STREAMS-based file descriptor that is attached to a name in the file system name space. All
2366 subsequent operations on the named STREAM act on the STREAM that was associated with the
2367 file descriptor until the name is disassociated from the STREAM.

2368 3.234 NaN (Not a Number)

2369 A value that can be stored in a floating type but that is not a valid floating point number. Not all
2370 systems support the NaN value.

2371 3.235 Native Language

2372 A computer user's spoken or written language, such as American English, British English,
2373 Danish, Dutch, French, German, Italian, Japanese, Norwegian, or Swedish.

2374 3.236 Negative Response

2375 An input string that matches one of the responses acceptable to the *LC_MESSAGES* category
2376 keyword **noexpr**, matching an extended regular expression in the current locale.

2377 **Note:** The *LC_MESSAGES* category is defined in detail in Section 7.3.6 (on page 174).

2378 3.237 Network

2379 A collection of interconnected hosts.

2380 **Note:** The term network in IEEE Std. 1003.1-200x is used to refer to the network of hosts.
2381 The term batch system is used to refer to the network of batch servers.

2382 3.238 Network Address

2383 A network-visible identifier used to designate specific endpoints in a network. Specific
2384 endpoints on host systems have addresses, and host systems may also have addresses.

2385 3.239 Network Byte Order

2386 The way of representing any **int** type such that, when transmitted over a network via a network
2387 endpoint, the **int** type is transmitted as an appropriate number of octets with the most
2388 significant octet first, followed by any other octets in descending order of significance.

2389 **Note:** This order is more commonly known as big-endian ordering.

2390 3.240 Newline Character (<newline>)

2391 A character that in the output stream indicates that printing should start at the beginning of the
2392 next line. The <newline> character is the character designated by ‘\n’ in the C language. It is
2393 unspecified whether this character is the exact sequence transmitted to an output device by the
2394 system to accomplish the movement to the next line.

2395 3.241 Nice Value

2396 A number used as advice to the system to alter process scheduling. Numerically smaller values
2397 give a process additional preference when scheduling a process to run. Numerically larger
2398 values reduce the preference and make a process less likely to run. Typically, a process with a
2399 smaller nice value runs to completion more quickly than an equivalent process with a higher
2400 nice value. The symbol {NZERO} specifies the default nice value of the system.

2401 3.242 Non-Blocking

2402 A property of an open file description that causes it to either perform the requested action or
2403 return an indication that the action could not be immediately performed, in either case returning
2404 without delay (other than normal scheduling delays) from the call.

2405 **Note:** The exact semantics are dependent on the type of file associated with the open file.
2406 For data reads from devices such as ttys and FIFOs, a successful return usually
2407 indicates that data sufficient to satisfy the read was immediately available. Similarly,
2408 for writes, that space to perform (at least part of) the write was available, and for
2409 networking not to await protocol events (for example, acknowledgements) to occur.

2410 3.243 Non-Spacing Characters

2411 A character, such as a character representing a diacritical mark in the ISO/IEC 6937:1994
2412 standard coded character set, which is used in combination with other characters to form
2413 composite graphic symbols.

2414 3.244 NUL

2415 A character with all bits set to zero.

2416 **3.245 Null Byte**

2417 A byte with all bits set to zero.

2418 **3.246 Null Pointer**

2419 The value that is obtained by converting the number 0 into a pointer; for example, **(void *) 0**. The
2420 C language guarantees that this value does not match that of any legitimate pointer, so it is used
2421 by many functions that return pointers to indicate an error.

2422 **3.247 Null String**2423 See *Empty String* in Section 3.147 (on page 66).2424 **3.248 Null Wide-Character Code**

2425 A wide-character code with all bits set to zero.

2426 **3.249 Number Sign**2427 The character '#', also known as *hash sign*.2428 **3.250 Object File**

2429 A regular file containing the output of a compiler, formatted as input to a linkage editor for
2430 linking with other object files into an executable form. The methods of linking are unspecified
2431 and may involve the dynamic linking of objects at runtime. The internal format of an object file
2432 is unspecified, but a conforming application cannot assume an object file is a text file.

2433 **3.251 Octet**

2434 Unit of data representation that consists of eight contiguous bits.

2435 3.252 Offset Maximum

2436 An attribute of an open file description representing the largest value that can be used as a file
2437 offset.

2438 3.253 Opaque Address

2439 An address such that the entity making use of it requires no details about its contents or format.

2440 3.254 Open File

2441 A file that is currently associated with a file descriptor.

2442 3.255 Open File Description

2443 A record of how a process or group of processes is accessing a file. Each file descriptor refers to
2444 exactly one open file description, but an open file description can be referred to by more than
2445 one file descriptor. A file offset, file status, and file access modes are attributes of an open file
2446 description.

2447 3.256 Operand

2448 An argument to a command that is generally used as an object supplying information to a utility
2449 necessary to complete its processing. Operands generally follow the options in a command line.

2450 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 227).

2451 3.257 Operator

2452 In the shell command language, either a control operator or a redirection operator.

2453 3.258 Option

2454 An argument to a command that is generally used to specify changes in the utility's default
2455 behavior.

2456 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 227).

2457 3.259 Option-Argument

2458 A parameter that follows certain options. In some cases an option-argument is included within
2459 the same argument string as the option—in most cases it is the next argument.

2460 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 227).

2461 3.260 Orientation

2462 A stream has one of three orientations: unoriented, byte-oriented, or wide-oriented.

2463 **Note:** For further information, see the System Interfaces volume of IEEE Std. 1003.1-200x,
2464 Section 2.5.2, Stream Orientation and Encoding Rules.

2465 3.261 Orphaned Process Group

2466 A process group in which the parent of every member is either itself a member of the group or is
2467 not a member of the group's session.

2468 3.262 Page

2469 The granularity of process memory mapping or locking.

2470 Physical memory and memory objects can be mapped into the address space of a process on
2471 page boundaries and in integral multiples of pages. Process address space can be locked into
2472 memory (made memory-resident) on page boundaries and in integral multiples of pages.

2473 3.263 Page Size

2474 The size, in bytes, of the system unit of memory allocation, protection, and mapping. On systems
2475 that have segment rather than page-based memory architectures, the term page means a
2476 segment.

2477 3.264 Parameter

2478 In the shell command language, an entity that stores values. There are three types of parameters:
2479 variables (named parameters), positional parameters, and special parameters. Parameter
2480 expansion is accomplished by introducing a parameter with the '\$' character.

2481 **Note:** See also the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.5,
2482 Parameters and Variables.

2483 In the C language, an object declared as part of a function declaration or definition that acquires
2484 a value on entry to the function, or an identifier following the macro name in a function-like
2485 macro definition.

2486 3.265 Parent Directory

2487 When discussing a given directory, the directory that both contains a directory entry for the
2488 given directory and is represented by the path name dot-dot in the given directory.

2489 When discussing other types of files, a directory containing a directory entry for the file under
2490 discussion.

2491 This concept does not apply to dot and dot-dot.

2492 3.266 Parent Process

2493 The process which created (or inherited) the process under discussion.

2494 3.267 Parent Process ID

2495 An attribute of a new process identifying the parent of the process. The parent process ID of a
2496 process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime
2497 has ended, the parent process ID is the process ID of an implementation-defined system process.

2498 3.268 Path Name

2499 A character string that is used to identify a file. In the context of IEEE Std. 1003.1-200x, a path
2500 name consists of, at most, {PATH_MAX} bytes, including the terminating null byte. It has an
2501 optional beginning slash, followed by zero or more file names separated by slashes. A path name
2502 may optionally contain one or more trailing slashes. Multiple successive slashes are considered
2503 to be the same as one slash.

2504 **Note:** Path Name Resolution is defined in detail in Section 4.9 (on page 123).

2505 3.269 Path Name Component

2506 See *File Name* in Section 3.171 (on page 70).

2507 3.270 Path Prefix

2508 A path name, with an optional ending slash, that refers to a directory.

2509 3.271 Pattern

2510 A sequence of characters used either with regular expression notation or for path name
2511 expansion, as a means of selecting various character strings or path names, respectively.

2512 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 195).

2513 See also the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.6.6, Path
2514 Name Expansion.

2515 The syntaxes of the two types of patterns are similar, but not identical; IEEE Std. 1003.1-200x
2516 always indicates the type of pattern being referred to in the immediate context of the use of the
2517 term.

2518 3.272 Period

2519 The character ' . '. The term period is contrasted with dot (see also Section 3.137 (on page 64)),
2520 which is used to describe a specific directory entry.

2521 3.273 Permissions

2522 Attributes of an object that determine the privilege necessary to access or manipulate the object.

2523 **Note:** File Access Permissions are defined in detail in Section 4.3 (on page 121).

2524 3.274 Persistence

2525 A mode for semaphores, shared memory, and message queues requiring that the object and its
2526 state (including data, if any) are preserved after the object is no longer referenced by any process.

2527 Persistence of an object does not imply that the state of the object is maintained across a system
2528 crash or a system reboot.

2529 3.275 Pipe

2530 An object accessed by one of the pair of file descriptors created by the *pipe()* function. Once
2531 created, the file descriptors can be used to manipulate it, and it behaves identically to a FIFO
2532 special file when accessed in this way. It has no name in the file hierarchy.

2533 **Note:** The *pipe()* function is defined in detail in the System Interfaces volume of
2534 IEEE Std. 1003.1-200x.

2535 **3.276 Polling**

2536 A scheduling scheme whereby the local process periodically checks until the prespecified events
2537 (for example, read, write) have occurred.

2538 **3.277 Portable Character Set**

2539 The collection of characters that are required to be present in all locales supported by
2540 conforming systems.

2541 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 133).

2542 This term is contrasted against the smaller *portable file name character set*; see also Section 3.278.

2543 **3.278 Portable File Name Character Set**

2544 The set of characters from which portable file names are constructed.

2545 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

2546 a b c d e f g h i j k l m n o p q r s t u v w x y z

2547 0 1 2 3 4 5 6 7 8 9 . _ -

2548 The last three characters are the period, underscore, and hyphen characters, respectively.

2549 **3.279 Positional Parameter**

2550 In the shell command language, a parameter denoted by a single digit or one or more digits in
2551 curly braces.

2552 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
2553 Section 2.5.1, Positional Parameters.

2554 **3.280 Preallocation**

2555 The reservation of resources in a system for a particular use.

2556 Preallocation does not imply that the resources are immediately allocated to that use, but merely
2557 indicates that they are guaranteed to be available in bounded time when needed.

2558 3.281 Preempted Process (or Thread)

2559 A running thread whose execution is suspended due to another thread becoming runnable at a
2560 higher priority.

2561 3.282 Previous Job

2562 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities if the
2563 current job exits. There is at most one previous job; see also Section 3.205 (on page 76).

2564 3.283 Printable Character

2565 One of the characters included in the **print** character classification of the *LC_CTYPE* category in
2566 the current locale.

2567 **Note:** The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 147).

2568 3.284 Printable File

2569 A text file consisting only of the characters included in the **print** and **space** character
2570 classifications of the *LC_CTYPE* category and the <backspace> character, all in the current locale.

2571 **Note:** The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 147).

2572 3.285 Priority

2573 A non-negative integer associated with processes or threads whose value is constrained to a
2574 range defined by the applicable scheduling policy. Numerically higher values represent higher
2575 priorities.

2576 3.286 Priority Band

2577 The queuing order applied to normal priority STREAMS messages. High priority STREAMS
2578 messages are not grouped by priority bands. The only differentiation made by the STREAMS
2579 mechanism is between zero and non-zero bands, but specific protocol modules may differentiate
2580 between priority bands.

2581 3.287 Priority Inversion

2582 A condition in which a thread that is not voluntarily suspended (waiting for an event or time
2583 delay) is not running while a lower priority thread is running. Such blocking of the higher
2584 priority thread is often caused by contention for a shared resource.

2585 3.288 Priority Scheduling

2586 A performance and determinism improvement facility to allow applications to determine the
2587 order in which threads that are ready to run are granted access to processor resources.

2588 3.289 Priority-Based Scheduling

2589 Scheduling in which the selection of a running thread is determined by the priorities of the
2590 runnable processes or threads.

2591 3.290 Privilege

2592 See *Appropriate Privileges* in Section 3.19 (on page 44).

2593 3.291 Process

2594 An address space with one or more threads executing within that address space, and the
2595 required system resources for those threads.

2596 **Note:** Many of the system resources defined by IEEE Std. 1003.1-200x are shared among all
2597 of the threads within a process. These include the process ID, the parent process ID,
2598 process group ID, session membership, real, effective, and saved-set user ID, real,
2599 effective, and saved-set group ID, supplementary group IDs, current working
2600 directory, root directory, file mode creation mask, and file descriptors.

2601 3.292 Process Group

2602 A collection of processes that permits the signaling of related processes. Each process in the
2603 system is a member of a process group that is identified by a process group ID. A newly created
2604 process joins the process group of its creator.

2605 3.293 Process Group ID

2606 The unique positive integer identifier representing a process group during its lifetime.

2607 **Note:** See also Process Group ID Reuse defined in Section 4.10 (on page 124).

2608 3.294 Process Group Leader

2609 A process whose process ID is the same as its process group ID.

2610 3.295 Process Group Lifetime

2611 A period of time that begins when a process group is created and ends when the last remaining
2612 process in the group leaves the group, due either to the end of the last process' lifetime or to the
2613 last remaining process calling the *setsid()* or *setpgid()* functions.

2614 **Note:** The *setsid()* and *setpgid()* functions are defined in detail in the System Interfaces
2615 volume of IEEE Std. 1003.1-200x.

2616 3.296 Process ID

2617 The unique positive integer identifier representing a process during its lifetime.

2618 **Note:** See also Process ID Reuse defined in Section 4.10 (on page 124).

2619 3.297 Process Lifetime

2620 The period of time that begins when a process is created and ends when its process ID is
2621 returned to the system. After a process is created with a *fork()* function, it is considered active.
2622 At least one thread of control and address space exist until it terminates. It then enters an
2623 inactive state where certain resources may be returned to the system, although some resources,
2624 such as the process ID, are still in use. When another process executes a *wait()*, *waitid()*, or
2625 *waitpid()* function for an inactive process, the remaining resources are returned to the system.
2626 The last resource to be returned to the system is the process ID. At this time, the lifetime of the
2627 process ends.

2628 **Note:** The *fork()*, *wait()*, *waitid()*, and *waitpid()* functions are defined in detail in the System
2629 Interfaces volume of IEEE Std. 1003.1-200x.

2630 3.298 Process Memory Locking

2631 A performance improvement facility to bind application programs into the high-performance
2632 random access memory of a computer system. This avoids potential latencies introduced by the
2633 operating system in storing parts of a program that were not recently referenced on secondary
2634 memory devices.

2635 3.299 Process Termination

2636 There are two kinds of process termination:

- 2637 1. Normal termination occurs by a return from *main()* or when requested with the *exit()* or
2638 *_exit()* functions.
- 2639 2. Abnormal termination occurs when requested by the *abort()* function or when some
2640 signals are received.

2641 **Note:** The *_exit()*, *abort()*, and *exit()* functions are defined in detail in the System Interfaces
2642 volume of IEEE Std. 1003.1-200x.

2643 3.300 Process-To-Process Communication

2644 The transfer of data between processes.

2645 3.301 Process Virtual Time

2646 The measurement of time in units elapsed by the system clock while a process is executing. |

2647 3.302 Program

2648 A prepared sequence of instructions to the system to accomplish a defined task. The term |
2649 program in IEEE Std. 1003.1-200x encompasses applications written in the Shell Command |
2650 Language, complex utility input languages (for example, *awk*, *lex*, *sed*, and so on), and high-level |
2651 languages.

2652 3.303 Protocol

2653 A set of semantic and syntactic rules for exchanging information.

2654 3.304 Pseudo-Terminal

2655 A pseudo-terminal provides the process with an interface that is identical to the terminal
2656 subsystem. A pseudo-terminal is composed of two devices: the *master device* and a *slave device*.
2657 The slave device provides processes with an interface that is identical to the terminal interface,
2658 although there need not be hardware behind that interface. Anything written on the master
2659 device is presented to the slave as an input and anything written on the slave device is presented
2660 as an input on the master side.

2661 3.305 Radix Character

2662 The character that separates the integer part of a number from the fractional part.

2663 3.306 Read-Only File System

2664 A file system that has implementation-defined characteristics restricting modifications.

2665 **Note:** File Times Update is described in detail in Section 4.6 (on page 122).

2666 3.307 Read-Write Lock

2667 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous
2668 read-only access to data while allowing only one thread to have write access at any given time.
2669 They are typically used to protect data that is read-only more frequently than it is changed.

2670 Read-write locks can be used to synchronize threads in the current process and other processes if
2671 they are allocated in memory that is writable and shared among the cooperating processes and
2672 have been initialized for this behavior.

2673 3.308 Real Group ID

2674 The attribute of a process that, at the time of process creation, identifies the group of the user
2675 who created the process; see also Section 3.190 (on page 73).

2676 3.309 Real Time

2677 Time measured as total units elapsed by the system clock without regard to which thread is
2678 executing.

2679 **3.310 Realtime Signal Extension**

2680 A determinism improvement facility to enable asynchronous signal notifications to an
 2681 application to be queued without impacting compatibility with the existing signal functions.

2682 **3.311 Real User ID**

2683 The attribute of a process that, at the time of process creation, identifies the user who created the
 2684 process; see also Section 3.427 (on page 115).

2685 **3.312 Record**

2686 A collection of related data units or words which is treated as a unit.

2687 **3.313 Redirection**

2688 In the shell command language, a method of associating files with the input or output of
 2689 commands.

2690 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
 2691 Section 2.7, Redirection.

2692 **3.314 Redirection Operator**

2693 In the shell command language, a token that performs a redirection function. It is one of the
 2694 following symbols:

2695 < > >| << >> <& >& <<- <>

2696 **3.315 Reentrant Function**

2697 A function whose effect, when called by two or more threads, is guaranteed to be as if the
 2698 threads each executed the function one after another in an undefined order, even if the actual
 2699 execution is interleaved.

2700 3.316 Referenced Shared Memory Object

2701 A shared memory object that is open or has one or more mappings defined on it.

2702 3.317 Refresh

2703 To ensure that the information on the user's terminal screen is up-to-date.

2704 3.318 Regular Expression

2705 A pattern that selects specific strings from a set of character strings.

2706 **Note:** Regular Expressions are described in detail in Chapter 9 (on page 195).

2707 3.319 Region

2708 In the context of the address space of a process, a sequence of addresses.

2709 In the context of a file, a sequence of offsets.

2710 3.320 Regular File

2711 A file that is a randomly accessible sequence of bytes, with no further structure imposed by the
2712 system.

2713 3.321 Relative Path Name

2714 A path name not beginning with a slash.

2715 **Note:** Path Name Resolution is defined in detail in Section 4.9 (on page 123).

2716 3.322 Relocatable File

2717 A file holding code or data suitable for linking with other object files to create an executable or a
2718 shared object file.

2719 3.323 Relocation

2720 The process of connecting symbolic references with symbolic definitions. For example, when a
2721 program calls a function, the associated call instruction transfers control to the proper
2722 destination address at execution.

2723 3.324 Requested Batch Service

2724 A service that is either rejected or performed prior to a response from the service to the
2725 requester.

2726 3.325 (Time) Resolution

2727 The minimum time interval that a clock can measure or whose passage a timer can detect.

2728 3.326 Root Directory

2729 A directory, associated with a process, that is used in path name resolution for path names that
2730 begin with a slash.

2731 3.327 Runnable Process (or Thread)

2732 A thread that is capable of being a running thread, but for which no processor is available.

2733 3.328 Running Process (or Thread)

2734 A thread currently executing on a processor. On multi-processor systems there may be more
2735 than one such thread in a system at a time.

2736 3.329 Saved Resource Limits

2737 An attribute of a process that provides some flexibility in the handling of unrepresentable
2738 resource limits, as described in the *exec* family of functions and *setrlimit()*.

2739 **Note:** The *exec* and *setrlimit()* functions are defined in detail in the System Interfaces
2740 volume of IEEE Std. 1003.1-200x.

2741 3.330 Saved Set-Group-ID

2742 An attribute of a process that allows some flexibility in the assignment of the effective group ID
2743 attribute, as described in the *exec* family of functions and *setgid()*.

2744 **Note:** The *exec* and *setgid()* functions are defined in detail in the System Interfaces volume
2745 of IEEE Std. 1003.1-200x.

2746 3.331 Saved Set-User-ID

2747 An attribute of a process that allows some flexibility in the assignment of the effective user ID
2748 attribute, as described in the *exec* family of functions and *setuid()*.

2749 **Note:** The *exec* and *setuid()* functions are defined in detail in the System Interfaces volume
2750 of IEEE Std. 1003.1-200x.

2751 3.332 Scheduling

2752 The application of a policy to select a runnable process or thread to become a running process or
2753 thread, or to alter one or more of the thread lists.

2754 3.333 Scheduling Allocation Domain

2755 The set of processors on which an individual thread can be scheduled at any given time.

2756 3.334 Scheduling Contention Scope

2757 A property of a thread that defines the set of threads against which that thread competes for
2758 resources.

2759 For example, in a scheduling decision, threads sharing scheduling contention scope compete for
2760 processor resources. In IEEE Std. 1003.1-200x, a thread has scheduling contention scope of either
2761 PTHREAD_SCOPE_SYSTEM or PTHREAD_SCOPE_PROCESS.

2762 3.335 Scheduling Policy

2763 A set of rules that is used to determine the order of execution of processes or threads to achieve
2764 some goal.

2765 **Note:** Scheduling Policy is defined in detail in Section 4.11 (on page 125).

2766 3.336 Screen

2767 A rectangular region of columns and lines on a terminal display. A screen may be a portion of a
2768 physical display device or may occupy the entire physical area of the display device.

2769 3.337 Scroll

2770 To move the representation of data vertically or horizontally relative to the terminal screen.
2771 There are two types of scrolling:

- 2772 1. The cursor moves with the data.
- 2773 2. The cursor remains stationary while the data moves.

2774 3.338 Semaphore

2775 A minimum synchronization primitive to serve as a basis for more complex synchronization
2776 mechanisms to be defined by the application program.

2777 **Note:** Semaphores are defined in detail in Section 4.13 (on page 126).

2778 3.339 Session

2779 A collection of process groups established for job control purposes. Each process group is a
2780 member of a session. A process is considered to be a member of the session of which its process
2781 group is a member. A newly created process joins the session of its creator. A process can alter
2782 its session membership; see *setsid()*. There can be multiple process groups in the same session.

2783 **Note:** The *setsid()* function is defined in detail in the System Interfaces volume of
2784 IEEE Std. 1003.1-200x.

2785 3.340 Session Leader

2786 A process that has created a session.

2787 **Note:** For further information, see the *setsid()* function defined in the System Interfaces
2788 volume of IEEE Std. 1003.1-200x.

2789 3.341 Session Lifetime

2790 The period between when a session is created and the end of the lifetime of all the process
2791 groups that remain as members of the session.

2792 3.342 Shared Memory Object

2793 An object that represents memory that can be mapped concurrently into the address space of
2794 more than one process.

2795 3.343 Shell

2796 A program that interprets sequences of text input as commands. It may operate on an input
2797 stream or it may interactively prompt and read commands from a terminal.

2798 3.344 Shell, the

2799 The Shell Command Language Interpreter; a specific instance of a shell.

2800 **Note:** For further information, see the *sh* utility defined in the Shell and Utilities volume of
2801 IEEE Std. 1003.1-200x.

2802 3.345 Shell Script

2803 A file containing shell commands. If the file is made executable, it can be executed by specifying
2804 its name as a simple command. Execution of a shell script causes a shell to execute the
2805 commands within the script. Alternatively, a shell can be requested to execute the commands in
2806 a shell script by specifying the name of the shell script as the operand to the *sh* utility.

2807 **Note:** Simple Commands are defined in detail in the Shell and Utilities volume of
2808 IEEE Std. 1003.1-200x, Section 2.9.1, Simple Commands.

2809 The *sh* utility is defined in detail in the Shell and Utilities volume of
2810 IEEE Std. 1003.1-200x.

2811 3.346 Signal

2812 A mechanism by which a process or thread may be notified of, or affected by, an event occurring
2813 in the system. Examples of such events include hardware exceptions and specific actions by
2814 processes. The term signal is also used to refer to the event itself.

2815 3.347 Signal Stack

2816 Memory established for a thread, in which signal handlers catching signals sent to that thread
2817 are executed.

2818 3.348 Single-Quote

2819 The character ' ' ', also known as *apostrophe*.

2820 3.349 Slash

2821 The character ' / ', also known as *solidus*.

2822 3.350 Socket

2823 A file of a particular type that is used as a communications endpoint for process-to-process
2824 communication as described in the System Interfaces volume of IEEE Std. 1003.1-200x.

2825 3.351 Socket Address

2826 An address associated with a socket or remote endpoint, including an address family identifier
2827 and addressing information specific to that address family. The address may include multiple
2828 parts, such as a network address associated with a host system and an identifier for a specific
2829 endpoint.

2830 3.352 Soft Limit

2831 A resource limitation established for each process that the process may set to any value less than
2832 or equal to the hard limit.

2833 3.353 Source Code

2834 When dealing with the Shell Command Language, input to the command language interpreter.
2835 The term shell script is synonymous with this meaning.

2836 When dealing with an ISO/IEC-conforming programming language, source code is input to a
2837 compiler conforming to that ISO/IEC standard.

2838 Source code also refers to the input statements prepared for the following standard utilities:
2839 *awk, bc, ed, lex, localedef, make, sed, and yacc*.

2840 Source code can also refer to a collection of sources meeting any or all of these meanings.

2841 **Note:** The *awk, bc, ed, lex, localedef, make, sed, and yacc* utilities are defined in detail in the
2842 Shell and Utilities volume of IEEE Std. 1003.1-200x.

2843 3.354 Space Character (<space>)

2844 The character defined in the portable character set as <space>. The <space> character is a
2845 member of the **space** character class of the current locale, but represents the single character, and
2846 not all of the possible members of the class; see also Section 3.433 (on page 116).

2847 3.355 Spawn

2848 A process creation primitive useful for systems that have difficulty with *fork()* and as an efficient
2849 replacement for *fork()/exec*.

2850 3.356 Special Built-In

2851 See *Built-In Utility* in Section 3.85 (on page 55).

2852 3.357 Special Parameter

2853 In the shell command language, a parameter named by a single character from the following list:

2854 * @ # ? ! - \$ 0

2855 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
2856 Section 2.5.2, Special Parameters.

2857 3.358 Spin Lock

2858 A synchronization object used to allow multiple threads to serialize their access to shared data.

2859 3.359 Sporadic Server

2860 A scheduling policy for threads and processes that reserves a certain amount of execution
2861 capacity for processing aperiodic events at a given priority level.

2862 3.360 Standard Error

2863 An output stream usually intended to be used for diagnostic messages.

2864 3.361 Standard Input

2865 An input stream usually intended to be used for primary data input.

2866 3.362 Standard Output

2867 An output stream usually intended to be used for primary data output.

2868 3.363 Standard Utilities

2869 The utilities described in the Shell and Utilities volume of IEEE Std. 1003.1-200x.

2870 3.364 Stream

2871 Appearing in lowercase, a stream is a file access object that allows access to an ordered sequence
2872 of characters, as described by the ISO C standard. Such objects can be created by the *fdopen()*,
2873 *fopen()*, or *popen()* functions, and are associated with a file descriptor. A stream provides the
2874 additional services of user-selectable buffering and formatted input and output; see also Section
2875 3.365.

2876 **Note:** For further information, see the System Interfaces volume of IEEE Std. 1003.1-200x,
2877 Section 2.5, Standard I/O Streams.

2878 The *fdopen()*, *fopen()*, or *popen()* functions are defined in detail in the System
2879 Interfaces volume of IEEE Std. 1003.1-200x.

2880 3.365 STREAM

2881 Appearing in uppercase, STREAM refers to a full duplex connection between a process and an
2882 open device or pseudo-device. It optionally includes one or more intermediate processing
2883 modules that are interposed between the process end of the STREAM and the device driver (or
2884 pseudo-device driver) end of the STREAM; see also Section 3.364.

2885 **Note:** For further information, see the System Interfaces volume of IEEE Std. 1003.1-200x,
2886 Section 2.6, STREAMS.

2887 3.366 STREAM End

2888 The STREAM end is the driver end of the STREAM and is also known as the downstream end of
2889 the STREAM.

2890 3.367 STREAM Head

2891 The STREAM head is the beginning of the STREAM and is at the boundary between the system
2892 and the application process. This is also known as the upstream end of the STREAM.

2893 3.368 STREAMS Multiplexor

2894 A driver with multiple STREAMS connected to it. Multiplexing with STREAMS connected above
2895 is referred to as N-to-1, or *upper multiplexing*. Multiplexing with STREAMS connected below is
2896 referred to as 1-to-N or *lower multiplexing*.

2897 3.369 String

2898 A contiguous sequence of bytes terminated by and including the first null byte.

2899 3.370 Subshell

2900 A shell execution environment, distinguished from the main or current shell execution
2901 environment.

2902 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
2903 Section 2.13, Shell Execution Environment.

2904 3.371 Successfully Transferred

2905 For a write operation to a regular file, when the system ensures that all data written is readable
2906 on any subsequent open of the file (even one that follows a system or power failure) in the
2907 absence of a failure of the physical storage medium.

2908 For a read operation, when an image of the data on the physical storage medium is available to
2909 the requesting process.

2910 3.372 Supplementary Group ID

2911 An attribute of a process used in determining file access permissions. A process has up to
2912 {NGROUPS_MAX} supplementary group IDs in addition to the effective group ID. The
2913 supplementary group IDs of a process are set to the supplementary group IDs of the parent
2914 process when the process is created.

2915 3.373 Suspended Job

2916 A job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the
2917 process group to stop. A suspended job is a background job, but a background job is not
2918 necessarily a suspended job.

2919 3.374 Symbolic Link

2920 A type of file with the property that when the file is encountered during path name resolution, a
2921 string stored by the file is used to modify the path name resolution. The stored string has a
2922 length of {SYMLINK_MAX} bytes or fewer.

2923 **Note:** Path Name Resolution is defined in detail in Section 4.9 (on page 123).

2924 3.375 Synchronized Input and Output

2925 A determinism and robustness improvement mechanism to enhance the data input and output
2926 mechanisms, so that an application can ensure that the data being manipulated is physically
2927 present on secondary mass storage devices.

2928 3.376 Synchronized I/O Completion

2929 The state of an I/O operation that has either been successfully transferred or diagnosed as
2930 unsuccessful.

2931 3.377 Synchronized I/O Data Integrity Completion

2932 For read, when the operation has been completed or diagnosed if unsuccessful. The read is
2933 complete only when an image of the data has been successfully transferred to the requesting
2934 process. If there were any pending write requests affecting the data to be read at the time that
2935 the synchronized read operation was requested, these write requests are successfully transferred
2936 prior to reading the data.

2937 For write, when the operation has been completed or diagnosed if unsuccessful. The write is
2938 complete only when the data specified in the write request is successfully transferred and all file
2939 system information required to retrieve the data is successfully transferred.

2940 File attributes that are not necessary for data retrieval (access time, modification time, status
2941 change time) need not be successfully transferred prior to returning to the calling process.

2942 3.378 Synchronized I/O File Integrity Completion

2943 Identical to a synchronized I/O data integrity completion with the addition that all file attributes
2944 relative to the I/O operation (including access time, modification time, status change time) are
2945 successfully transferred prior to returning to the calling process.

2946 3.379 Synchronized I/O Operation

2947 An I/O operation performed on a file that provides the application assurance of the integrity of
2948 its data and files.

2949 3.380 Synchronous I/O Operation

2950 An I/O operation that causes the thread requesting the I/O to be blocked from further use of the
2951 processor until that I/O operation completes.

2952 **Note:** A synchronous I/O operation does not imply synchronized I/O data integrity
2953 completion or synchronized I/O file integrity completion.

2954 3.381 Synchronously-Generated Signal

2955 A signal that is attributable to a specific thread.

2956 For example, a thread executing an illegal instruction or touching invalid memory causes a
2957 synchronously-generated signal. Being synchronous is a property of how the signal was
2958 generated and not a property of the signal number.

2959 3.382 System

2960 An implementation of IEEE Std. 1003.1-200x.

2961 3.383 System Crash

2962 An interval initiated by an unspecified circumstance that causes all processes (possibly other
2963 than special system processes) to be terminated in an undefined manner, after which any
2964 changes to the state and contents of files created or written to by an application prior to the
2965 interval are undefined, except as required elsewhere in IEEE Std. 1003.1-200x.

2966 3.384 System Console

2967 An optional file that receives messages sent by *fmtmsg()* when the MM_CONSOLE flag is set. |

2968 **Note:** The *fmtmsg()* function is defined in detail in the System Interfaces volume of |
2969 IEEE Std. 1003.1-200x.

2970 3.385 System Databases

2971 An implementation provides two system databases.

2972 The *group database* contains the following information for each group:

- 2973 1. Group name
- 2974 2. Numerical group ID
- 2975 3. List of all users allowed in the group

2976 The *user database* contains the following information for each user:

- 2977 1. User name
- 2978 2. Numerical user ID
- 2979 3. Numerical group ID
- 2980 4. Initial working directory
- 2981 5. Initial user program

2982 If the initial user program field is null, the system default is used. If the initial working directory
2983 field is null, the interpretation of that field is implementation-defined. These databases may |
2984 contain other fields that are unspecified by IEEE Std. 1003.1-200x.

2985 3.386 System Documentation

2986 All documentation provided with an implementation except for the conformance document or |
2987 Conformance Statement Questionnaire (CSQ). Electronically distributed documents for an |
2988 implementation are considered part of the system documentation.

2989 3.387 System Process

2990 An implementation-defined object, other than a process executing an application, that has a |
2991 process ID.

2992 3.388 System Reboot

2993 An implementation-defined sequence of events that may result in the loss of transitory data; that
2994 is, data that is not saved in permanent storage. For example, message queues, shared memory,
2995 semaphores, and processes.

2996 3.389 System Trace Event

2997 A trace event that is generated by the implementation, in response either to a system-initiated
2998 action or to an application-requested action, except for a call to *posix_trace_event()*. When
2999 supported by the implementation, a system-initiated action generates a process-independent
3000 system trace event and an application-requested action generates a process-dependent system
3001 trace event. For a system trace event not defined by IEEE Std. 1003.1-200x, the associated trace
3002 event type identifier is derived from the implementation-defined name for this trace event, and
3003 the associated data is of implementation-defined content and length.

3004 3.390 System-Wide

3005 Pertaining to events occurring in all processes existing in an implementation at a given point in
3006 time.

3007 3.391 Tab Character (<tab>)

3008 A character that in the output stream indicates that printing or displaying should start at the
3009 next horizontal tabulation position on the current line. The <tab> character is the character
3010 designated by '\t' in the C language. If the current position is at or past the last defined
3011 horizontal tabulation position, the behavior is unspecified. It is unspecified whether this
3012 character is the exact sequence transmitted to an output device by the system to accomplish the
3013 tabulation.

3014 3.392 Terminal (or Terminal Device)

3015 A character special file that obeys the specifications of the general terminal interface.

3016 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 213).

3017 3.393 Text Column

3018 A roughly rectangular block of characters capable of being laid out side-by-side next to other
3019 text columns on an output page or terminal screen. The widths of text columns are measured in
3020 column positions.

3021 3.394 Text File

3022 A file that contains characters organized into one or more lines. The lines do not contain NUL
3023 characters and none can exceed {LINE_MAX} bytes in length, including the <newline> character.
3024 Although IEEE Std. 1003.1-200x does not distinguish between text files and binary files (see the
3025 ISO C standard), many utilities only produce predictable or meaningful output when operating
3026 on text files. The standard utilities that have such restrictions always specify *text files* in their
3027 STDIN or INPUT FILES sections.

3028 3.395 Thread

3029 A single flow of control within a process. Each thread has its own thread ID, scheduling priority
3030 and policy, *errno* value, thread-specific key/value bindings, and the required system resources to
3031 support a flow of control. Anything whose address may be determined by a thread, including
3032 but not limited to static variables, storage obtained via *malloc()*, directly addressable storage
3033 obtained through implementation-defined functions, and automatic variables, are accessible to
3034 all threads in the same process.

3035 **Note:** The *malloc()* function is defined in detail in the System Interfaces volume of
3036 IEEE Std. 1003.1-200x.

3037 3.396 Thread ID

3038 Each thread in a process is uniquely identified during its lifetime by a value of type **pthread_t**
3039 called a thread ID.

3040 3.397 Thread List

3041 An ordered set of runnable threads that all have the same ordinal value for their priority.

3042 The ordering of threads on the list is determined by a scheduling policy or policies. The set of
3043 thread lists includes all runnable threads in the system.

3044 3.398 Thread-Safe

3045 A function that may be safely invoked concurrently by multiple threads. Each function defined
3046 in the System Interfaces volume of IEEE Std. 1003.1-200x is thread-safe unless explicitly stated
3047 otherwise. Examples are any “pure” function, a function which holds a mutex locked while it is
3048 accessing static storage, or objects shared among threads.

3049 3.399 Thread-Specific Data Key

3050 A process global handle of type `pthread_key_t` which is used for naming thread-specific data.

3051 Although the same key value may be used by different threads, the values bound to the key by
3052 `pthread_setspecific()` and accessed by `pthread_getspecific()` are maintained on a per-thread basis
3053 and persist for the life of the calling thread.

3054 **Note:** The `pthread_getspecific()` and `pthread_setspecific()` functions are defined in detail in the
3055 System Interfaces volume of IEEE Std. 1003.1-200x.

3056 3.400 Tilde

3057 The character ‘~’.

3058 3.401 Timeouts

3059 A method of limiting the length of time an interface will block; see also Section 3.77 (on page 54).

3060 3.402 Timer

3061 A mechanism that can notify a thread when the time as measured by a particular clock has
3062 reached or passed a specified value, or when a specified amount of time has passed.

3063 3.403 Timer Overrun

3064 A condition that occurs each time a timer, for which there is already an expiration signal queued
3065 to the process, expires.

3066 3.404 Token

3067 In the shell command language, a sequence of characters that the shell considers as a single unit
3068 when reading input. A token is either an operator or a word.

3069 **Note:** The rules for reading input are defined in detail in the Shell and Utilities volume of
3070 IEEE Std. 1003.1-200x, Section 2.3, Token Recognition.

3071 3.405 Trace Analyzer Process

3072 A process that extracts trace events from a trace stream to retrieve information about the
3073 behavior of an application. A trace controller process may also be a trace analyzer process. Trace
3074 analysis can be done concurrently with the traced process or can be done off-line, in the same or
3075 in a different platform.

3076 3.406 Trace Controller Process

3077 A process that creates a trace stream for tracing a process. Only the trace controller process has
3078 control of the trace stream it has created. The control of the operation of a trace stream is done
3079 using its corresponding trace stream identifier. The trace controller process is able to:

- 3080 • Initialize the attributes of a trace stream
- 3081 • Create the trace stream
- 3082 • Start and stop tracing
- 3083 • Know the mapping of the traced process
- 3084 • If the Trace Event Filter option is supported, filter the type of trace events to be recorded
- 3085 • Shut the trace stream down

3086 A traced process may also be a trace controller process. Only the trace controller process can
3087 control its trace stream(s). A trace stream created by a trace controller process is shut down if its
3088 controller process terminates or executes another file.

3089 3.407 Trace Event

3090 A data object that represents an action executed by the system, and that is recorded in a trace
3091 stream. Each trace event is of a particular trace event type, and is associated with a trace event
3092 type identifier. The execution of a trace point generates a trace event if a trace stream has been
3093 created and started for the process that executed the trace point and if the corresponding trace
3094 event type identifier is not ignored by filtering.

3095 A generated trace event should be recorded in a trace stream and optionally also in a trace log if
3096 a trace log was associated with the trace stream.

3097 The only case in which a generated trace event is not recorded in the trace stream is when no
3098 resources are available for it in the trace stream. In this case, the trace event is lost.

3099 The only two cases in which a generated trace event is not recorded in the trace log are when no
3100 resources are available for it in the trace log or when a flush operation does not succeed.

3101 A trace event recorded in an active trace stream may be retrieved by an application having the
3102 appropriate privileges.

3103 A trace event recorded in a trace log may be retrieved by an application having the appropriate
3104 privileges after opening the trace log as a pre-recorded trace stream, with the function
3105 *posix_trace_open()*.

3106 When a trace event is reported it is possible to retrieve the following:

- 3107 • A trace event type identifier
- 3108 • A timestamp
- 3109 • The process ID of the traced process, if the trace event is process-dependent
- 3110 • Any optional trace event data including its length
- 3111 • If the Threads option is supported, the thread ID, if the trace event is process-dependent
- 3112 • The program address at which the trace point was invoked

3113 **3.408 Trace Event Type**

3114 A data object type that defines a class of trace event. A trace event type is identified on the one
3115 hand by a trace event type name, also referenced as a trace event name, and on the other hand by
3116 a trace event type identifier. A trace event name is a human-readable string. A trace event type
3117 identifier is an opaque identifier used by the trace system. There is a one-to-one relationship
3118 between trace event type identifiers and trace event names for a given trace stream and also for a
3119 given traced process. The trace event type identifier is generated automatically from a trace
3120 event name by the trace system either when a trace controller process invokes
3121 *posix_trace_trid_eventid_open()* or when an instrumented application process invokes
3122 *posix_trace_eventid_open()*. Trace event type identifiers are used to filter trace event types, to
3123 allow interpretation of user data, and to identify the kind of trace point that generated a trace
3124 event.

3125 **3.409 Trace Event Type Mapping**

3126 A one-to-one mapping between trace event types and trace event names. One such mapping is
3127 associated with each trace stream. An active trace stream is associated to a traced process, and
3128 also to its children if the Trace Inherit option is supported and also the inheritance policy is set to
3129 `_POSIX_TRACE_INHERIT`. Therefore each traced process has a mapping of the trace event
3130 names to trace event type identifiers that have been defined for that process.

3131 3.410 Trace Filter

3132 A filter that allows the trace controller process to specify those trace event types that are to be
3133 ignored; that is, not generated. The operation of the filter is to filter out (ignore) selected trace
3134 events. By default, no trace events are filtered.

3135 3.411 Trace Generation Version

3136 A data object that is an implementation-defined character string, generated by the trace system
3137 and describing the origin and version of the trace system.

3138 3.412 Trace Log

3139 The flushed image of a trace stream, if the trace stream is created with a trace log. The trace log
3140 is recorded when the *posix_trace_shutdown()* operation is invoked or during tracing, depending
3141 on the tracing strategy which is defined by a log policy. After the trace stream has been shut
3142 down, the trace information can be retrieved from the associated trace log using the same
3143 interface used to retrieve information from an active trace stream.

3144 3.413 Trace Point

3145 An action that may cause a trace event to be generated. This may be an implementation-defined
3146 action such as a context switch, or an application-programmed action such as a call to a specific
3147 operating system service (for example, *fork()*) or a call to *posix_trace_event()*.

3148 3.414 Trace Stream

3149 An opaque object that contains trace events plus internal data needed to interpret those trace
3150 events. The implementation and format of a trace stream are unspecified. A trace stream need
3151 not be and generally is not persistent. A trace stream may be either active or pre-recorded:

- 3152 • An active trace stream is a trace stream that has been created and has not yet been shut
3153 down. It can be of one of the two following classes:
 - 3154 1. An active trace stream without a trace log that was created with the *posix_trace_create()*
3155 function
 - 3156 2. If the Trace Log option is supported, an active trace stream with a trace log that was
3157 created with the *posix_trace_create_withlog()* function
- 3158 • A pre-recorded trace stream is a trace stream that was opened from a trace log object using
3159 the *posix_trace_open()* function.

3160 An active trace stream can loop. This behavior means that when the resources allocated by the
3161 trace system for the trace stream are exhausted, the trace system reuses the resources associated
3162 with the oldest recorded trace events to record new trace events.

3163 If the Trace Log option is supported, an active trace stream with a trace log can be flushed. This
3164 operation causes the trace system to write trace events from the trace stream to the associated
3165 trace log, following the defined policies or using an explicit function call. After this operation,
3166 the trace system may reuse the resources associated with the flushed trace events.

3167 An active trace stream with or without a trace log can be cleared. This operation causes all the
3168 resources associated with this trace stream to be reinitialized. The trace stream behaves as if it
3169 was returning from its creation, except that the mapping of trace event type identifiers to trace
3170 event names is not cleared. If a trace log was associated with this trace stream, the trace log is
3171 also reinitialized.

3172 **3.415 Trace Stream Identifier**

3173 A handle to manage tracing operations in a trace stream.

3174 **3.416 Trace System**

3175 A system that allows both system and user trace events to be generated into a trace stream.
3176 These trace events can be retrieved later.

3177 **3.417 Traced Process**

3178 A process for which at least one trace stream has been created. A traced process is also called a
3179 target process. If the Trace Inherit option is supported and the trace stream's inheritance
3180 attribute is `_POSIX_TRACE_INHERIT`, the initial targeted traced process is traced together with
3181 all of its future children. The *posix_pid* member of each trace event in a trace stream is the
3182 process ID of the traced process.

3183 **3.418 Tracing Status of a Trace Stream**

3184 A status that describes the state of an active trace stream. The tracing status of a trace stream can
3185 be retrieved from the trace stream attributes. An active trace stream can be in one of two states:
3186 running or suspended.

3187 **3.419 Typed Memory Name Space**

3188 A system-wide name space that contains the names of the typed memory objects present in the
3189 system. It is configurable for a given implementation.

3190 3.420 Typed Memory Object

3191 A combination of a typed memory pool and a typed memory port. The entire contents of the
3192 pool are accessible from the port. The typed memory object is identified through a name that
3193 belongs to the typed memory name space.

3194 3.421 Typed Memory Pool

3195 An extent of memory with the same operational characteristics. Typed memory pools may be
3196 contained within each other.

3197 3.422 Typed Memory Port

3198 A hardware access path to one or more typed memory pools.

3199 3.423 Unbind

3200 Remove the association between a network address and an endpoint.

3201 3.424 Unit Data

3202 See *Datagram* in Section 3.126 (on page 62).

3203 3.425 Upshifting

3204 The conversion of a lowercase character that has a single-character uppercase representation
3205 into this uppercase representation.

3206 3.426 User Database

3207 A system database of implementation-defined format that contains at least the following
3208 information for each user ID:

- 3209 • User name
- 3210 • Numerical user ID
- 3211 • Initial numerical group ID
- 3212 • Initial working directory
- 3213 • Initial user program

3214 The initial numerical group ID is used by the *newgrp* utility. Any other circumstances under
3215 which the initial values are operative are implementation-defined.

3216 If the initial user program field is null, an implementation-defined program is used.

3217 If the initial working directory field is null, the interpretation of that field is implementation-
3218 defined.

3219 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of
3220 IEEE Std. 1003.1-200x.

3221 **3.427 User ID**

3222 A non-negative integer that is used to identify a system user. When the identity of a user is
3223 associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or a
3224 saved set-user-ID.

3225 **3.428 User Name**

3226 A string that is used to identify a user; see also Section 3.426 (on page 114). To be portable across
3227 systems conforming to IEEE Std. 1003.1-200x, the value is composed of characters from the
3228 portable file name character set. The hyphen should not be used as the first character of a
3229 portable user name.

3230 **3.429 User Trace Event**

3231 A trace event that is generated explicitly by the application as a result of a call to
3232 *posix_trace_event()*.

3233 **3.430 Utility**

3234 A program, excluding special built-in utilities provided as part of the Shell Command Language,
3235 that can be called by name from a shell to perform a specific task, or related set of tasks.

3236 **Note:** For further information on special built-in utilities, see the Shell and Utilities volume
3237 of IEEE Std. 1003.1-200x, Section 2.15, Special Built-In Utilities.

3238 **3.431 Variable**

3239 In the shell command language, a named parameter.

3240 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
3241 Section 2.5, Parameters and Variables.

3242 3.432 Vertical-Tab Character (<vertical-tab>)

3243 A character that in the output stream indicates that printing should start at the next vertical
3244 tabulation position. The <vertical-tab> character is the character designated by '\v' in the C
3245 language. If the current position is at or past the last defined vertical tabulation position, the
3246 behavior is unspecified. It is unspecified whether this character is the exact sequence transmitted
3247 to an output device by the system to accomplish the tabulation.

3248 3.433 White Space

3249 A sequence of one or more characters that belong to the **space** character class as defined via the
3250 *LC_CTYPE* category in the current locale.

3251 In the POSIX locale, white space consists of one or more <blank> characters (<space> and <tab>
3252 characters), <newline> characters, <carriage-return> characters, <form-feed> characters, and
3253 <vertical-tab> characters.

3254 3.434 Wide-Character Code (C Language)

3255 An integer value corresponding to a single graphic symbol or control code.

3256 **Note:** C Language Wide-Character Codes are defined in detail in Section 6.3 (on page 137).

3257 3.435 Wide-Character Input/Output Functions

3258 The functions that perform wide-oriented input from streams or wide-oriented output to
3259 streams: *fgetwc()*, *fputwc()*, *fputws()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *getws()*, *putwc()*,
3260 *putwchar()*, *ungetwc()*, *vfwprintf()*, *vwprintf()*, *wprintf()*, and *wscanf()*.

3261 **Note:** These functions are defined in detail in the System Interfaces volume of
3262 IEEE Std. 1003.1-200x.

3263 3.436 Wide-Character String

3264 A contiguous sequence of wide-character codes terminated by and including the first null wide-
3265 character code.

3266 3.437 Word

3267 In the shell command language, a token other than an operator. In some cases a word is also a
3268 portion of a word token: in the various forms of parameter expansion, such as $\${name-word}$, and
3269 variable assignment, such as $name=word$, the word is the portion of the token depicted by *word*.
3270 The concept of a word is no longer applicable following word expansions—only fields remain.

3271 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
3272 Section 2.6.2, Parameter Expansion and the Shell and Utilities volume of
3273 IEEE Std. 1003.1-200x, Section 2.6, Word Expansions.

3274 3.438 Working Directory (or Current Working Directory)

3275 A directory, associated with a process, that is used in path name resolution for path names that
3276 do not begin with a slash.

3277 3.439 Worldwide Portability Interface

3278 Functions for handling characters in a codeset-independent manner.

3279 3.440 Write

3280 To output characters to a file, such as standard output or standard error. Unless otherwise
3281 stated, standard output is the default output destination for all uses of the term write; see the
3282 distinction between display and write in Section 3.135 (on page 64).

3283 3.441 XSI

3284 The X/Open System Interface is the core application programming interface for C and *sh*
3285 programming for systems conforming to the Single UNIX Specification. This is a superset of the
3286 mandatory requirements for conformance to IEEE Std. 1003.1-200x.

3287 3.442 XSI-Conformant

3288 A system which allows an application to be built using a set of services that are consistent across
3289 all systems that conform to IEEE Std. 1003.1-200x and that support the XSI extension.

3290 **Note:** See also Chapter 2 (on page 19).

3291 **3.443 Zombie Process**

3292 A process that has terminated and that is deleted when its exit status has been reported to
 3293 another process which is waiting for that process to terminate.

3294 **3.444 ±0**

3295 The algebraic sign provides additional information about any variable that has the value zero
 3296 when the representation allows the sign to be determined.

3297 **CHANGE HISTORY**3298 **Issue 4**

3299 Numerous changes and additions are made for alignment with the ISO C standard and
 3300 the ISO POSIX-1 standard.

3301 **Issue 4, Version 2**

3302 The following terms are added to support the adoption of additional traditional UNIX
 3303 interfaces: *alternate signal stack*, *break value*, *data segment*, *driver*, *hard limit*, *host byte*
 3304 *order*, *named STREAM*, *network byte order*, *network host database*, *network net database*,
 3305 *network protocol database*, *network service database*, *pad*, *parent window*, *priority band*,
 3306 *process virtual time*, *pseudo-terminal*, *real time*, *signal stack*, *socket*, *soft limit*, *STREAM*
 3307 (second definition), *STREAM end*, *STREAM head*, *STREAMS multiplexor*, *symbolic link*,
 3308 *system console*, and *timer*.

3309 **Issue 5**

3310 Numerous terms are added to support adoption of the POSIX Threads Extension and
 3311 the POSIX Realtime Extension.

3312 **Issue 6**

3313 Additional terms are added to cover material from the ISO POSIX-1: 1996 standard and
 3314 the ISO POSIX-2: 1993 standard not previously included.

3315 Various XSI-related terms are added.

3316 The following definitions are added for alignment with IEEE Std. 1003.1d-1999: *Spawn*,
 3317 *Timeouts*, *Execution Time Monitoring*, *Sporadic Server*, *Advisory Information*, *CPU*
 3318 *Time*, *CPU-Time Clock*, *CPU-Time Timer*, and *Execution Time*.

3319 The definition of *Memory Object* is modified to include typed memory objects for
 3320 alignment with IEEE Std. 1003.1j-2000.

3321 Definitions of *Barrier*, *Clock Jump*, *Monotonic Clock*, *Read-Write Lock*, *Spin Lock*,
 3322 *Typed Memory Name Space*, *Typed Memory Object*, *Typed Memory Pool*, and *Typed*
 3323 *Memory Port* are added for alignment with IEEE Std. 1003.1j-2000.

3324 The *Read-Write Lock* definition is moved under the *RWL* option for alignment with
 3325 IEEE Std. 1003.1j-2000.

3326 **Notes to Reviewers**

3327 *This section with side shading will not appear in the final copy. - Ed.*

3328 To be further expanded.

3330

3331 4.1 Concurrent Execution

3332 Functions that suspend the execution of the calling thread shall not cause the execution of other
3333 threads to be indefinitely suspended.

3334 4.2 Extended Security Controls

3335 An implementation may provide implementation-defined extended security controls (see
3336 Section 3.161 (on page 68)). These permit an implementation to provide security mechanisms to
3337 implement different security policies than those described in IEEE Std. 1003.1-200x. These
3338 mechanisms shall not alter or override the defined semantics of any of the interfaces in
3339 IEEE Std. 1003.1-200x.

3340 4.3 File Access Permissions

3341 The standard file access control mechanism uses the file permission bits, as described below.

3342 Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An
3343 additional access control mechanism shall only further restrict the access permissions defined by
3344 the file permission bits. An alternate file access control mechanism shall:

- 3345 • Specify file permission bits for the file owner class, file group class, and file other class of that
3346 file, corresponding to the access permissions.
- 3347 • Be enabled only by explicit user action, on a per-file basis by the file owner or a user with the
3348 appropriate privilege.
- 3349 • Be disabled for a file after the file permission bits are changed for that file with *chmod()*. The
3350 disabling of the alternate mechanism need not disable any additional mechanisms supported
3351 by an implementation.

3352 Whenever a process requests file access permission for read, write, or execute/search, if no
3353 additional mechanism denies access, access is determined as follows:

- 3354 • If a process has the appropriate privilege:
 - 3355 — If read, write, or directory search permission is requested, access is granted.
 - 3356 — If execute permission is requested, access is granted if execute permission is granted to at
3357 least one user by the file permission bits or by an alternate access control mechanism;
3358 otherwise, access is denied.
- 3359 • Otherwise:
 - 3360 — The file permission bits of a file contain read, write, and execute/search permissions for
3361 the file owner class, file group class, and file other class.
 - 3362 — Access is granted if an alternate access control mechanism is not enabled and the
3363 requested access permission bit is set for the class (file owner class, file group class, or file
3364 other class) to which the process belongs, or if an alternate access control mechanism is

3365 enabled and it allows the requested access; otherwise, access is denied.

3366 **4.4 File Hierarchy**

3367 Files in the system are organized in a hierarchical structure in which all of the non-terminal
3368 nodes are directories and all of the terminal nodes are any other type of file. Because multiple
3369 directory entries may refer to the same file, the hierarchy is properly described as a *directed*
3370 *graph*.

3371 **4.5 File Names**

3372 For a file name to be portable across implementations conforming to IEEE Std. 1003.1-200x, it
3373 shall consist only of the Portable File Name Character Set as defined in Section 3.278 (on page
3374 88).

3375 The hyphen character shall not be used as the first character of a portable file name. Uppercase
3376 and lowercase letters retain their unique identities between conforming implementations. In the
3377 case of a portable path name, the slash character may also be used.

3378 **4.6 File Times Update**

3379 Each file has three distinct associated time values: *st_atime*, *st_mtime*, and *st_ctime*. The *st_atime*
3380 field is associated with the times that the file data is accessed; *st_mtime* is associated with the
3381 times that the file data is modified; and *st_ctime* is associated with the times that the file status is
3382 changed. These values are returned in the file characteristics structure, as described in
3383 `<sys/stat.h>`.

3384 Each function or utility in IEEE Std. 1003.1-200x that reads or writes data or changes file status
3385 indicates which of the appropriate time-related fields shall be “marked for update”. If an
3386 implementation of such a function or utility marks for update a time-related field not specified
3387 by IEEE Std. 1003.1-200x, this shall be documented, except that any changes caused by path
3388 name resolution need not be documented. For the other functions or utilities in
3389 IEEE Std. 1003.1-200x (those that are not explicitly required to read or write file data or change
3390 file status, but that in some implementations happen to do so), the effect is unspecified.

3391 An implementation may update fields that are marked for update immediately, or it may update
3392 such fields periodically. At an update point in time, any marked fields are set to the current time
3393 and the update marks are cleared. All fields that are marked for update shall be updated when
3394 the file ceases to be open by any process, or when a *stat()*, *lstat()*, or *lstat()* is performed on the
3395 file. Other times at which updates are done are unspecified. Marks for update, and updates
3396 themselves, are not done for files on read-only file systems; see Section 3.306 (on page 93).

3397 4.7 Measurement of Execution Time

3398 The mechanism used to measure execution time shall be implementation-defined. The
 3399 implementation shall also define to whom the CPU time that is consumed by interrupt handlers
 3400 and system services on behalf of the operating system will be charged. See Section 3.120 (on
 3401 page 62).

3402 4.8 Memory Synchronization

3403 Applications shall ensure that access to any memory location by more than one thread of control
 3404 (threads or processes) is restricted such that no thread of control can read or modify a memory
 3405 location while another thread of control may be modifying it. Such access is restricted using
 3406 functions that synchronize thread execution and also synchronize memory with respect to other
 3407 threads. The following functions synchronize memory with respect to other threads:

3408	<code>fork()</code>	<code>pthread_mutex_trylock()</code>		
3409	<code>pthread_cond_broadcast()</code>	<code>pthread_mutex_unlock()</code>		
3410	<code>pthread_cond_signal()</code>	<code>sem_post()</code>		
3411	<code>pthread_cond_timedwait()</code>	<code>sem_trywait()</code>		
3412	<code>pthread_cond_wait()</code>	<code>sem_wait()</code>		
3413	<code>pthread_create()</code>	<code>wait()</code>		
3414	<code>pthread_join()</code>	<code>waitpid()</code>		
3415	<code>pthread_mutex_lock()</code>			

3416 Notes to Reviewers

3417 *This section with side shading will not appear in the final copy. - Ed.*

3418 We need to check whether there should be any additional functions listed.

3419 Unless explicitly stated otherwise, if one of the above functions returns an error, it is unspecified
 3420 whether the invocation causes memory to be synchronized.

3421 Applications may allow more than one thread of control to read a memory location
 3422 simultaneously.

3423 4.9 Path Name Resolution

3424 Path name resolution is performed for a process to resolve a path name to a particular file in a
 3425 file hierarchy. There may be multiple path names that resolve to the same file.

3426 Each file name in the path name is located in the directory specified by its predecessor (for
 3427 example, in the path name fragment **a/b**, file **b** is located in directory **a**). Path name resolution
 3428 fails if this cannot be accomplished. If the path name begins with a slash, the predecessor of the
 3429 first file name in the path name is taken to be the root directory of the process (such path names
 3430 are referred to as *absolute path names*). If the path name does not begin with a slash, the
 3431 predecessor of the first file name of the path name is taken to be the current working directory of
 3432 the process (such path names are referred to as *relative path names*).

3433 The interpretation of a path name component is dependent on the value of {NAME_MAX} and
 3434 _POSIX_NO_TRUNC associated with the path prefix of that component. If any path name
 3435 component is longer than {NAME_MAX}, the implementation shall consider this an error.

3436 A path name that contains at least one non-slash character and that ends with one or more
 3437 trailing slashes shall be resolved as if a single dot character ('.') were appended to the path

3438 name.

3439 If a symbolic link is encountered during path name resolution, the behavior shall depend on
3440 whether the path name component is at the end of the path name and on the function being
3441 performed. If all of the following are true, then path name resolution is complete:

- 3442 1. This is the last path name component of the path name.
- 3443 2. The path name has no trailing slash.
- 3444 3. The function is required to act on the symbolic link itself, or certain arguments direct that
3445 the function act on the symbolic link itself.

3446 In all other cases, the system shall prefix the remaining path name, if any, with the contents of
3447 the symbolic link. If the combined length exceeds {PATH_MAX}, and the implementation
3448 considers this to be an error, *errno* shall be set to [ENAMETOOLONG] and an error indication
3449 shall be returned. Otherwise, the resolved path name shall be the resolution of the path name
3450 just created. If the resulting path name does not begin with a slash, the predecessor of the first
3451 file name of the path name is taken to be the directory containing the symbolic link.

3452 If the system detects a loop in the path name resolution process, it shall set *errno* to [ELOOP] and
3453 return an error indication. The same may happen if during the resolution process more symbolic
3454 links were followed than the implementation allows. This implementation-defined limit shall
3455 not be smaller than {SYMLOOP_MAX}.

3456 The special file name dot refers to the directory specified by its predecessor. The special file
3457 name dot-dot refers to the parent directory of its predecessor directory. As a special case, in the
3458 root directory, dot-dot may refer to the root directory itself.

3459 A path name consisting of a single slash resolves to the root directory of the process. A null path
3460 name shall not be successfully resolved. A path name that begins with two successive slashes
3461 may be interpreted in an implementation-defined manner, although more than two leading
3462 slashes shall be treated as a single slash.

3463 **4.10 Process ID Reuse**

3464 A process group ID shall not be reused by the system until the process group lifetime ends.

3465 A process ID shall not be reused by the system until the process lifetime ends. In addition, if
3466 there exists a process group whose process group ID is equal to that process ID, the process ID
3467 shall not be reused by the system until the process group lifetime ends. A process that is not a
3468 system process shall not have a process ID of 1.

3469 4.11 Scheduling Policy

3470 A scheduling policy affects process or thread ordering:

- 3471 • When a process or thread is a running thread and it becomes a blocked thread
- 3472 • When a process or thread is a running thread and it becomes a preempted thread
- 3473 • When a process or thread is a blocked thread and it becomes a runnable thread
- 3474 • When a running thread calls a function that can change the priority or scheduling policy of a
- 3475 process or thread
- 3476 • In other scheduling policy-defined circumstances

3477 Conforming implementations are required to define the manner in which each of the scheduling
 3478 policies may modify the priorities or otherwise affect the ordering of processes or threads at
 3479 each of the occurrences listed above. Additionally, conforming implementations define in what
 3480 other circumstances and in what manner each scheduling policy may modify the priorities or
 3481 affect the ordering of processes or threads.

3482 4.12 Seconds Since the Epoch

3483 A value that approximates the number of seconds that have elapsed since the Epoch. A
 3484 Coordinated Universal Time name (specified in terms of seconds (*tm_sec*), minutes (*tm_min*),
 3485 hours (*tm_hour*), days since January 1 of the year (*tm_yday*), and calendar year minus 1900
 3486 (*tm_year*)) is related to a time represented as seconds since the Epoch, according to the
 3487 expression below.

3488 If the year is <1970 or the value is negative, the relationship is undefined. If the year is ≥1970 and
 3489 the value is non-negative, the value is related to a Coordinated Universal Time name according
 3490 to the C-language expression, where *tm_sec*, *tm_min*, *tm_hour*, *tm_yday*, and *tm_year* are all
 3491 integer types:

$$\begin{aligned}
 &tm_sec + tm_min*60 + tm_hour*3600 + tm_yday*86400 + \\
 & (tm_year-70)*31536000 + ((tm_year-69)/4)*86400 - \\
 & ((tm_year-1/100)*86400 + ((tm_year+299)/400)*86400
 \end{aligned}$$

3495 Whether and how the implementation accounts for leap seconds is unspecified.

3496 **Note:** The last term of the current expression adds in a day for every 4th year starting in
 3497 1973. (January 1st of each year following a leap year starting with the first leap year
 3498 after 1970). The first term above subtracts a day every 100 years starting in 2001. The
 3499 last term above adds a day back in every 400 years starting in 2001.

3500 4.13 Semaphore

3501 A minimum synchronization primitive to serve as a basis for more complex synchronization
3502 mechanisms to be defined by the application program.

3503 For the semaphores associated with the Semaphores option, a semaphore is represented as a
3504 shareable resource that has a non-negative integral value. When the value is zero, there is a
3505 (possibly empty) set of threads awaiting the availability of the semaphore.

3506 For the semaphores associated with the X/Open System Interface Extension (XSI), a semaphore
3507 is a positive integer (0 through 32767). The *semget()* function can be called to create a set or array
3508 of semaphores. A semaphore set can contain one or more semaphores up to an implementation-
3509 defined value.

3510 Semaphore Lock Operation

3511 An operation that is applied to a semaphore. If, prior to the operation, the value of the
3512 semaphore is zero, the semaphore lock operation shall cause the calling thread to be blocked and
3513 added to the set of threads awaiting the semaphore; otherwise, the value is decremented.

3514 Semaphore Unlock Operation

3515 An operation that is applied to a semaphore. If, prior to the operation, there are any threads in
3516 the set of threads awaiting the semaphore, then some thread from that set shall be removed from
3517 the set and becomes unblocked; otherwise, the semaphore value is incremented.

3518 4.14 Thread-Safety

3519 Refer to the System Interfaces volume of IEEE Std. 1003.1-200x, Section 2.9, Threads.

3520 4.15 Utility

3521 A utility program shall be either an executable file, such as might be produced by a compiler or
3522 linker system from computer source code, or a file of shell source code, directly interpreted by
3523 the shell. The program may have been produced by the user, provided by the system
3524 implementor, or acquired from an independent distributor.

3525 The system may implement certain utilities as shell functions (see the Shell and Utilities volume
3526 of IEEE Std. 1003.1-200x, Section 2.9.5, Function Definition Command) or built-in utilities, but
3527 only an application that is aware of the command search order described in the Shell and
3528 Utilities volume of IEEE Std. 1003.1-200x, Section 2.9.1.1, Command Search and Execution or of
3529 performance characteristics can discern differences between the behavior of such a function or
3530 built-in utility and that of an executable file.

3531 **4.16 Variable Assignment**

3532 In the shell command language, a word consisting of the following parts:

3533 *varname=value*

3534 When used in a context where assignment is defined to occur and at no other time, the *value*
3535 (representing a word or field) shall be assigned as the value of the variable denoted by *varname*.

3536 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
3537 Section 2.9.1, Simple Commands.

3538 The *varname* and *value* parts meet the requirements for a name and a word, respectively, except
3539 that they are delimited by the embedded unquoted equals-sign, in addition to other delimiters.

3540 **Note:** Additional delimiters are described in the Shell and Utilities volume of
3541 IEEE Std. 1003.1-200x, Section 2.3, Token Recognition.

3542 When a variable assignment is done, the variable shall be created if it did not already exist. If
3543 *value* is not specified, the variable shall be given a null value.

3544 **Note:** An alternative form of variable assignment:

3545 *symbol=value*

3546 (where *symbol* is a valid word delimited by an equals-sign, but not a valid name)
3547 produces unspecified results. The form *symbol=value* is used by the KornShell
3548 *name[expression]=value* syntax.



File Format Notation

3550

3551 The STDIN, STDOUT, STDERR, INPUT FILES, and OUTPUT FILES sections of the utility
 3552 descriptions use a syntax to describe the data organization within the files, when that
 3553 organization is not otherwise obvious. The syntax is similar to that used by the System Interfaces
 3554 volume of IEEE Std. 1003.1-200x *printf()* function, as described in this chapter. When used in
 3555 STDIN or INPUT FILES sections of the utility descriptions, this syntax describes the format that
 3556 could have been used to write the text to be read, not a format that could be used by the System
 3557 Interfaces volume of IEEE Std. 1003.1-200x *scanf()* function to read the input file.

3558 The description of an individual record is as follows:

3559 "*format*", [*arg1*, *arg2*, . . . , *argn*]

3560 The *format* is a character string that contains three types of objects defined below:

- 3561 1. *Characters* that are not *escape sequences* or *conversion specifications*, as described below, shall
 3562 be copied to the output.
- 3563 2. *Escape Sequences* represent non-graphic characters.
- 3564 3. *Conversion Specifications* specify the output format of each argument; (see below).

3565 The following characters have the following special meaning in the format string:

3566 ' ' (An empty character position.) Represents one or more <blank> characters.

3567 Δ Represents exactly one <space> character.

3568 Table 5-1 lists escape sequences and associated actions on display devices capable of the action.

3569

Table 5-1 Escape Sequences and Associated Actions

3570

3571

3572

3573

3574

3575

3576

3577

3578

3579

3580

3581

3582

3583

3584

3585

Escape Sequence	Represents Character	Terminal Action
'\\'	backslash	Print the character '\\ '.
'\a'	alert	Attempts to alert the user through audible or visible notification.
'\b'	backspace	Moves the printing position to one column before the current position, unless the current position is the start of a line.
'\f'	form-feed	Moves the printing position to the initial printing position of the next logical page.
'\n'	newline	Moves the printing position to the start of the next line.
'\r'	carriage-return	Moves the printing position to the start of the current line.
'\t'	tab	Moves the printing position to the next tab position on the current line. If there are no more tab positions remaining on the line, the behavior is undefined.
'\v'	vertical-tab	Moves the printing position to the start of the next vertical tab position. If there are no more vertical tab positions left on the page, the behavior is undefined.

3586

3587

Each conversion specification shall be introduced by the percent-sign character ('%'). After the character '%', the following shall appear in sequence:

3588

3589

flags Zero or more *flags*, in any order, that modify the meaning of the conversion specification.

3590

3591

3592

3593

field width An optional string of decimal digits to specify a minimum *field width*. For an output field, if the converted value has fewer bytes than the field width, it shall be padded on the left (or right, if the left-adjustment flag ('-'), described below, has been given) to the field width.

3594

3595

3596

3597

3598

3599

precision Gives the minimum number of digits to appear for the *d*, *o*, *i*, *u*, *x*, or *X* conversions (the field is padded with leading zeros), the number of digits to appear after the radix character for the *e* and *f* conversions, the maximum number of significant digits for the *g* conversion; or the maximum number of bytes to be written from a string in *s* conversion. The precision shall take the form of a period ('.') followed by a decimal digit string; a null digit string is treated as zero.

3600

3601

3602

conversion characters

A conversion character (see below) that indicates the type of conversion to be applied.

3603

The *flag* characters and their meanings are:

3604

3605

3606

3607

3608

- The result of the conversion shall be left-justified within the field.

+ The result of a signed conversion shall always begin with a sign ('+' or '-').

<space> If the first character of a signed conversion is not a sign, a <space> character shall be prefixed to the result. This means that if the <space> character and '+' flags both appear, the <space> character flag shall be ignored.

3609

3610

3611

3612

3613

The value is to be converted to an alternative form. For *c*, *d*, *i*, *u*, and *s* conversions, the behavior is undefined. For *o* conversion, it shall increase the precision to force the first digit of the result to be a zero. For *x* or *X* conversion, a non-zero result has 0x or 0X prefixed to it, respectively. For *e*, *E*, *f*, *g*, and *G* conversions, the result shall always contain a radix character, even if no digits follow the radix character. For *g*

3614		and <i>G</i> conversions, trailing zeros shall not be removed from the result as they
3615		usually are.
3616	0	For <i>d</i> , <i>i</i> , <i>o</i> , <i>u</i> , <i>x</i> , <i>X</i> , <i>e</i> , <i>E</i> , <i>f</i> , <i>g</i> , and <i>G</i> conversions, leading zeros (following any
3617		indication of sign or base) shall be used to pad to the field width; no space padding
3618		is performed. If the '0' and '-' flags both appear, the '0' flag shall be ignored.
3619		For <i>d</i> , <i>i</i> , <i>o</i> , <i>u</i> , <i>x</i> , and <i>X</i> conversions, if a precision is specified, the '0' flag shall be
3620		ignored. For other conversions, the behavior is undefined.
3621		Each conversion character shall result in fetching zero or more arguments. The results are
3622		undefined if there are insufficient arguments for the format. If the format is exhausted while
3623		arguments remain, the excess arguments shall be ignored.
3624		The <i>conversion characters</i> and their meanings are:
3625	<i>d,i,o,u,x,X</i>	The integer argument shall be written as signed decimal (<i>d</i> or <i>i</i>), unsigned octal (<i>o</i>),
3626		unsigned decimal (<i>u</i>), or unsigned hexadecimal notation (<i>x</i> and <i>X</i>). The <i>d</i> and <i>i</i>
3627		specifiers shall convert to signed decimal in the style <code>[-]dddd</code> . The <i>x</i> conversion
3628		shall use the numbers and letters <code>0123456789abcdef</code> and the <i>X</i> conversion shall use
3629		the numbers and letters <code>0123456789ABCDEF</code> . The <i>precision</i> component of the
3630		argument shall specify the minimum number of digits to appear. If the value being
3631		converted can be represented in fewer digits than the specified minimum, it shall
3632		be expanded with leading zeros. The default precision shall be 1. The result of
3633		converting a zero value with a precision of 0 shall be no characters. If both the field
3634		width and precision are omitted, the implementation may precede, follow, or
3635		precede and follow numeric arguments of types <i>d</i> , <i>i</i> , and <i>u</i> with <blank>
3636		characters; arguments of type <i>o</i> (octal) may be preceded with leading zeros.
3637	<i>f</i>	The floating point number argument shall be written in decimal notation in the
3638		style <code>[-]ddd.ddd</code> , where the number of digits after the radix character (shown here
3639		as a decimal point) shall be equal to the <i>precision</i> specification. The <i>LC_NUMERIC</i>
3640		locale category shall determine the radix character to use in this format. If the
3641		<i>precision</i> is omitted from the argument, six digits shall be written after the radix
3642		character; if the <i>precision</i> is explicitly 0, no radix character shall appear.
3643	<i>e,E</i>	The floating point number argument shall be written in the style <code>[-]d.ddd±dd</code> (the
3644		symbol '±' indicates either a plus or minus sign), where there is one digit before
3645		the radix character (shown here as a decimal point) and the number of digits after
3646		it is equal to the precision. The <i>LC_NUMERIC</i> locale category shall determine the
3647		radix character to use in this format. When the precision is missing, six digits shall
3648		be written after the radix character; if the precision is 0, no radix character shall
3649		appear. The <i>E</i> conversion character shall produce a number with <i>E</i> instead of <i>e</i>
3650		introducing the exponent. The exponent shall always contain at least two digits.
3651		However, if the value to be written requires an exponent greater than two digits,
3652		additional exponent digits shall be written as necessary.
3653	<i>g,G</i>	The floating point number argument shall be written in style <i>f</i> or <i>e</i> (or in style <i>E</i> in
3654		the case of a <i>G</i> conversion character), with the precision specifying the number of
3655		significant digits. The style used depends on the value converted: style <i>e</i> (or <i>E</i>)
3656		shall be used only if the exponent resulting from the conversion is less than -4 or
3657		greater than or equal to the precision. Trailing zeros are removed from the result. A
3658		radix character shall appear only if it is followed by a digit.
3659	<i>c</i>	The integer argument shall be converted to an unsigned char and the resulting
3660		byte shall be written.

3661 s The argument shall be taken to be a string and bytes from the string shall be
 3662 written until the end of the string or the number of bytes indicated by the *precision*
 3663 specification of the argument is reached. If the precision is omitted from the
 3664 argument, it shall be taken to be infinite, so all bytes up to the end of the string
 3665 shall be written.

3666 % Write a ' % ' character; no argument is converted.

3667 In no case does a nonexistent or insufficient *field width* cause truncation of a field; if the result of
 3668 a conversion is wider than the field width, the field is simply expanded to contain the conversion
 3669 result. The term *field width* should not be confused with the term *precision* used in the description
 3670 of %s.

3671 **Examples**

3672 To represent the output of a program that prints a date and time in the form Sunday, July 3,
 3673 10:02, where *weekday* and *month* are strings:

3674 " %s, Δ %s Δ %d, Δ %d : % . 2d \n " <weekday> , <month> , <day> , <hour> , <min>

3675 To show ' π ' written to 5 decimal places:

3676 " pi Δ = Δ % . 5f \n " , <value of π >

3677 To show an input file format consisting of five colon-separated fields:

3678 " %s : %s : %s : %s : %s \n " , <arg1> , <arg2> , <arg3> , <arg4> , <arg5>

3679

Character Set

3680

3681 6.1 Portable Character Set

3682 Conforming implementations shall support one or more coded character sets. Each supported
 3683 locale shall include the *portable character set*, which is the set of symbolic names for characters in
 3684 Table 6-1. This is used to describe characters within the text of IEEE Std. 1003.1-200x. The first
 3685 eight entries in Table 6-1 are defined in the ISO/IEC 6429:1992 standard and the rest of the
 3686 characters are defined in the ISO/IEC 10646-1:1993 standard.

3687 **Table 6-1** Portable Character Set

3688

3689

3690

3691

3692

3693

3694

3695

3696

3697

3698

3699

3700

3701

3702

3703

3704

3705

3706

3707

3708

3709

3710

3711

3712

3713

3714

3715

3716

3717

3718

3719

3720

Symbolic Name	Glyph	UCS	Description
<NUL>		<U0000>	NULL (NUL)
<alert>		<U0007>	BELL (BEL)
<backspace>		<U0008>	BACKSPACE (BS)
<tab>		<U0009>	CHARACTER TABULATION (HT)
<carriage-return>		<U000D>	CARRIAGE RETURN (CR)
<newline>		<U000A>	LINE FEED (LF)
<vertical-tab>		<U000B>	LINE TABULATION (VT)
<form-feed>		<U000C>	FORM FEED (FF)
<space>		<U0020>	SPACE
<exclamation-mark>	!	<U0021>	EXCLAMATION MARK
<quotation-mark>	"	<U0022>	QUOTATION MARK
<number-sign>	#	<U0023>	NUMBER SIGN
<dollar-sign>	\$	<U0024>	DOLLAR SIGN
<percent-sign>	%	<U0025>	PERCENT SIGN
<ampersand>	&	<U0026>	AMPERSAND
<apostrophe>	'	<U0027>	APOSTROPHE
<left-parenthesis>	(<U0028>	LEFT PARENTHESIS
<right-parenthesis>)	<U0029>	RIGHT PARENTHESIS
<asterisk>	*	<U002A>	ASTERISK
<plus-sign>	+	<U002B>	PLUS SIGN
<comma>	,	<U002C>	COMMA
<hyphen-minus>	-	<U002D>	HYPHEN-MINUS
<hyphen>	-	<U002D>	HYPHEN-MINUS
<full-stop>	.	<U002E>	FULL STOP
<period>	.	<U002E>	FULL STOP
<slash>	/	<U002F>	SOLIDUS
<solidus>	/	<U002F>	SOLIDUS
<zero>	0	<U0030>	DIGIT ZERO
<one>	1	<U0031>	DIGIT ONE
<two>	2	<U0032>	DIGIT TWO
<three>	3	<U0033>	DIGIT THREE

	Symbolic Name	Glyph	UCS	Description
3721	<four>	4	<U0034>	DIGIT FOUR
3722	<five>	5	<U0035>	DIGIT FIVE
3723	<six>	6	<U0036>	DIGIT SIX
3726	<seven>	7	<U0037>	DIGIT SEVEN
3727	<eight>	8	<U0038>	DIGIT EIGHT
3728	<nine>	9	<U0039>	DIGIT NINE
3729	<colon>	:	<U003A>	COLON
3730	<semicolon>	;	<U003B>	SEMICOLON
3731	<less-than-sign>	<	<U003C>	LESS-THAN SIGN
3732	<equals-sign>	=	<U003D>	EQUALS SIGN
3733	<greater-than-sign>	>	<U003E>	GREATER-THAN SIGN
3734	<question-mark>	?	<U003F>	QUESTION MARK
3735	<commercial-at>	@	<U0040>	<U0040>
3736	<A>	A	<U0041>	LATIN CAPITAL LETTER A
3737		B	<U0042>	LATIN CAPITAL LETTER B
3738	<C>	C	<U0043>	LATIN CAPITAL LETTER C
3739	<D>	D	<U0044>	LATIN CAPITAL LETTER D
3740	<E>	E	<U0045>	LATIN CAPITAL LETTER E
3741	<F>	F	<U0046>	LATIN CAPITAL LETTER F
3742	<G>	G	<U0047>	LATIN CAPITAL LETTER G
3743	<H>	H	<U0048>	LATIN CAPITAL LETTER H
3744	<I>	I	<U0049>	LATIN CAPITAL LETTER I
3745	<J>	J	<U004A>	LATIN CAPITAL LETTER J
3746	<K>	K	<U004B>	LATIN CAPITAL LETTER K
3747	<L>	L	<U004C>	LATIN CAPITAL LETTER L
3748	<M>	M	<U004D>	LATIN CAPITAL LETTER M
3749	<N>	N	<U004E>	LATIN CAPITAL LETTER N
3750	<O>	O	<U004F>	LATIN CAPITAL LETTER O
3751	<P>	P	<U0050>	LATIN CAPITAL LETTER P
3752	<Q>	Q	<U0051>	LATIN CAPITAL LETTER Q
3753	<R>	R	<U0052>	LATIN CAPITAL LETTER R
3754	<S>	S	<U0053>	LATIN CAPITAL LETTER S
3755	<T>	T	<U0054>	LATIN CAPITAL LETTER T
3756	<U>	U	<U0055>	LATIN CAPITAL LETTER U
3757	<V>	V	<U0056>	LATIN CAPITAL LETTER V
3758	<W>	W	<U0057>	LATIN CAPITAL LETTER W
3759	<X>	X	<U0058>	LATIN CAPITAL LETTER X
3760	<Y>	Y	<U0059>	LATIN CAPITAL LETTER Y
3761	<Z>	Z	<U005A>	LATIN CAPITAL LETTER Z
3762	<left-square-bracket>	[<U005B>	LEFT SQUARE BRACKET
3763	<backslash>	\	<U005C>	REVERSE SOLIDUS
3764	<reverse-solidus>	\	<U005C>	REVERSE SOLIDUS
3765	<right-square-bracket>]	<U005D>	RIGHT SQUARE BRACKET
3766	<circumflex-accent>	^	<U005E>	CIRCUMFLEX ACCENT
3767	<circumflex>	^	<U005E>	CIRCUMFLEX ACCENT
3768	<low-line>	_	<U005F>	LOW LINE
3769	<underscore>	_	<U005F>	LOW LINE

3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804

Symbolic Name	Glyph	UCS	Description
<grave-accent>	`	<U0060>	GRAVE ACCENT
<a>	a	<U0061>	LATIN SMALL LETTER A
	b	<U0062>	LATIN SMALL LETTER B
<c>	c	<U0063>	LATIN SMALL LETTER C
<d>	d	<U0064>	LATIN SMALL LETTER D
<e>	e	<U0065>	LATIN SMALL LETTER E
<f>	f	<U0066>	LATIN SMALL LETTER F
<g>	g	<U0067>	LATIN SMALL LETTER G
<h>	h	<U0068>	LATIN SMALL LETTER H
<i>	i	<U0069>	LATIN SMALL LETTER I
<j>	j	<U006A>	LATIN SMALL LETTER J
<k>	k	<U006B>	LATIN SMALL LETTER K
<l>	l	<U006C>	LATIN SMALL LETTER L
<m>	m	<U006D>	LATIN SMALL LETTER M
<n>	n	<U006E>	LATIN SMALL LETTER N
<o>	o	<U006F>	LATIN SMALL LETTER O
<p>	p	<U0070>	LATIN SMALL LETTER P
<q>	q	<U0071>	LATIN SMALL LETTER Q
<r>	r	<U0072>	LATIN SMALL LETTER R
<s>	s	<U0073>	LATIN SMALL LETTER S
<t>	t	<U0074>	LATIN SMALL LETTER T
<u>	u	<U0075>	LATIN SMALL LETTER U
<v>	v	<U0076>	LATIN SMALL LETTER V
<w>	w	<U0077>	LATIN SMALL LETTER W
<x>	x	<U0078>	LATIN SMALL LETTER X
<y>	y	<U0079>	LATIN SMALL LETTER Y
<z>	z	<U007A>	LATIN SMALL LETTER Z
<left-brace>	{	<U007B>	LEFT CURLY BRACKET
<left-curly-bracket>	{	<U007B>	LEFT CURLY BRACKET
<vertical-line>		<U007C>	VERTICAL LINE
<right-brace>	}	<U007D>	RIGHT CURLY BRACKET
<right-curly-bracket>	}	<U007D>	RIGHT CURLY BRACKET
<tilde>	~	<U007E>	TILDE

3805
3806
3807

IEEE Std. 1003.1-200x uses other character names than the above, but only in an informative way; for example, in examples to illustrate the use of characters beyond the portable character set with the facilities of IEEE Std. 1003.1-200x.

3808
3809
3810
3811

Table 6-1 (on page 133) defines the characters in the portable character set and the corresponding symbolic character names used to identify each character in a character set description file. The table contains more than one symbolic character name for characters whose traditional name differs from the chosen name.

3812
3813

IEEE Std. 1003.1-200x places only the following requirements on the encoded values of the characters in the portable character set:

3814
3815
3816

- If the encoded values associated with each member of the portable character set are not invariant across all locales supported by the implementation, the results achieved by an application accessing those locales are unspecified.

3817
3818

- The encoded values associated with the digits 0 to 9 shall be such that the value of each character after 0 shall be one greater than the value of the previous character.

- 3819 • A null character, NUL, which has all bits set to zero, shall be in the set of characters.
- 3820 • The encoded values associated with the members of the portable character set are each
3821 represented in a single byte. Moreover, if the value is stored in an object of C-language type
3822 **char**, it is guaranteed to be positive (except the NUL, which is always zero).
- 3823 Conforming implementations shall support certain character and character set attributes, as
3824 defined in Section 7.2 (on page 144).

3825 **6.2 Character Encoding**

3826 The POSIX locale contains the characters in Table 6-1 (on page 133), which have the properties
3827 listed in Section 7.3.1 (on page 147). Implementations may also add other characters. In other
3828 locales, the presence, meaning, and representation of any additional characters is locale-specific.

3829 In locales other than the POSIX locale, a character may have a state-dependent encoding. There
3830 are two types of these encodings:

- 3831 • A single-shift encoding (where each character not in the initial shift state is preceded by a
3832 shift code) can be defined if each shift-code and character sequence is considered a multi-
3833 byte character. This is done using the concatenated-constant format in a character set
3834 description file, as described in Section 6.4 (on page 137). If the implementation supports a
3835 character encoding of this type, all of the standard utilities in the Shell and Utilities volume of
3836 IEEE Std. 1003.1-200x support it. Use of a single-shift encoding with any of the functions in
3837 the System Interfaces volume of IEEE Std. 1003.1-200x that do not specifically mention the
3838 effects of state-dependent encoding is implementation-defined.
- 3839 • A locking-shift encoding (where the state of the character is determined by a shift code that
3840 may affect more than the single character following it) cannot be defined with the current
3841 character set description file format. Use of a locking-shift encoding with any of the standard
3842 utilities in the Shell and Utilities volume of IEEE Std. 1003.1-200x or with any of the functions
3843 in the System Interfaces volume of IEEE Std. 1003.1-200x that do not specifically mention the
3844 effects of state-dependent encoding is implementation-defined.

3845 While in the initial shift state, all characters in the portable character set retain their usual
3846 interpretation and do not alter the shift state. The interpretation for subsequent bytes in the
3847 sequence is a function of the current shift state. A byte with all bits zero is interpreted as the null
3848 character independent of shift state. Thus a byte with all bits zero shall never occur in the second
3849 or subsequent bytes of a character.

3850 The maximum allowable number of bytes in a character in the current locale is indicated by
3851 {MB_CUR_MAX}, defined in the `<stdlib.h>` header and by the `<mb_cur_max>` value in a
3852 character set description file; see Section 6.4 (on page 137). The implementation's maximum
3853 number of bytes in a character is defined by the C-language macro {MB_LEN_MAX}.

3854 6.3 C Language Wide-Character Codes

3855 In the shell, the standard utilities are written so that the encodings of characters are described by
 3856 the locale's *LC_CTYPE* definition (see Section 7.3.1 (on page 147)) and there is no differentiation
 3857 between characters consisting of single octets (8-bit bytes), larger bytes, or multiple bytes.
 3858 However, in the C language, a differentiation is made. To ease the handling of variable length
 3859 characters, the C language has introduced the concept of wide-character codes.

3860 All wide-character codes in a given process consist of an equal number of bits. This is in contrast
 3861 to characters, which can consist of a variable number of bytes. The byte or byte sequence that
 3862 represents a character can also be represented as a wide-character code. Wide-character codes
 3863 thus provide a uniform size for manipulating text data. A wide-character code having all bits
 3864 zero is the null wide-character code (see Section 3.248 (on page 83)), and terminates wide-
 3865 character strings (see Section 3.434 (on page 116)). The wide-character value for each member of
 3866 the Portable Character Set equals its value when used as the lone character in an integer
 3867 character constant. Wide-character codes for other characters are locale and implementation-
 3868 defined. State shift bytes do not have a wide-character code representation.

3869 6.4 Character Set Description File

3870 Implementations shall provide a character set description file for at least one coded character set
 3871 supported by the implementation. These files are referred to elsewhere in IEEE Std. 1003.1-200x
 3872 as *charmap* files. It is implementation-defined whether or not users or applications can provide
 3873 additional character set description files.

3874 IEEE Std. 1003.1-200x does not require that multiple character sets or codesets be supported.
 3875 Although multiple charmap files are supported, it is the responsibility of the implementation to
 3876 provide the file or files; if only one is provided, only that one is accessible using the *localedef*
 3877 utility's *-f* option.

3878 Each character set description file, except those that use the ISO/IEC 10646-1:1993 standard
 3879 position values as the encoding values, shall define characteristics for the coded character set
 3880 and the encoding for the characters specified in Table 6-1 (on page 133), and may define
 3881 encoding for additional characters supported by the implementation. Other information about
 3882 the coded character set may also be in the file. Coded character set character values shall be
 3883 defined using symbolic character names followed by character encoding values.

3884 Each symbolic name specified in Table 6-1 (on page 133) shall be included in the file and shall be
 3885 mapped to a unique encoding value (except for those symbolic names that are shown with
 3886 identical glyphs). If the control characters commonly associated with the symbolic names in the
 3887 following table are supported by the implementation, the symbolic names and their
 3888 corresponding encoding values shall be included in the file. Some of the encodings associated
 3889 with the symbolic names in this table may be the same as characters in the portable character set
 3890 table.

3891 **Table 6-2** Control Character Set

3892	<ACK>	<DC2>	<ENQ>	<FS>	<IS4>	<SOH>
3893	<BEL>	<DC3>	<EOT>	<GS>	<LF>	<STX>
3894	<BS>	<DC4>	<ESC>	<HT>	<NAK>	<SUB>
3895	<CAN>		<ETB>	<IS1>	<RS>	<SYN>
3896	<CR>	<DLE>	<ETX>	<IS2>	<SI>	<US>
3897	<DC1>		<FF>	<IS3>	<SO>	<VT>

3898 The following declarations can precede the character definitions. Each consists of the symbol
 3899 shown in the following list, starting in column 1, including the surrounding brackets, followed
 3900 by one or more <blank> characters, followed by the value to be assigned to the symbol.

3901 **<code_set_name>** The name of the coded character set for which the character set
 3902 description file is defined. The characters of the name shall be taken from
 3903 the set of characters with visible glyphs defined in Table 6-1 (on page
 3904 133).

3905 **<mb_cur_max>** The maximum number of bytes in a multi-byte character. This defaults to
 3906 1.

3907 **<mb_cur_min>** An unsigned positive integer value that defines the minimum number of
 3908 XSI bytes in a character for the encoded character set. On XSI-conformant
 3909 systems, **<mb_cur_min>** shall always be 1.

3910 **<escape_char>** The escape character used to indicate that the characters following are
 3911 interpreted in a special way, as defined later in this section. This defaults
 3912 to backslash ('\''), which is the character glyph used in all the following
 3913 text and examples, unless otherwise noted.

3914 **<comment_char>** The character that, when placed in column 1 of a charmap line, is used to
 3915 indicate that the line is to be ignored. The default character is the number
 3916 sign ('#').

3917 The character set mapping definitions shall be all the lines immediately following an identifier
 3918 line containing the string "CHARMAP" starting in column 1, and preceding a trailer line
 3919 containing the string "END CHARMAP" starting in column 1. Empty lines and lines containing a
 3920 **<comment_char>** in the first column shall be ignored. Each non-comment line of the character
 3921 set mapping definition (that is, between the "CHARMAP" and "END CHARMAP" lines of the file)
 3922 shall be in either of two forms:

3923 " %s %s %s\n", <symbolic-name>, <encoding>, <comments>

3924 or:

3925 " %s...%s %s %s\n", <symbolic-name>, <symbolic-name>,
 3926 <encoding>, <comments>

3927 In the first format, the line in the character set mapping definition defines a single symbolic
 3928 name and a corresponding encoding. A symbolic name is one or more characters from the set
 3929 shown with visible glyphs in Table 6-1 (on page 133), enclosed between angle brackets. A
 3930 character following an escape character is interpreted as itself; for example, the sequence
 3931 "<\\>" represents the symbolic name ">" enclosed between angle brackets.

3932 In the second format, the line in the character set mapping definition defines a range of one or
 3933 more symbolic names. In this form, the symbolic names shall consist of zero or more non-
 3934 numeric characters from the set shown with visible glyphs in Table 6-1 (on page 133), followed
 3935 by an integer formed by one or more decimal digits. Both integers shall contain the same number
 3936 of digits. The characters preceding the integer shall be identical in the two symbolic names, and
 3937 the integer formed by the digits in the second symbolic name shall be equal to or greater than the
 3938 integer formed by the digits in the first name. This shall be interpreted as a series of symbolic
 3939 names formed from the common part and each of the integers between the first and the second
 3940 integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the symbolic names
 3941 <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

3942 A character set mapping definition line shall exist for all symbolic names specified in Table 6-1
 3943 (on page 133), and defines the coded character value that corresponds to the character glyph

3944 indicated in the table, or the coded character value that corresponds to the control character
 3945 symbolic name. If the control characters commonly associated with the symbolic names in Table
 3946 6-2 (on page 137) are supported by the implementation, the symbolic name and the
 3947 corresponding encoding value shall be included in the file. Additional unique symbolic names
 3948 may be included. A coded character value can be represented by more than one symbolic name.

3949 The encoding part is expressed as one (for single-byte character values) or more concatenated
 3950 decimal, octal, or hexadecimal constants in the following formats:

```
3951     "%cd%u", <escape_char>, <decimal byte value>
3952     "%cx%x", <escape_char>, <hexadecimal byte value>
3953     "%co", <escape_char>, <octal byte value>
```

3954 Decimal constants are represented by two or three decimal digits, preceded by the escape
 3955 character and the lowercase letter 'd'; for example, "\d05", "\d97", or "\d143".
 3956 Hexadecimal constants are represented by two hexadecimal digits, preceded by the escape
 3957 character and the lowercase letter 'x'; for example, "\x05", "\x61", or "\x8f". Octal
 3958 constants are represented by two or three octal digits, preceded by the escape character; for
 3959 example, "\05", "\141", or "\217". In a portable charmap file, each constant represents an 8-
 3960 bit byte. Implementations supporting other byte sizes may allow constants to represent values
 3961 larger than those that can be represented in 8-bit bytes, and to allow additional digits in
 3962 constants. When constants are concatenated for multi-byte character values, they shall be of the
 3963 same type, and interpreted in byte order from first to last with the least significant byte of the
 3964 multi-byte character specified by the last constant. The manner in which these constants are
 3965 represented in the character stored in the system is implementation-defined. (This notation was
 3966 chosen for reasons of portability. There is no requirement that the internal representation in the
 3967 computer memory be in this same order.) Omitting bytes from a multi-byte character definition
 3968 produces undefined results.

3969 In lines defining ranges of symbolic names, the encoded value is the value for the first symbolic
 3970 name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic names
 3971 defined by the range shall have encoding values in increasing order. For example, the line:

```
3972     <j0101>...<j0104> \d129\d254
```

3973 is interpreted as:

```
3974     <j0101>          \d129\d254
3975     <j0102>          \d129\d255
3976     <j0103>          \d130\d0
3977     <j0104>          \d130\d1
```

3978 Note that this line is interpreted as the example even on systems with bytes larger than 8 bits.

3979 In lines defining ranges of symbolic names that also use the ISO/IEC 10646-1:1993 standard
 3980 position constant values, the conversion to the target codeset encoding value shall be performed
 3981 before assignment of encoding values to symbolic names.

3982 The comment is optional.

3983 The following declarations can follow the character set mapping definitions (after the "END
 3984 CHARMAP" statement). Each shall consist of the keyword shown in the following list, starting in
 3985 column 1, followed by the value(s) to be associated to the keyword, as defined below.

3986 **WIDTH** An unsigned positive integer value defining the column width (see Section 3.106
 3987 (on page 59)) for the printable characters in the coded character set specified in
 3988 Table 6-1 (on page 133) and Table 6-2 (on page 137).

3989 **Notes to Reviewers**3990 *This section with side shading will not appear in the final copy. - Ed.*3991 D3, XBD, ERN 90 suggests alternative wording for the text above: "the printable
3992 characters specified between the CHARMAP and END CHARMAP statements".
3993 The current wording is as per P1003.2b. When .2b is approved, an interpretation
3994 should be filed.3995 Coded character set character values shall be defined using symbolic character
3996 names followed by column width values. Defining a character with more than one
3997 **WIDTH** produces undefined results. The **END WIDTH** keyword shall be used to
3998 terminate the **WIDTH** definitions. Specifying the width of a non-printable
3999 character in a **WIDTH** declaration produces undefined results.4000 **WIDTH_DEFAULT**4001 An unsigned positive integer value defining the default column width for any
4002 printable character not listed by one of the **WIDTH** keywords. If no
4003 **WIDTH_DEFAULT** keyword is included in the charmap, the default character
4004 width shall be 1.4005 **Example**

4006 After the "END CHARMAP" statement, a syntax for a width definition would be:

4007 WIDTH
4008 <A> 1
4009 1
4010 <C>...<Z> 1
4011 <fool>...<foon> 2
4012 END WIDTH4013 In this example, the numerical code point values represented by the symbols <A> and are
4014 assigned a width of 1. The code point values <C> to <Z> inclusive (<C>, <D>, <E>, and so on)
4015 are also assigned a width of 1. Using <A>...<Z> would have required fewer lines, but the
4016 alternative was shown to demonstrate flexibility. The keyword **WIDTH_DEFAULT** could have
4017 been added as appropriate.4018 **6.4.1 State-Dependent Character Encodings**4019 This section addresses the use of state-dependent character encodings (that is, those in which the
4020 encoding of a character is dependent on one or more shift codes that may precede it).4021 A single-shift encoding (where each character not in the initial shift state is preceded by a shift
4022 code) can be defined in the charmap format if each shift-code/character sequence is considered a
4023 multi-byte character, defined using the concatenated-constant format described in Section 6.4
4024 (on page 137). If the implementation supports a character encoding of this type, all of the
4025 standard utilities shall support it. A locking-shift encoding (where the state of the character is
4026 determined by a shift code that may affect more than the single character following it) could be
4027 defined with an extension to the charmap format described in Section 6.4 (on page 137). If the
4028 implementation supports a character encoding of this type, any of the standard utilities that
4029 describe character (*versus* byte) or text-file manipulation shall have the following characteristics:

- 4030 1. The utility shall process the statefully encoded data as a concatenation of state-
-
- 4031 independent characters. The presence of redundant locking shifts shall not affect the
-
- 4032 comparison of two statefully encoded strings.

- 4033 2. A utility that divides, truncates, or extracts substrings from statefully encoded data shall
4034 produce output that contains locking shifts at the beginning or end of the resulting data, if
4035 appropriate, to retain correct state information.



4038 **7.1 General**

4039 A *locale* is the definition of the subset of a user's environment that depends on language and
 4040 cultural conventions. It is made up from one or more categories. Each category is identified by
 4041 its name and controls specific aspects of the behavior of components of the system. Category
 4042 names correspond to the following environment variable names:

4043 *LC_CTYPE* Character classification and case conversion.

4044 *LC_COLLATE* Collation order.

4045 *LC_TIME* Date and time formats.

4046 *LC_NUMERIC* Numeric, non-monetary formatting.

4047 *LC_MONETARY* Monetary formatting.

4048 *LC_MESSAGES* Formats of informative and diagnostic messages and interactive responses.

4049 The standard utilities in the Shell and Utilities volume of IEEE Std. 1003.1-200x shall base their
 4050 behavior on the current locale, as defined in the ENVIRONMENT VARIABLES section for each
 4051 utility. The behavior of some of the C-language functions defined in the System Interfaces
 4052 volume of IEEE Std. 1003.1-200x shall also be modified based on the current locale, as defined by
 4053 the last call to *setlocale()*.

4054 Locales other than those supplied by the implementation can be created via the *localedef* utility,
 4055 provided that the *_POSIX2_LOCALEDEF* symbol is defined on the system. Even if *localedef* is not
 4056 provided, all implementations conforming to the System Interfaces volume of
 4057 IEEE Std. 1003.1-200x shall provide one or more locales that behave as described in this chapter.
 4058 The input to the utility is described in Section 7.3 (on page 145). The value that is used to specify
 4059 a locale when using environment variables shall be the string specified as the *name* operand to
 4060 the *localedef* utility when the locale was created. The strings "C" and "POSIX" are reserved as
 4061 identifiers for the POSIX locale (see Section 7.2 (on page 144)). When the value of a locale
 4062 environment variable begins with a slash ('/'), it is interpreted as the path name of the locale
 4063 definition; the type of file (regular, directory, and so on) used to store the locale definition is
 4064 implementation-defined. If the value does not begin with a slash, the mechanism used to locate
 4065 the locale is implementation-defined.

4066 If different character sets are used by the locale categories, the results achieved by an application
 4067 utilizing these categories are undefined. Likewise, if different codesets are used for the data
 4068 being processed by interfaces whose behavior is dependent on the current locale, or the codeset
 4069 is different from the codeset assumed when the locale was created, the result is also undefined.

4070 Applications can select the desired locale by invoking the *setlocale()* function (or equivalent)
 4071 with the appropriate value. If the function is invoked with an empty string, such as:

```
4072     setlocale(LC_ALL, "");
```

4073 the value of the corresponding environment variable is used. If the environment variable is
 4074 unset or is set to the empty string, the implementation shall set the appropriate environment as
 4075 defined in Chapter 8 (on page 187).

4076 **7.2 POSIX Locale**

4077 Conforming systems shall provide a *POSIX locale*, also known as the C locale. The behavior of
4078 standard utilities and functions in the POSIX locale shall be as if the locale was defined via the
4079 *localedef* utility with input data from the POSIX locale tables in Section 7.3 (on page 145).

4080 The tables in Section 7.3 (on page 145) describe the characteristics and behavior of the POSIX
4081 locale for data consisting entirely of characters from the portable character set and the control
4082 character set. For other characters, the behavior is unspecified. For C-language programs, the
4083 POSIX locale is the default locale when the *setlocale()* function is not called.

4084 The POSIX locale can be specified by assigning to the appropriate environment variables the
4085 values "C" or "POSIX".

4086 All implementations shall define a locale as the default locale, to be invoked when no
4087 environment variables are set, or set to the empty string. This default locale can be the POSIX
4088 locale or any other implementation-defined locale. Some implementations may provide facilities |
4089 for local installation administrators to set the default locale, customizing it for each location. |
4090 IEEE Std. 1003.1-200x does not require such a facility.

4091 **7.3 Locale Definition**

4092 The capability to specify additional locales to those provided by an implementation is optional,
 4093 denoted by the `_POSIX2_LOCALEDEF` symbol. If the option is not supported, only
 4094 implementation-supplied locales are available.

4095 Locales can be described with the file format presented in this section. The file format is that
 4096 accepted by the *localedef* utility. For the purposes of this section, the file is referred to as the *locale*
 4097 *definition file*, but no locales shall be affected by this file unless it is processed by *localedef* or some
 4098 similar mechanism. Any requirements in this section imposed upon the utility shall apply to
 4099 *localedef* or to any other similar utility used to install locale information using the locale
 4100 definition file format described here.

4101 The locale definition file shall contain one or more locale category source definitions, and shall
 4102 not contain more than one definition for the same locale category. If the file contains source
 4103 definitions for more than one category, implementation-defined categories, if present, shall
 4104 appear after the categories defined by Section 7.1 (on page 143). A category source definition
 4105 contains either the definition of a category or a **copy** directive. For a description of the **copy**
 4106 directive, see *localedef*. In the event that some of the information for a locale category, as
 4107 specified in this volume of IEEE Std. 1003.1-200x, is missing from the locale source definition, the
 4108 behavior of that category, if it is referenced, is unspecified.

4109 A category source definition consists of a category header, a category body, and a category
 4110 trailer. A category header consists of the character string naming of the category, beginning with
 4111 the characters *LC_*. The category trailer consists of the string "END", followed by one or more
 4112 <blank> characters and the string used in the corresponding category header.

4113 The category body consists of one or more lines of text. Each line contains an identifier,
 4114 optionally followed by one or more operands. Identifiers are either keywords, identifying a
 4115 particular locale element, or collating elements. In addition to the keywords defined in this
 4116 volume of IEEE Std. 1003.1-200x, the source can contain implementation-defined keywords.
 4117 Each keyword within a locale has a unique name (that is, two categories cannot have a
 4118 commonly-named keyword); no keyword can start with the characters *LC_*. Identifiers are
 4119 separated from the operands by one or more <blank> characters.

4120 Operands shall be characters, collating elements, or strings of characters. Strings are enclosed in
 4121 double-quotes. Literal double-quotes within strings are preceded by the <escape character>,
 4122 described below. When a keyword is followed by more than one operand, the operands are
 4123 separated by semicolons; <blank> characters are allowed both before and after a semicolon.

4124 The first category header in the file can be preceded by a line modifying the comment character.
 4125 It shall have the following format, starting in column 1:

```
4126 "comment_char %c\n", <comment character>
```

4127 The comment character defaults to the number sign ('#'). Blank lines and lines containing the
 4128 <comment character> in the first position are ignored.

4129 The first category header in the file can be preceded by a line modifying the escape character to
 4130 be used in the file. It shall have the following format, starting in column 1:

```
4131 "escape_char %c\n", <escape character>
```

4132 The escape character defaults to backslash, which is the character used in all examples shown in
 4133 this volume of IEEE Std. 1003.1-200x.

4134 A line can be continued by placing an escape character as the last character on the line; this
 4135 continuation character is discarded from the input. Although the implementation need not
 4136 accept any one portion of a continued line with a length exceeding {LINE_MAX} bytes, it shall

4137 place no limits on the accumulated length of the continued line. Comment lines cannot be
4138 continued on a subsequent line using an escaped newline character.

4139 Individual characters, characters in strings, and collating elements shall be represented using
4140 symbolic names, as defined below. In addition, characters can be represented using the
4141 characters themselves or as octal, hexadecimal, or decimal constants. When non-symbolic
4142 notation is used, the resultant locale definitions are in many cases not portable between systems.
4143 The left angle bracket ('<') is a reserved symbol, denoting the start of a symbolic name; when
4144 used to represent itself it shall be preceded by the escape character. The following rules apply to
4145 character representation:

4146 1. A character can be represented via a symbolic name, enclosed within angle brackets '<'
4147 and '>'. The symbolic name, including the angle brackets, shall exactly match a symbolic
4148 name defined in the charmap file specified via the *localedef* -f option, and it shall be
4149 replaced by a character value determined from the value associated with the symbolic
4150 name in the charmap file. The use of a symbolic name not found in the charmap file shall
4151 constitute an error, unless the category is *LC_CTYPE* or *LC_COLLATE*, in which case it
4152 shall constitute a warning condition (see *localedef* for a description of action resulting from
4153 errors and warnings). The specification of a symbolic name in a **collating-element** or
4154 **collating-symbol** section that duplicates a symbolic name in the charmap file (if present)
4155 shall be an error. Use of the escape character or a right angle bracket within a symbolic
4156 name is invalid unless the character is preceded by the escape character.

4157 For example:

```
4158 <c>;<c-cedilla> " <M><a><y>"
```

4159 2. A character in the portable character set can be represented by the character itself, in which
4160 case the value of the character is implementation-defined. (Implementations may allow
4161 other characters to be represented as themselves, but such locale definitions are not
4162 portable.) Within a string, the double-quote character, the escape character, and the right
4163 angle bracket character shall be escaped (preceded by the escape character) to be
4164 interpreted as the character itself. Outside strings, the characters:

```
4165 , ; < > escape_char
```

4166 shall be escaped to be interpreted as the character itself.

4167 For example:

```
4168 c β "May"
```

4169 3. A character can be represented as an octal constant. An octal constant is specified as the
4170 escape character followed by two or three octal digits. Each constant represents a byte
4171 value. Multi-byte values can be represented by concatenated constants specified in byte
4172 order with the last constant specifying the least significant byte of the character.

4173 For example:

```
4174 \143;\347;\143\150 "\115\141\171"
```

4175 4. A character can be represented as a hexadecimal constant. A hexadecimal constant shall be
4176 specified as the escape character followed by an 'x' followed by two hexadecimal digits.
4177 Each constant shall represent a byte value. Multi-byte values can be represented by
4178 concatenated constants specified in byte order with the last constant specifying the least
4179 significant byte of the character.

4180 For example:

4181 \0x63;\0xe7;\0x63\0x68 "\0x4d\0x61\0x79"

4182 5. A character can be represented as a decimal constant. A decimal constant shall be specified
4183 as the escape character followed by a 'd' followed by two or three decimal digits. Each
4184 constant represents a byte value. Multi-byte values can be represented by concatenated
4185 constants specified in byte order with the last constant specifying the least significant byte
4186 of the character.

4187 For example:

4188 \0d99;\0d231;\0d99\0d104 "\0d77\0d97\0d121"

4189 Implementations supporting other byte sizes may allow constants to represent values larger
4190 than those that can be represented in 8-bit bytes, and to allow additional digits in constants.

4191 Implementations may accept single-digit octal, decimal, or hexadecimal constants following the
4192 escape character. Only characters existing in the character set for which the locale definition is
4193 created can be specified, whether using symbolic names, the characters themselves, or octal,
4194 decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the
4195 charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not
4196 present in the charmap file can be specified and shall be ignored, as specified under item 1
4197 above.

4198 7.3.1 LC_CTYPE

4199 The *LC_CTYPE* category shall define character classification, case conversion, and other
4200 character attributes. In addition, a series of characters can be represented by three adjacent
4201 periods representing an ellipsis symbol ("..."). The ellipsis specification shall be interpreted as
4202 meaning that all values between the values preceding and following it represent valid
4203 characters. The ellipsis specification is valid only within a single encoded character set; that is,
4204 within a group of characters of the same size. An ellipsis shall be interpreted as including in the
4205 list all characters with an encoded value higher than the encoded value of the character
4206 preceding the ellipsis and lower than the encoded value of the character following the ellipsis.

4207 For example:

4208 \0x30;...;\0x39;

4209 includes in the character class all characters with encoded values between the endpoints.

4210 The following keywords shall be recognized. In the descriptions, the term “automatically
4211 included” means that it shall not be an error either to include or omit any of the referenced
4212 characters; the implementation provides them if missing (even if the entire keyword is missing)
4213 and accepts them silently if present. When the implementation automatically includes a missing
4214 character, it shall have an encoded value dependent on the charmap file in effect (see the
4215 description of the *localedef -f* option); otherwise, it has a value derived from an implementation-
4216 defined character mapping.

4217 The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included
4218 characters. These only need to be specified if the character values (that is, encoding) differ from
4219 the implementation default values. It is not possible to define a locale without these
4220 automatically included characters unless some implementation extension is used to prevent
4221 their inclusion. Such a definition would not be a proper superset of the C or POSIX locale and
4222 thus, it might not be possible for conforming applications to work properly.

4223 **copy** Specify the name of an existing locale to be used as the definition of this
4224 category. If this keyword is specified, no other keyword can be specified.

4225	upper	Define characters to be classified as uppercase letters.
4226		In the POSIX locale, the 26 uppercase letters are included:
4227		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
4228		In a locale definition file, no character specified for the keywords cntrl , digit ,
4229		punct , or space can be specified. The uppercase letters <A> to <Z>, as defined
4230		in Section 6.4 (on page 137) (the portable character set), are automatically
4231		included in this class.
4232	lower	Define characters to be classified as lowercase letters.
4233		In the POSIX locale, the 26 lowercase letters are included:
4234		a b c d e f g h i j k l m n o p q r s t u v w x y z
4235		In a locale definition file, no character specified for the keywords cntrl , digit ,
4236		punct , or space can be specified. The lowercase letters <a> to <z> of the
4237		portable character set are automatically included in this class.
4238	alpha	Define characters to be classified as letters.
4239		In the POSIX locale, all characters in the classes upper and lower are included.
4240		In a locale definition file, no character specified for the keywords cntrl , digit ,
4241		punct , or space can be specified. Characters classified as either upper or lower
4242		are automatically included in this class.
4243	digit	Define the characters to be classified as numeric digits.
4244		In the POSIX locale, only:
4245		0 1 2 3 4 5 6 7 8 9
4246		are included.
4247		In a locale definition file, only the digits <zero>, <one>, <two>, <three>,
4248		<four>, <five>, <six>, <seven>, <eight>, and <nine> can be specified, and in
4249		contiguous ascending sequence by numerical value. The digits <zero> to
4250		<nine> of the portable character set are automatically included in this class.
4251		The definition of character class digit requires that only ten characters—the
4252		ones defining digits—can be specified; alternative digits (for example, Hindi
4253		or Kanji) cannot be specified here. However, the encoding may vary if an
4254		implementation supports more than one encoding.
4255	alnum	Define characters to be classified as letters and numeric digits. Only the
4256		characters specified for the alpha and digit keywords shall be specified.
4257		Characters specified for the keywords alpha and digit are automatically
4258		included in this class.
4259	space	Define characters to be classified as white-space characters.
4260		In the POSIX locale, at a minimum, the characters <space>, <form-feed>,
4261		<newline>, <carriage-return>, <tab>, and <vertical-tab> are included.
4262		In a locale definition file, no character specified for the keywords upper ,
4263		lower , alpha , digit , graph , or xdigit can be specified. The characters <space>,
4264		<form-feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> of the
4265		portable character set, and any characters included in the class blank are
4266		automatically included in this class.

4267	cntrl	Define characters to be classified as control characters.
4268		In the POSIX locale, no characters in classes alpha or print are included.
4269		In a locale definition file, no character specified for the keywords upper ,
4270		lower , alpha , digit , punct , graph , print , or xdigit can be specified.
4271	punct	Define characters to be classified as punctuation characters.
4272		In the POSIX locale, neither the <space> character nor any characters in
4273		classes alpha , digit , or cntrl are included.
4274		In a locale definition file, no character specified for the keywords upper ,
4275		lower , alpha , digit , cntrl , xdigit , or as the <space> character can be specified.
4276	graph	Define characters to be classified as printable characters, not including the
4277		<space> character.
4278		In the POSIX locale, all characters in classes alpha , digit , and punct are
4279		included; no characters in class cntrl are included.
4280		In a locale definition file, characters specified for the keywords upper , lower ,
4281		alpha , digit , xdigit , and punct are automatically included in this class. No
4282		character specified for the keyword cntrl can be specified.
4283	print	Define characters to be classified as printable characters, including the
4284		<space> character.
4285		In the POSIX locale, all characters in class graph are included; no characters in
4286		class cntrl are included.
4287		In a locale definition file, characters specified for the keywords upper , lower ,
4288		alpha , digit , xdigit , punct , graph , and the <space> character are automatically
4289		included in this class. No character specified for the keyword cntrl can be
4290		specified.
4291	xdigit	Define the characters to be classified as hexadecimal digits.
4292		In the POSIX locale, only:
4293		0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
4294		are included.
4295		In a locale definition file, only the characters defined for the class digit can be
4296		specified, in contiguous ascending sequence by numerical value, followed by
4297		one or more sets of six characters representing the hexadecimal digits 10 to 15
4298		inclusive, with each set in ascending order (for example, <A>, , <C>, <D>,
4299		<E>, <F>, <a>, , <c>, <d>, <e>, <f>). The digits <zero> to <nine>, the
4300		uppercase letters <A> to <F>, and the lowercase letters <a> to <f> of the
4301		portable character set are automatically included in this class.
4302		The definition of character class xdigit requires that the characters included in
4303		character class digit be included here also.
4304	blank	Define characters to be classified as <blank> characters.
4305		In the POSIX locale, only the <space> and <tab> characters are included.
4306		In a locale definition file, the characters <space> and <tab> are automatically
4307		included in this class.

4308	charclass	Define one or more locale-specific character class names as strings separated by semicolons. Each named character class can then be defined subsequently in the <i>LC_CTYPE</i> definition. A character class name consists of at least one and at most {CHARCLASS_NAME_MAX} bytes of alphanumeric characters from the portable file name character set. The first character of a character class name cannot be a digit. The name cannot match any of the <i>LC_CTYPE</i> keywords defined in this volume of IEEE Std. 1003.1-200x. Future revisions of IEEE Std. 1003.1-200x will not specify any <i>LC_CTYPE</i> keywords containing uppercase letters.
4309		
4310		
4311		
4312		
4313		
4314		
4315		
4316		
4317	<i>charclass-name</i>	Define characters to be classified as belonging to the named locale-specific character class. In the POSIX locale, the locale-specific named character classes need not exist.
4318		
4319		
4320		If a class name is defined by a charclass keyword, but no characters are subsequently assigned to it, this is not an error; it represents a class without any characters belonging to it.
4321		
4322		
4323		The <i>charclass-name</i> can be used as the <i>property</i> argument to the <i>wctype()</i> function, in regular expression and shell pattern-matching bracket expressions, and by the <i>tr</i> command.
4324		
4325		
4326	toupper	Define the mapping of lowercase letters to uppercase letters.
4327		In the POSIX locale, at a minimum, the 26 lowercase characters:
4328		a b c d e f g h i j k l m n o p q r s t u v w x y z
4329		are mapped to the corresponding 26 uppercase characters:
4330		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
4331		In a locale definition file, the operand consists of character pairs, separated by semicolons. The characters in each character pair are separated by a comma and the pair enclosed by parentheses. The first character in each pair is the lowercase letter, the second the corresponding uppercase letter. Only characters specified for the keywords lower and upper can be specified. The lowercase letters <a> to <z>, and their corresponding uppercase letters <A> to <Z>, of the portable character set are automatically included in this mapping, but only when the toupper keyword is omitted from the locale definition.
4332		
4333		
4334		
4335		
4336		
4337		
4338		
4339	tolower	Define the mapping of uppercase letters to lowercase letters.
4340		In the POSIX locale, at a minimum, the 26 uppercase characters:
4341		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
4342		are mapped to the corresponding 26 lowercase characters:
4343		a b c d e f g h i j k l m n o p q r s t u v w x y z
4344		In a locale definition file, the operand consists of character pairs, separated by semicolons. The characters in each character pair are separated by a comma and the pair enclosed by parentheses. The first character in each pair is the uppercase letter, the second the corresponding lowercase letter. Only characters specified for the keywords lower and upper can be specified. If the tolower keyword is omitted from the locale definition, the mapping is the reverse mapping of the one specified for toupper .
4345		
4346		
4347		
4348		
4349		
4350		

4351 The following table shows the character class combinations allowed:

4352 **Table 7-1** Valid Character Class Combinations

4353 4354 In Class	4353 Can Also Belong To										
	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank
4355 upper	—	—	A	x	x	x	x	A	A	—	x
4356 lower	—	—	A	x	x	x	x	A	A	—	x
4357 alpha	—	—	—	x	x	x	x	A	A	—	x
4358 digit	x	x	x	x	x	x	x	A	A	A	x
4359 space	x	x	x	x	—	—	*	*	*	x	—
4360 cntrl	x	x	x	x	—	—	x	x	x	x	—
4361 punct	x	x	x	x	—	x	—	A	A	x	—
4362 graph	—	—	—	—	—	x	—	—	A	—	—
4363 print	—	—	—	—	—	x	—	—	—	—	—
4364 xdigit	—	—	—	—	x	x	x	A	A	—	x
4365 blank	x	x	x	x	A	—	*	*	*	x	—

4366 **Notes:**

- 4367 1. Explanation of codes:
- 4368 A Automatically included; see text.
- 4369 — Permitted.
- 4370 x Mutually-exclusive.
- 4371 * See note 2.
- 4372 2. The <space> character, which is part of the **space** and **blank** classes, cannot
- 4373 belong to **punct** or **graph**, but automatically belongs to the **print** class. Other
- 4374 **space** or **blank** characters can be classified as any of **punct**, **graph**, or **print**.

4375 The character classifications for the POSIX locale follow; the code listing depicting the *localedef*

4376 input, the table representing the same information, sorted by character.

```

4377 LC_CTYPE
4378 # The following is the POSIX locale LC_CTYPE.
4379 # "alpha" is by default "upper" and "lower"
4380 # "alnum" is by definition "alpha" and "digit"
4381 # "print" is by default "alnum", "punct" and the <space> character
4382 # "graph" is by default "alnum" and "punct"
4383 #
4384 upper <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
4385 <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
4386 #
4387 lower <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
4388 <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
4389 #
4390 digit <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
4391 <seven>;<eight>;<nine>
4392 #
4393 space <tab>;<newline>;<vertical-tab>;<form-feed>;\
4394 <carriage-return>;<space>
4395 #
4396 cntrl <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\

```

```

4397         <form-feed>;<carriage-return>;\
4398         <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
4399         <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
4400         <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
4401         <IS1>;<DEL>
4402     #
4403     punct <exclamation-mark>;<quotation-mark>;<number-sign>;\
4404           <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
4405           <left-parenthesis>;<right-parenthesis>;<asterisk>;\
4406           <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
4407           <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
4408           <greater-than-sign>;<question-mark>;<commercial-at>;\
4409           <left-square-bracket>;<backslash>;<right-square-bracket>;\
4410           <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
4411           <vertical-line>;<right-curly-bracket>;<tilde>
4412     #
4413     xdigit <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
4414           <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
4415     #
4416     blank <space>;<tab>
4417     #
4418     toupper (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
4419            (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
4420            (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
4421            (<p>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
4422            (<u>,<U>);(<v>,<V>);(<w>,<W>);(<x>,<X>);(<y>,<Y>);(<z>,<Z>)
4423     #
4424     tolower (<A>,<a>);(<B>,<b>);(<C>,<c>);(<D>,<d>);(<E>,<e>);\
4425            (<F>,<f>);(<G>,<g>);(<H>,<h>);(<I>,<i>);(<J>,<j>);\
4426            (<K>,<k>);(<L>,<l>);(<M>,<m>);(<N>,<n>);(<O>,<o>);\
4427            (<P>,<p>);(<Q>,<q>);(<R>,<r>);(<S>,<s>);(<T>,<t>);\
4428            (<U>,<u>);(<V>,<v>);(<W>,<w>);(<X>,<x>);(<Y>,<y>);(<Z>,<z>)
4429     END LC_CTYPE

```

4430
4431
4432
4433
4434
4435
4436
4437
4438
4439
4440
4441
4442
4443
4444
4445
4446
4447
4448
4449
4450
4451
4452
4453
4454
4455
4456
4457
4458
4459
4460
4461
4462
4463
4464
4465
4466
4467
4468
4469
4470
4471
4472
4473
4474
4475
4476
4477
4478

Symbolic Name	Other Case	Character Classes
<NUL>		cntrl
<SOH>		cntrl
<STX>		cntrl
<ETX>		cntrl
<EOT>		cntrl
<ENQ>		cntrl
<ACK>		cntrl
<alert>		cntrl
<backspace>		cntrl
<tab>		cntrl, space, blank
<newline>		cntrl, space
<vertical-tab>		cntrl, space
<form-feed>		cntrl, space
<carriage-return>		cntrl, space
<SO>		cntrl
<SI>		cntrl
<DLE>		cntrl
<DC1>		cntrl
<DC2>		cntrl
<DC3>		cntrl
<DC4>		cntrl
<NAK>		cntrl
<SYN>		cntrl
<ETB>		cntrl
<CAN>		cntrl
		cntrl
<SUB>		cntrl
<ESC>		cntrl
<IS4>		cntrl
<IS3>		cntrl
<IS2>		cntrl
<IS1>		cntrl
<space>		space, print, blank
<exclamation-mark>		punct, print, graph
<quotation-mark>		punct, print, graph
<number-sign>		punct, print, graph
<dollar-sign>		punct, print, graph
<percent-sign>		punct, print, graph
<ampersand>		punct, print, graph
<apostrophe>		punct, print, graph
<left-parenthesis>		punct, print, graph
<right-parenthesis>		punct, print, graph
<asterisk>		punct, print, graph
<plus-sign>		punct, print, graph
<comma>		punct, print, graph
<hyphen>		punct, print, graph
<period>		punct, print, graph

4479
4480
4481
4482
4483
4484
4485
4486
4487
4488
4489
4490
4491
4492
4493
4494
4495
4496
4497
4498
4499
4500
4501
4502
4503
4504
4505
4506
4507
4508
4509
4510
4511
4512
4513
4514
4515
4516
4517
4518
4519
4520
4521
4522
4523
4524
4525
4526
4527

Symbolic Name	Other Case	Character Classes
<slash>		punct, print, graph
<zero>		digit, xdigit, print, graph
<one>		digit, xdigit, print, graph
<two>		digit, xdigit, print, graph
<three>		digit, xdigit, print, graph
<four>		digit, xdigit, print, graph
<five>		digit, xdigit, print, graph
<six>		digit, xdigit, print, graph
<seven>		digit, xdigit, print, graph
<eight>		digit, xdigit, print, graph
<nine>		digit, xdigit, print, graph
<colon>		punct, print, graph
<semicolon>		punct, print, graph
<less-than-sign>		punct, print, graph
<equals-sign>		punct, print, graph
<greater-than-sign>		punct, print, graph
<question-mark>		punct, print, graph
<commercial-at>		punct, print, graph
<A>	<a>	upper, xdigit, alpha, print, graph
		upper, xdigit, alpha, print, graph
<C>	<c>	upper, xdigit, alpha, print, graph
<D>	<d>	upper, xdigit, alpha, print, graph
<E>	<e>	upper, xdigit, alpha, print, graph
<F>	<f>	upper, xdigit, alpha, print, graph
<G>	<g>	upper, alpha, print, graph
<H>	<h>	upper, alpha, print, graph
<I>	<i>	upper, alpha, print, graph
<J>	<j>	upper, alpha, print, graph
<K>	<k>	upper, alpha, print, graph
<L>	<l>	upper, alpha, print, graph
<M>	<m>	upper, alpha, print, graph
<N>	<n>	upper, alpha, print, graph
<O>	<o>	upper, alpha, print, graph
<P>	<p>	upper, alpha, print, graph
<Q>	<q>	upper, alpha, print, graph
<R>	<r>	upper, alpha, print, graph
<S>	<s>	upper, alpha, print, graph
<T>	<t>	upper, alpha, print, graph
<U>	<u>	upper, alpha, print, graph
<V>	<v>	upper, alpha, print, graph
<W>	<w>	upper, alpha, print, graph
<X>	<x>	upper, alpha, print, graph
<Y>	<y>	upper, alpha, print, graph
<Z>	<z>	upper, alpha, print, graph
<left-square-bracket>		punct, print, graph
<backslash>		punct, print, graph
<right-square-bracket>		punct, print, graph

4528
4529
4530
4531
4532
4533
4534
4535
4536
4537
4538
4539
4540
4541
4542
4543
4544
4545
4546
4547
4548
4549
4550
4551
4552
4553
4554
4555
4556
4557
4558
4559
4560
4561
4562
4563

Symbolic Name	Other Case	Character Classes
<circumflex>		punct, print, graph
<underscore>		punct, print, graph
<grave-accent>		punct, print, graph
<a>	<A>	lower, xdigit, alpha, print, graph
		lower, xdigit, alpha, print, graph
<c>	<C>	lower, xdigit, alpha, print, graph
<d>	<D>	lower, xdigit, alpha, print, graph
<e>	<E>	lower, xdigit, alpha, print, graph
<f>	<F>	lower, xdigit, alpha, print, graph
<g>	<G>	lower, alpha, print, graph
<h>	<H>	lower, alpha, print, graph
<i>	<I>	lower, alpha, print, graph
<j>	<J>	lower, alpha, print, graph
<k>	<K>	lower, alpha, print, graph
<l>	<L>	lower, alpha, print, graph
<m>	<M>	lower, alpha, print, graph
<n>	<N>	lower, alpha, print, graph
<o>	<O>	lower, alpha, print, graph
<p>	<P>	lower, alpha, print, graph
<q>	<Q>	lower, alpha, print, graph
<r>	<R>	lower, alpha, print, graph
<s>	<S>	lower, alpha, print, graph
<t>	<T>	lower, alpha, print, graph
<u>	<U>	lower, alpha, print, graph
<v>	<V>	lower, alpha, print, graph
<w>	<W>	lower, alpha, print, graph
<x>	<X>	lower, alpha, print, graph
<y>	<Y>	lower, alpha, print, graph
<z>	<Z>	lower, alpha, print, graph
<left-curly-bracket>		punct, print, graph
<vertical-line>		punct, print, graph
<right-curly-bracket>		punct, print, graph
<tilde>		punct, print, graph
		cntrl

4564 **7.3.2 LC_COLLATE**

4565 The *LC_COLLATE* category provides a collation sequence definition for numerous utilities in the
4566 Shell and Utilities volume of IEEE Std. 1003.1-200x (*sort*, *uniq*, and so on), regular expression
4567 matching (see Chapter 9 (on page 195)) and the *strcoll()*, *strxfrm()*, *wscoll()*, and *wcsxfrm()*
4568 functions in the System Interfaces volume of IEEE Std. 1003.1-200x.

4569 A collation sequence definition shall define the relative order between collating elements
4570 (characters and multi-character collating elements) in the locale. This order is expressed in terms
4571 of collation values; that is, by assigning each element one or more collation values (also known
4572 as collation weights). This does not imply that implementations shall assign such values, but
4573 that ordering of strings using the resultant collation definition in the locale behaves as if such
4574 assignment is done and used in the collation process. At least the following capabilities are
4575 provided:

- 4576 1. **Multi-character collating elements.** Specification of multi-character collating elements
4577 (that is, sequences of two or more characters to be collated as an entity).

- 4578 2. **User-defined ordering of collating elements.** Each collating element shall be assigned a
 4579 collation value defining its order in the character (or basic) collation sequence. This
 4580 ordering is used by regular expressions and pattern matching and, unless collation weights
 4581 are explicitly specified, also as the collation weight to be used in sorting.
- 4582 3. **Multiple weights and equivalence classes.** Collating elements can be assigned one or
 4583 more (up to the limit {COLL_WEIGHTS_MAX}, as defined in <limits.h>) collating weights
 4584 for use in sorting. The first weight is hereafter referred to as the primary weight.
- 4585 4. **One-to-many mapping.** A single character is mapped into a string of collating elements.
- 4586 5. **Equivalence class definition.** Two or more collating elements have the same collation
 4587 value (primary weight).
- 4588 6. **Ordering by weights.** When two strings are compared to determine their relative order,
 4589 the two strings are first broken up into a series of collating elements; the elements in each
 4590 successive pair of elements are then compared according to the relative primary weights
 4591 for the elements. If equal, and more than one weight has been assigned, then the pairs of
 4592 collating elements are recompared according to the relative subsequent weights, until
 4593 either a pair of collating elements compare unequal or the weights are exhausted.

4594 The following keywords shall be recognized in a collation sequence definition. They are
 4595 described in detail in the following sections.

4596	copy	Specify the name of an existing locale to be used as the definition of this category. If this keyword is specified, no other keyword can be specified.
4597		
4598	collating-element	Define a collating-element symbol representing a multi-character collating element. This keyword is optional.
4599		
4600	collating-symbol	Define a collating symbol for use in collation order statements. This keyword is optional.
4601		
4602	order_start	Define collation rules. This statement is followed by one or more collation order statements, assigning character collation values and collation weights to collating elements.
4603		
4604		
4605	order_end	Specify the end of the collation-order statements.

4606 7.3.2.1 *The collating-element Keyword*

4607 In addition to the collating elements in the character set, the **collating-element** keyword can be
 4608 used to define multi-character collating elements. The syntax is as follows:

```
4609     "collating-element %s from \"%s\"\\n", <collating-symbol>, <string>
```

4610 The <collating-symbol> operand shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand is a string of two or more characters that collates as an entity. A <collating-element> defined via this keyword is only recognized with the *LC_COLLATE* category.

4615 For example:

```
4616     collating-element <ch> from "<c><h>"
4617     collating-element <e-acute> from "<acute><e>"
4618     collating-element <ll> from "ll"
```


4619 7.3.2.2 *The collating-symbol Keyword*

4620 This keyword can be used to define symbols for use in collation sequence statements; that is,
4621 between the **order_start** and the **order_end** keywords. The syntax is as follows:

```
4622 "collating-symbol %s\n", <collating-symbol>
```

4623 The *<collating-symbol>* shall be a symbolic name, enclosed between angle brackets ('<' and
4624 '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any
4625 other symbolic name defined in this collation definition. A *<collating-symbol>* defined via this
4626 keyword is only recognized with the *LC_COLLATE* category.

4627 For example:

```
4628 collating-symbol <UPPER_CASE>  
4629 collating-symbol <HIGH>
```

4630 The **collating-symbol** keyword defines a symbolic name that can be associated with a relative
4631 position in the character order sequence. While such a symbolic name does not represent any
4632 collating element, it can be used as a weight.

4633 7.3.2.3 *The order_start Keyword*

4634 The **order_start** keyword shall precede collation order entries and also define the number of
4635 weights for this collation sequence definition and other collation rules.

4636 The syntax of the **order_start** keyword is as follows:

```
4637 "order_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...
```

4638 The operands to the **order_start** keyword are optional. If present, the operands define rules to be
4639 applied when strings are compared. The number of operands define how many weights each
4640 element is assigned; if no operands are present, one **forward** operand is assumed. If present, the
4641 first operand defines rules to be applied when comparing strings using the first (primary)
4642 weight; the second when comparing strings using the second weight, and so on. Operands shall
4643 be separated by semicolons (';'). Each operand shall consist of one or more collation
4644 directives, separated by commas (','),. If the number of operands exceeds the
4645 {COLL_WEIGHTS_MAX} limit, the utility shall issue a warning message. The following
4646 directives shall be supported:

4647 **forward** Specifies that comparison operations for the weight level shall proceed from start
4648 of string towards the end of string.

4649 **backward** Specifies that comparison operations for the weight level shall proceed from end of
4650 string towards the beginning of string.

4651 **position** Specifies that comparison operations for the weight level shall consider the relative
4652 position of elements in the strings not subject to **IGNORE**. The string containing
4653 an element not subject to **IGNORE** after the fewest collating elements subject to
4654 **IGNORE** from the start of the compare collates first. If both strings contain a
4655 character not subject to **IGNORE** in the same relative position, the collating values
4656 assigned to the elements shall determine the ordering. In case of equality,
4657 subsequent characters not subject to **IGNORE** are considered in the same manner.

4658 The directives **forward** and **backward** are mutually-exclusive.

4659 For example:

```
4660 order_start forward;backward
```

4661 If no operands are specified, a single **forward** operand shall be assumed.

4662 The character (and collating element) order is defined by the order in which characters and
4663 elements are specified between the **order_start** and **order_end** keywords. This character order is
4664 used in range expressions in regular expressions (see Chapter 9). Weights assigned to the
4665 characters and elements define the collation sequence; in the absence of weights, the character
4666 order is also the collation sequence.

4667 The **position** keyword provides the capability to consider, in a compare, the relative position of
4668 characters not subject to **IGNORE**. As an example, consider the two strings "o-ring" and
4669 "or-ing". Assuming the hyphen is subject to **IGNORE** on the first pass, the two strings
4670 compare equal, and the position of the hyphen is immaterial. On second pass, all characters
4671 except the hyphen are subject to **IGNORE**, and in the normal case the two strings would again
4672 compare equal. By taking position into account, the first collates before the second.

4673 7.3.2.4 Collation Order

4674 The **order_start** keyword shall be followed by collating identifier entries. The syntax for the
4675 collating element entries is as follows:

```
4676 "%s %s;%s;...;%s\n", <collating-identifier>, <weight>, <weight>, ...
```

4677 Each *collating-identifier* shall consist of either a character (in any of the forms defined in Section
4678 7.3 (on page 145)), a *collating-element*, a *collating-symbol*, an ellipsis, or the special symbol
4679 **UNDEFINED**. The order in which collating elements are specified determines the character
4680 order sequence, such that each collating element shall compare less than the elements following
4681 it.

4682 A *collating-element* shall be used to specify multi-character collating elements, and indicates
4683 that the character sequence specified via the *collating-element* is to be collated as a unit and in
4684 the relative order specified by its place.

4685 A *collating-symbol* can be used to define a position in the relative order for use in weights. No
4686 weights can be specified with a *collating-symbol*.

4687 The ellipsis symbol specifies that a sequence of characters collates according to their encoded
4688 character values. It shall be interpreted as indicating that all characters with a coded character
4689 set value higher than the value of the character in the preceding line, and lower than the coded
4690 character set value for the character in the following line, in the current coded character set, shall
4691 be placed in the character collation order between the previous and the following character in
4692 ascending order according to their coded character set values. An initial ellipsis shall be
4693 interpreted as if the preceding line specified the NUL character, and a trailing ellipsis as if the
4694 following line specified the highest coded character set value in the current coded character set.
4695 An ellipsis shall be treated as invalid if the preceding or following lines do not specify characters
4696 in the current coded character set. The use of the ellipsis symbol ties the definition to a specific
4697 coded character set and may preclude the definition from being portable between
4698 implementations.

4699 The symbol **UNDEFINED** shall be interpreted as including all coded character set values not
4700 specified explicitly or via the ellipsis symbol. Such characters shall be inserted in the character
4701 collation order at the point indicated by the symbol, and in ascending order according to their
4702 coded character set values. If no **UNDEFINED** symbol is specified, and the current coded
4703 character set contains characters not specified in this section, the utility shall issue a warning
4704 message and place such characters at the end of the character collation order.

4705 The optional operands for each collation-element shall be used to define the primary, secondary,
4706 or subsequent weights for the collating element. The first operand specifies the relative primary

4707 weight, the second the relative secondary weight, and so on. Two or more collation-elements can
 4708 be assigned the same weight; they belong to the same *equivalence class* if they have the same
 4709 primary weight. Collation shall behave as if, for each weight level, elements subject to **IGNORE**
 4710 are removed, unless the **position** collation directive is specified for the corresponding level with
 4711 the **order_start** keyword. Then each successive pair of elements shall be compared according to
 4712 the relative weights for the elements. If the two strings compare equal, the process is repeated
 4713 for the next weight level, up to the limit {COLL_WEIGHTS_MAX}.

4714 Weights shall be expressed as characters (in any of the forms specified in Section 7.3 (on page
 4715 145)), *<collating-symbol>*s, *<collating-element>*s, an ellipsis, or the special symbol **IGNORE**. A
 4716 single character, a *<collating-symbol>*, or a *<collating-element>* shall represent the relative position
 4717 in the character collating sequence of the character or symbol, rather than the character or
 4718 characters themselves. Thus, rather than assigning absolute values to weights, a particular
 4719 weight is expressed using the relative order value assigned to a collating element based on its
 4720 order in the character collation sequence.

4721 One-to-many mapping is indicated by specifying two or more concatenated characters or
 4722 symbolic names. For example, if the character `<eszet>` is given the string "`<s><s>`" as a weight,
 4723 comparisons are performed as if all occurrences of the character `<eszet>` are replaced by
 4724 "`<s><s>`" (assuming that "`<s>`" has the collating weight "`<s>`"). If it is necessary to define
 4725 `<eszet>` and "`<s><s>`" as an equivalence class, then a collating element must be defined for the
 4726 string "ss".

4727 All characters specified via an ellipsis shall by default be assigned unique weights, equal to the
 4728 relative order of characters. Characters specified via an explicit or implicit **UNDEFINED** special
 4729 symbol shall by default be assigned the same primary weight (that is, they belong to the same
 4730 equivalence class). An ellipsis symbol as a weight shall be interpreted to mean that each
 4731 character in the sequence has unique weights, equal to the relative order of their character in the
 4732 character collation sequence. The use of the ellipsis as a weight shall be treated as an error if the
 4733 collating element is neither an ellipsis nor the special symbol **UNDEFINED**.

4734 The special keyword **IGNORE** as a weight shall indicate that when strings are compared using
 4735 the weights at the level where **IGNORE** is specified, the collating element shall be ignored; that
 4736 is, as if the string did not contain the collating element. In regular expressions and pattern
 4737 matching, all characters that are subject to **IGNORE** in their primary weight form an
 4738 equivalence class.

4739 An empty operand shall be interpreted as the collating element itself.

4740 For example, the order statement:

4741 `<a> <a> i <a>`

4742 is equal to:

4743 `<a>`

4744 An ellipsis can be used as an operand if the collating element was an ellipsis, and shall be
 4745 interpreted as the value of each character defined by the ellipsis.

4746 The collation order as defined in this section defines the interpretation of bracket expressions in
 4747 regular expressions (see Section 9.3.5 (on page 199)).

4748 For example:

```

4749     order_start  forward;backward
4750     UNDEFINED   IGNORE;IGNORE
4751     <LOW>
4752     <space>     <LOW>;<space>
4753     ...         <LOW>;...
4754     <a>         <a>;<a>
4755     <a-acute>   <a>;<a-acute>
4756     <a-grave>   <a>;<a-grave>
4757     <A>         <a>;<A>
4758     <A-acute>   <a>;<A-acute>
4759     <A-grave>   <a>;<A-grave>
4760     <ch>       <ch>;<ch>
4761     <Ch>       <ch>;<Ch>
4762     <s>        <s>;<s>
4763     <eszet>    "<s><s>";"<eszet><eszet>"
4764     order_end

```

4765 This example is interpreted as follows:

- 4766 1. The **UNDEFINED** means that all characters not specified in this definition (explicitly or
4767 via the ellipsis) shall be ignored for collation purposes; for regular expression purposes
4768 they are ordered first.
- 4769 2. All characters between <space> and 'a' shall have the same primary equivalence class
4770 and individual secondary weights based on their ordinal encoded values.
- 4771 3. All characters based on the uppercase or lowercase character 'a' belong to the same
4772 primary equivalence class.
- 4773 4. The multi-character collating element <ch> is represented by the collating symbol <ch>
4774 and belongs to the same primary equivalence class as the multi-character collating element
4775 <Ch>.

4776 7.3.2.5 The order_end Keyword

4777 The collating order entries shall be terminated with an **order_end** keyword.

4778 The collation sequence definition of the POSIX locale follows; the code listing depicts the
4779 *localedef* input.

```

4780 LC_COLLATE
4781 # This is the POSIX locale definition for the LC_COLLATE category.
4782 # The order is the same as in the ASCII codeset.
4783 order_start forward
4784 <NUL>
4785 <SOH>
4786 <STX>
4787 <ETX>
4788 <EOT>
4789 <ENQ>
4790 <ACK>
4791 <alert>
4792 <backspace>
4793 <tab>
4794 <newline>
4795 <vertical-tab>

```

4796	<form-feed>
4797	<carriage-return>
4798	<SO>
4799	<SI>
4800	<DLE>
4801	<DC1>
4802	<DC2>
4803	<DC3>
4804	<DC4>
4805	<NAK>
4806	<SYN>
4807	<ETB>
4808	<CAN>
4809	
4810	<SUB>
4811	<ESC>
4812	<IS4>
4813	<IS3>
4814	<IS2>
4815	<IS1>
4816	<space>
4817	<exclamation-mark>
4818	<quotation-mark>
4819	<number-sign>
4820	<dollar-sign>
4821	<percent-sign>
4822	<ampersand>
4823	<apostrophe>
4824	<left-parenthesis>
4825	<right-parenthesis>
4826	<asterisk>
4827	<plus-sign>
4828	<comma>
4829	<hyphen>
4830	<period>
4831	<slash>
4832	<zero>
4833	<one>
4834	<two>
4835	<three>
4836	<four>
4837	<five>
4838	<six>
4839	<seven>
4840	<eight>
4841	<nine>
4842	<colon>
4843	<semicolon>
4844	<less-than-sign>
4845	<equals-sign>
4846	<greater-than-sign>
4847	<question-mark>

4848 <commercial-at>
4849 <A>
4850
4851 <C>
4852 <D>
4853 <E>
4854 <F>
4855 <G>
4856 <H>
4857 <I>
4858 <J>
4859 <K>
4860 <L>
4861 <M>
4862 <N>
4863 <O>
4864 <P>
4865 <Q>
4866 <R>
4867 <S>
4868 <T>
4869 <U>
4870 <V>
4871 <W>
4872 <X>
4873 <Y>
4874 <Z>
4875 <left-square-bracket>
4876 <backslash>
4877 <right-square-bracket>
4878 <circumflex>
4879 <underscore>
4880 <grave-accent>
4881 <a>
4882
4883 <c>
4884 <d>
4885 <e>
4886 <f>
4887 <g>
4888 <h>
4889 <i>
4890 <j>
4891 <k>
4892 <l>
4893 <m>
4894 <n>
4895 <o>
4896 <p>
4897 <q>
4898 <r>
4899 <s>

```

4900     <t>
4901     <u>
4902     <v>
4903     <w>
4904     <x>
4905     <y>
4906     <z>
4907     <left-curly-bracket>
4908     <vertical-line>
4909     <right-curly-bracket>
4910     <tilde>
4911     <DEL>
4912     order_end
4913     #
4914     END LC_COLLATE

```

4915 7.3.3 LC_MONETARY

4916 The *LC_MONETARY* category shall define the rules and symbols that are used to format
 4917 XSI monetary numeric information. This information is available through the *localeconv()* function
 4918 and is used by the *strfmon()* function.

4919 XSI Some of the information is also available in an alternative form via the *nl_langinfo()* function
 4920 (see CRNCYSTR in *<langinfo.h>*).

4921 The following items are defined in this category of the locale. The item names are the keywords
 4922 recognized by the *localedef* utility when defining a locale. They are also similar to the member
 4923 names of the *lconv* structure defined in *<locale.h>*; see *<locale.h>* for the exact symbols in the
 4924 header. The *localeconv()* function returns {CHAR_MAX} for unspecified integer items and the
 4925 empty string (" ") for unspecified or size zero string items.

4926 In a locale definition file, the operands are strings, formatted as indicated by the grammar in
 4927 Section 7.4 (on page 176). For some keywords, the strings can contain only integers. Keywords
 4928 that are not provided, string values set to the empty string (" "), or integer keywords set to -1,
 4929 are used to indicate that the value is not available in the locale.

4930 **copy** Specify the name of an existing locale to be used as the definition of this
 4931 category. If this keyword is specified, no other keyword can be specified.

4932 **Note:** This is a *localedef* utility keyword, unavailable through
 4933 *localeconv()*.

4934 **int_curr_symbol** The international currency symbol. The operand is a four-character string,
 4935 with the first three characters containing the alphabetic international
 4936 currency symbol in accordance with those specified in the ISO 4217:1995
 4937 standard. The fourth character is the character used to separate the
 4938 international currency symbol from the monetary quantity.

4939 **currency_symbol** The string that shall be used as the local currency symbol.

4940 **mon_decimal_point** The operand is a string containing the symbol that shall be used as the
 4941 decimal delimiter (radix character) in monetary formatted quantities. In
 4942 contexts where standards (such as the ISO C standard) limit the
 4943 **mon_decimal_point** to a single byte, the result of specifying a multi-byte
 4944 operand is unspecified.

4945 **mon_thousands_sep** The operand is a string containing the symbol that shall be used as a
 4946 separator for groups of digits to the left of the decimal delimiter in
 4947 formatted monetary quantities. In contexts where standards limit the
 4948 **mon_thousands_sep** to a single byte, the result of specifying a multi-byte
 4949 operand is unspecified.

4950 **mon_grouping** Define the size of each group of digits in formatted monetary quantities.
 4951 The operand is a sequence of integers separated by semicolons. Each
 4952 integer specifies the number of digits in each group, with the initial
 4953 integer defining the size of the group immediately preceding the decimal
 4954 delimiter, and the following integers defining the preceding groups. If the
 4955 last integer is not -1, then the size of the previous group (if any) shall be
 4956 repeatedly used for the remainder of the digits. If the last integer is -1,
 4957 then no further grouping shall be performed.

4958 The following is an example of the interpretation of the **mon_grouping**
 4959 keyword. Assuming that the value to be formatted is 123456789 and the
 4960 **mon_thousands_sep** is ' ', then the following table shows the result.
 4961 The third column shows the equivalent string in the ISO C standard that
 4962 would be used by the *localeconv()* function to accommodate this
 4963 grouping.

mon_grouping	Formatted Value	ISO C String
3;-1	123456'789	"\3\177"
3	123'456'789	"\3"
3;2;-1	1234'56'789	"\3\2\177"
3;2	12'34'56'789	"\3\2"
-1	123456789	"\177"

4970 In these examples, the octal value of {CHAR_MAX} is 177.

4971 **positive_sign** A string that shall be used to indicate a non-negative-valued formatted
 4972 monetary quantity.

4973 **negative_sign** A string that shall be used to indicate a negative-valued formatted
 4974 monetary quantity.

4975 **int_frac_digits** An integer representing the number of fractional digits (those to the right
 4976 of the decimal delimiter) to be written in a formatted monetary quantity
 4977 using **int_curr_symbol**.

4978 **frac_digits** An integer representing the number of fractional digits (those to the right
 4979 of the decimal delimiter) to be written in a formatted monetary quantity
 4980 using **currency_symbol**.

4981 **p_cs_precedes** An integer set to 1 if the **currency_symbol** or **int_curr_symbol** precedes
 4982 the value for a monetary quantity with a non-negative value, and set to 0
 4983 if the symbol succeeds the value.

4984 **p_sep_by_space** An integer set to 0 if no space separates the **currency_symbol** or
 4985 **int_curr_symbol** from the value for a monetary quantity with a non-
 4986 negative value, set to 1 if a space separates the symbol from the value,
 4987 and set to 2 if a space separates the symbol and the sign string, if adjacent.

4988 **n_cs_precedes** An integer set to 1 if the **currency_symbol** or **int_curr_symbol** precedes
 4989 the value for a monetary quantity with a negative value, and set to 0 if the
 4990 symbol succeeds the value.

- 4991 **n_sep_by_space** An integer set to 0 if no space separates the **currency_symbol** or
- 4992 **int_curr_symbol** from the value for a monetary quantity with a negative
- 4993 value, set to 1 if a space separates the symbol from the value, and set to 2
- 4994 if a space separates the symbol and the sign string, if adjacent.

- 4995 **p_sign_posn** An integer set to a value indicating the positioning of the **positive_sign**
- 4996 for a monetary quantity with a non-negative value. The following integer
- 4997 values shall be recognized for both **p_sign_posn** and **n_sign_posn**:

- 4998 0 Parentheses enclose the quantity and the **currency_symbol** or
- 4999 **int_curr_symbol**.

- 5000 1 The sign string precedes the quantity and the **currency_symbol** or
- 5001 **int_curr_symbol**.

- 5002 2 The sign string succeeds the quantity and the **currency_symbol** or
- 5003 **int_curr_symbol**.

- 5004 3 The sign string precedes the **currency_symbol** or **int_curr_symbol**.
- 5005 4 The sign string succeeds the **currency_symbol** or **int_curr_symbol**.

- 5006 **n_sign_posn** An integer set to a value indicating the positioning of the **negative_sign**
- 5007 for a negative formatted monetary quantity.

5008 The following table shows the result of various combinations:

		p_sep_by_space			
		2	1	0	
5009	p_cs_precedes = 1	p_sign_posn = 0	(\$1.25)	(\$ 1.25)	(\$1.25)
5010		p_sign_posn = 1	+ \$1.25	+\$ 1.25	+\$1.25
5011		p_sign_posn = 2	\$1.25 +	\$ 1.25+	\$1.25+
5012		p_sign_posn = 3	+ \$1.25	+\$ 1.25	+\$1.25
5013		p_sign_posn = 4	\$ +1.25	\$+ 1.25	+\$1.25
5014	p_cs_precedes = 0	p_sign_posn = 0	(1.25 \$)	(1.25 \$)	(1.25\$)
5015		p_sign_posn = 1	+1.25 \$	+1.25 \$	+1.25\$
5016		p_sign_posn = 2	1.25\$ +	1.25 \$+	1.25\$+
5017		p_sign_posn = 3	1.25+ \$	1.25 +\$	1.25+\$
5018		p_sign_posn = 4	1.25\$ +	1.25 \$+	1.25\$+

5019 The monetary formatting definitions for the POSIX locale follow; the code listing depicting the

5020 *localedef* input, the table representing the same information with the addition of *localeconv()* and

5021 *nl_langinfo()* formats. All values are unspecified in the POSIX locale.

```

5022 XSI LC_MONETARY
5023 # This is the POSIX locale definition for
5024 # the LC_MONETARY category.
5025 #
5026 int_curr_symbol      " "
5027 currency_symbol     " "
5028 mon_decimal_point   " "
5029 mon_thousands_sep  " "
5030 mon_grouping        -1
5031 positive_sign       " "
5032 negative_sign       " "
5033 int_frac_digits     -1
5034 frac_digits         -1

```

```

5037 p_cs_precedes -1
5038 p_sep_by_space -1
5039 n_cs_precedes -1
5040 n_sep_by_space -1
5041 p_sign_posn -1
5042 n_sign_posn -1
5043 #
5044 END LC_MONETARY

```

	Item	POSIX locale Value	langinfo Constant	localeconv() Value	localedef Value
5045	currency_symbol	N/A	CRNCYSTR	" "	" "
5046	frac_digits	N/A	—	CHAR_MAX	-1
5047	int_curr_symbol	N/A	—	" "	" "
5048	int_frac_digits	N/A	—	CHAR_MAX	-1
5049	mon_decimal_point	N/A	—	" "	" "
5050	mon_thousands_sep	N/A	—	" "	" "
5051	mon_grouping	N/A	—	" "	" "
5052	positive_sign	N/A	—	" "	" "
5053	negative_sign	N/A	—	" "	" "
5054	p_cs_precedes	N/A	CRNCYSTR	CHAR_MAX	-1
5055	n_cs_precedes	N/A	CRNCYSTR	CHAR_MAX	-1
5056	p_sep_by_space	N/A	—	CHAR_MAX	-1
5057	n_sep_by_space	N/A	—	CHAR_MAX	-1
5058	p_sign_posn	N/A	—	CHAR_MAX	-1
5059	n_sign_posn	N/A	—	CHAR_MAX	-1

5062 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.
5063 The entry N/A indicates that the value is not available in the POSIX locale.

5064 7.3.4 LC_NUMERIC

5065 The *LC_NUMERIC* category shall define the rules and symbols that are used to format non-
5066 XSI monetary numeric information. This information is available through the *localeconv()* function.
5067 Some of the information is also available in an alternative form via the *nl_langinfo()* function.

5068 The following items are defined in this category of the locale. The item names are the keywords
5069 recognized by the *localedef* utility when defining a locale. They are also similar to the member
5070 names of the *lconv* structure defined in *<locale.h>*; see *<locale.h>* for the exact symbols in the
5071 header. The *localeconv()* function returns {CHAR_MAX} for unspecified integer items and the
5072 empty string (" ") for unspecified or size zero string items.

5073 In a locale definition file, the operands are strings, formatted as indicated by the grammar in
5074 Section 7.4 (on page 176). For some keywords, the strings can only contain integers. Keywords
5075 that are not provided, string values set to the empty string (" "), or integer keywords set to -1,
5076 shall be used to indicate that the value is not available in the locale. The following keywords
5077 shall be recognized:

5078 **copy** Specify the name of an existing locale to be used as the definition of this
5079 category. If this keyword is specified, no other keyword can be specified.

5080 **Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.

5081 **decimal_point** The operand is a string containing the symbol that shall be used as the
5082 decimal delimiter (radix character) in numeric, non-monetary formatted
5083 quantities. This keyword cannot be omitted and cannot be set to the empty

5084 string. In contexts where standards limit the **decimal_point** to a single byte,
 5085 the result of specifying a multi-byte operand shall be unspecified.

5086 **thousands_sep** The operand is a string containing the symbol that shall be used as a separator
 5087 for groups of digits to the left of the decimal delimiter in numeric, non-
 5088 monetary formatted monetary quantities. In contexts where standards limit
 5089 the **thousands_sep** to a single byte, the result of specifying a multi-byte
 5090 operand shall be unspecified.

5091 **grouping** Define the size of each group of digits in formatted non-monetary quantities.
 5092 The operand is a sequence of integers separated by semicolons. Each integer
 5093 specifies the number of digits in each group, with the initial integer defining
 5094 the size of the group immediately preceding the decimal delimiter, and the
 5095 following integers defining the preceding groups. If the last integer is not -1,
 5096 then the size of the previous group (if any) shall be repeatedly used for the
 5097 remainder of the digits. If the last integer is -1, then no further grouping shall
 5098 be performed.

5099 The non-monetary numeric formatting definitions for the POSIX locale follow; the code listing
 5100 depicting the *localedef* input, the table representing the same information with the addition of
 5101 XSI *localeconv()* values, and *nl_langinfo()* constants.

```
5102 LC_NUMERIC
5103 # This is the POSIX locale definition for
5104 # the LC_NUMERIC category.
5105 #
5106 decimal_point      "<period>"
5107 thousands_sep      " "
5108 grouping           -1
5109 #
5110 END LC_NUMERIC
```

Item	POSIX Locale Value	langinfo Constant	localeconv() Value	localedef Value
decimal_point	"."	RADIXCHAR	"."	."
thousands_sep	N/A	THOUSEP	" "	" "
grouping	N/A	—	" -1"	-1

5116 **Notes to Reviewers**

5117 *This section with side shading will not appear in the final copy. - Ed.*

5118 D1, XBD, ERN 112 asked why the grouping in the POSIX locale is -1, but the grouping line in the
 5119 POSIX Locale Value column of this table is N/A. The response from Gary Miller (via Mark
 5120 Brown) was that they are saying the same thing; the -1 means that there is no grouping, therefore
 5121 the grouping is not applicable.

5122 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conforming extension.
 5123 The entry N/A indicates that the value is not available in the POSIX locale.

5124 **7.3.5 LC_TIME**

5125 The *LC_TIME* category shall define the interpretation of the field descriptors supported by the
 5126 XSI *date* utility and affects the behavior of the *strptime()*, *wcsftime()*, *strptime()*, and *nl_langinfo()*
 5127 functions. Because the interfaces for C-language access and locale definition differ significantly,
 5128 they are described separately.

5129 **7.3.5.1 LC_TIME Locale Definition**

5130 For locale definition, the following mandatory keywords shall be recognized:

5131 **copy** Specify the name of an existing locale to be used as the definition of this
 5132 category. If this keyword is specified, no other keyword can be specified.

5133 **abday** Define the abbreviated weekday names, corresponding to the *%a* field
 5134 descriptor (conversion specification in the *strptime()*, *wcsftime()*, and *strptime()*
 5135 functions). The operand consists of seven semicolon-separated strings, each
 5136 surrounded by double-quotes. The first string shall be the abbreviated name of
 5137 the day corresponding to Sunday, the second the abbreviated name of the day
 5138 corresponding to Monday, and so on.

5139 **day** Define the full weekday names, corresponding to the *%A* field descriptor. The
 5140 operand consists of seven semicolon-separated strings, each surrounded by
 5141 double-quotes. The first string is the full name of the day corresponding to
 5142 Sunday, the second the full name of the day corresponding to Monday, and so
 5143 on.

5144 **abmon** Define the abbreviated month names, corresponding to the *%b* field
 5145 descriptor. The operand consists of twelve semicolon-separated strings, each
 5146 surrounded by double-quotes. The first string shall be the abbreviated name of
 5147 the first month of the year (January), the second the abbreviated name of the
 5148 second month, and so on.

5149 **mon** Define the full month names, corresponding to the *%B* field descriptor. The
 5150 operand consists of twelve semicolon-separated strings, each surrounded by
 5151 double-quotes. The first string shall be the full name of the first month of the
 5152 year (January), the second the full name of the second month, and so on.

5153 **d_t_fmt** Define the appropriate date and time representation, corresponding to the *%c*
 5154 field descriptor. The operand consists of a string, and can contain any
 5155 combination of characters and field descriptors. In addition, the string can
 5156 contain escape sequences defined in the table in Table 5-1 (on page 130) ('\\',
 5157 '\\a', '\\b', '\\f', '\\n', '\\r', '\\t', '\\v').

5158 **d_fmt** Define the appropriate date representation, corresponding to the *%x* field
 5159 descriptor. The operand consists of a string, and can contain any combination
 5160 of characters and field descriptors. In addition, the string can contain escape
 5161 sequences defined in the table in Table 5-1 (on page 130).

5162 **t_fmt** Define the appropriate time representation, corresponding to the *%X* field
 5163 descriptor. The operand consists of a string, and can contain any combination
 5164 of characters and field descriptors. In addition, the string can contain escape
 5165 sequences defined in the table in Table 5-1 (on page 130).

5166 **am_pm** Define the appropriate representation of the *ante meridiem* and *post meridiem*
 5167 strings, corresponding to the *%p* field descriptor. The operand consists of two
 5168 strings, separated by a semicolon, each surrounded by double-quotes. The
 5169 first string shall represent the *ante meridiem* designation, the last string the *post*

5170		<i>meridiem</i> designation.
5171	t_fmt_ampm	Define the appropriate time representation in the 12-hour clock format with am_pm , corresponding to the %r field descriptor. The operand consists of a string and can contain any combination of characters and field descriptors. If the string is empty, the 12-hour format is not supported in the locale.
5172		
5173		
5174		
5175	era	Define how years are counted and displayed for each era in a locale. The operand consists of semicolon-separated strings. Each string is an era description segment with the format:
5176		
5177		
5178		<i>direction:offset:start_date:end_date:era_name:era_format</i>
5179		according to the definitions below. There can be as many era description segments as are necessary to describe the different eras.
5180		
5181	Note:	The start of an era might not be the earliest point in the era—it may be the latest. For example, the Christian era BC starts on the day before January 1, AD 1, and increases with earlier time.
5182		
5183		
5184	<i>direction</i>	Either a '+' or a '-' character. The '+' character indicates that years closer to the <i>start_date</i> have lower numbers than those closer to the <i>end_date</i> . The '-' character indicates that years closer to the <i>start_date</i> have higher numbers than those closer to the <i>end_date</i> .
5185		
5186		
5187		
5188		
5189	<i>offset</i>	The number of the year closest to the <i>start_date</i> in the era, corresponding to the %Ey field descriptor.
5190		
5191	<i>start_date</i>	A date in the form yyyy/mm/dd, where yyyy, mm, and dd are the year, month, and day numbers respectively of the start of the era. Years prior to AD 1 are represented as negative numbers.
5192		
5193		
5194	<i>end_date</i>	The ending date of the era, in the same format as the <i>start_date</i> , or one of the two special values "-*" or "+*". The value "-*" indicates that the ending date is the beginning of time. The value "+*" indicates that the ending date is the end of time.
5195		
5196		
5197		
5198	<i>era_name</i>	A string representing the name of the era, corresponding to the %EC field descriptor.
5199		
5200	<i>era_format</i>	A string for formatting the year in the era, corresponding to the %EY field descriptor.
5201		
5202	era_d_fmt	Define the format of the date in alternative era notation, corresponding to the %Ex field descriptor.
5203		
5204	era_t_fmt	Define the locale's appropriate alternative time format, corresponding to the %EX field descriptor.
5205		
5206	era_d_t_fmt	Define the locale's appropriate alternative date and time format, corresponding to the %Ec field descriptor.
5207		
5208	alt_digits	Define alternative symbols for digits, corresponding to the %O field descriptor modifier. The operand consists of semicolon-separated strings, each surrounded by double-quotes. The first string is the alternative symbol corresponding with zero, the second string the symbol corresponding with one, and so on. Up to 100 alternative symbol strings can be specified. The %O modifier indicates that the string corresponding to the value specified via the field descriptor is used instead of the value.
5209		
5210		
5211		
5212		
5213		
5214		

5215 7.3.5.2 LC_TIME C-Language Access

5216 XSI The following information can be accessed. These correspond to constants defined in
5217 <langinfo.h> and used as arguments to the *nl_langinfo()* function.

5218 ABDAY_x The abbreviated weekday names (for example Sun), where *x* is a number from
5219 1 to 7.

5220 DAY_x The full weekday names (for example Sunday), where *x* is a number from 1 to
5221 7.

5222 ABMON_x The abbreviated month names (for example Jan), where *x* is a number from 1
5223 to 12.

5224 MON_x The full month names (for example January), where *x* is a number from 1 to
5225 12.

5226 D_T_FMT The appropriate date and time representation.

5227 D_FMT The appropriate date representation.

5228 T_FMT The appropriate time representation.

5229 AM_STR The appropriate ante-meridiem affix.

5230 PM_STR The appropriate post-meridiem affix.

5231 T_FMT_AMPM The appropriate time representation in the 12-hour clock format with
5232 AM_STR and PM_STR.

5233 ERA The era description segments, which describe how years are counted and
5234 displayed for each era in a locale. Each era description segment has the format:

5235 *direction:offset:start_date:end_date:era_name:era_format*

5236 according to the definitions below. There are as many era description
5237 segments as are necessary to describe the different eras. Era description
5238 segments are separated by semicolons.

5239 *direction* Either a '+' or a '-' character. The '+' character indicates that
5240 years closer to the *start_date* have lower numbers than those
5241 closer to the *end_date*. The '-' character indicates that years
5242 closer to the *start_date* have higher numbers than those closer to
5243 the *end_date*.

5244 *offset* The number of the year closest to the *start_date* in the era.

5245 *start_date* A date in the form *yyyy/mm/dd*, where *yyyy*, *mm*, and *dd* are the
5246 year, month, and day numbers respectively of the start of the
5247 era. Years prior to AD 1 are represented as negative numbers.

5248 *end_date* The ending date of the era, in the same format as the *start_date*,
5249 or one of the two special values "-*" or "+*". The value "-*"
5250 indicates that the ending date is the beginning of time. The value
5251 "+*" indicates that the ending date is the end of time.

5252 *era_name* The era, corresponding to the %EC conversion specification.

5253 *era_format* The format of the year in the era, corresponding to the %EY
5254 conversion specification.

5255 ERA_D_FMT The era date format.

- 5256 ERA_T_FMT The locale's appropriate alternative time format, corresponding to the %EX
- 5257 field descriptor.
- 5258 ERA_D_T_FMT The locale's appropriate alternative date and time format, corresponding to
- 5259 the %Ec field descriptor.
- 5260 ALT_DIGITS The alternative symbols for digits, corresponding to the %O conversion
- 5261 specification modifier. The value consists of semicolon-separated symbols.
- 5262 The first is the alternative symbol corresponding to zero, the second is the
- 5263 symbol corresponding to one, and so on. Up to 100 alternative symbols may
- 5264 be specified.

5265 The following table displays the correspondence between the items described above and the
 5266 conversion specifiers used by the *date* utility and the *strftime()*, *wcsftime()*, and *strptime()*
 5267 functions.

5268

5269

localedef Keyword	langinfo Constant	Conversion Specifier
5270 abday	ABDAY_x	%a
5271 day	DAY_x	%A
5272 abmon	ABMON_x	%b
5273 mon	MON	%B
5274 d_t_fmt	D_T_FMT	%c
5275 d_fmt	D_FMT	%x
5276 t_fmt	T_FMT	%X
5277 am_pm	AM_STR	%p
5278 am_pm	PM_STR	%p
5279 t_fmt_ampm	T_FMT_AMP	%r
5280 era	ERA	%EC, %Ey, %EY
5281 era_d_fmt	ERA_D_FMT	%Ex
5282 era_t_fmt	ERA_T_FMT	%EX
5283 era_d_t_fmt	ERA_D_T_FMT	%Ec
5284 alt_digits	ALT_DIGITS	%O

5285 In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.

5286 7.3.5.3 LC_TIME General Information

5287 The following is an example for Japan that supports the current plus last three Emperors and
 5288 reverts to Western style numbering for years prior to the Meiji era. The example also allows for
 5289 the custom of using a special name for the first year of an era instead of using 1. (The examples
 5290 substitute romaji where kanji should be used.)

```

5291     era_d_fmt "%EY%mgatsu%dnichi (%a)"
5292     era      "+:2:1990/01/01:+*:Heisei:%EC%Eyren";\
5293             "+:1:1989/01/08:1989/12/31:Heisei:%ECgannen";\
5294             "+:2:1927/01/01:1989/01/07:Shouwa:%EC%Eyren";\
5295             "+:1:1926/12/25:1926/12/31:Shouwa:%ECgannen";\
5296             "+:2:1913/01/01:1926/12/24:Taishou:%EC%Eyren";\
5297             "+:1:1912/07/30:1912/12/31:Taishou:%ECgannen";\
5298             "+:2:1869/01/01:1912/07/29:Meiji:%EC%Eyren";\
5299             "+:1:1868/09/08:1868/12/31:Meiji:%ECgannen";\
5300             "-:1868:1868/09/07:-*::%Ey"
```

5301 Assuming that the current date is September 21, 1991, a request to *date* or *strftime()* would yield
5302 the following results:

```
5303      %Ec - Heisei3nen9gatsu21nichi (Sat) 14:39:26
5304      %EC - Heisei
5305      %Ex - Heisei3nen9gatsu21nichi (Sat)
5306      %Ey - 3
5307      %EY - Heisei3nen
```

5308 Example era definitions for the Republic of China:

```
5309      era      "+:2:1913/01/01:+*:ChungHwaMingGuo:%EC%EyNen";\
5310             "+:1:1912/1/1:1912/12/31:ChungHwaMingGuo:%ECYuenNen";\
5311             "+:1:1911/12/31:-*:MingChien:%EC%EyNen"
```

5312 Example definitions for the Christian Era:

```
5313      era      "+:0:0000/01/01:+*:AD:%EC %Ey";\
5314             "+:1:-0001/12/31:-*:BC:%Ey %EC"
```

5315 The *LC_TIME* category definition of the POSIX locale follows; the code listing depicts the
5316 XSI *localedef* input; the table depicts the *langinfo* items defined in this category.

```
5317 LC_TIME
5318 # This is the POSIX locale definition for
5319 # the LC_TIME category.
5320 #
5321 # Abbreviated weekday names (%a)
5322 abday      "<S><u><n>"; "<M><o><n>"; "<T><u><e>"; "<W><e><d>";\
5323            "<T><h><u>"; "<F><r><i>"; "<S><a><t>"
5324 #
5325 # Full weekday names (%A)
5326 day        "<S><u><n><d><a><y>"; "<M><o><n><d><a><y>";\
5327            "<T><u><e><s><d><a><y>"; "<W><e><d><n><e><s><d><a><y>";\
5328            "<T><h><u><r><s><d><a><y>"; "<F><r><i><d><a><y>";\
5329            "<S><a><t><u><r><d><a><y>"
5330 #
5331 # Abbreviated month names (%b)
5332 abmon      "<J><a><n>"; "<F><e><b>"; "<M><a><r>";\
5333            "<A><p><r>"; "<M><a><y>"; "<J><u><n>";\
5334            "<J><u><l>"; "<A><u><g>"; "<S><e><p>";\
5335            "<O><c><t>"; "<N><o><v>"; "<D><e><c>"
5336 #
5337 # Full month names (%B)
5338 mon        "<J><a><n><u><a><r><y>"; "<F><e><b><r><u><a><r><y>";\
5339            "<M><a><r><c><h>"; "<A><p><r><i><l>";\
5340            "<M><a><y>"; "<J><u><n><e>";\
5341            "<J><u><l><y>"; "<A><u><g><u><s><t>";\
5342            "<S><e><p><t><e><m><b><e><r>"; "<O><c><t><o><b><e><r>";\
5343            "<N><o><v><e><m><b><e><r>"; "<D><e><c><e><m><b><e><r>"
5344 #
5345 # Equivalent of AM/PM (%p)      "AM"; "PM"
5346 am_pm      "<A><M>"; "<P><M>"
5347 #
5348 # Appropriate date and time representation (%c)
5349 #      "%a %b %e %H:%M:%S %Y"
```



```

5350     d_t_fmt      "<percent-sign><a><space><percent-sign><b>\
5351                  <space><percent-sign><e><space><percent-sign><H>\
5352                  <colon><percent-sign><M><colon><percent-sign><S>\
5353                  <space><percent-sign><Y>"
5354     #
5355     # Appropriate date representation (%x)    "%m/%d/%y"
5356     d_fmt        "<percent-sign><m><slash><percent-sign><d>\
5357                  <slash><percent-sign><y>"
5358     #
5359     # Appropriate time representation (%X)    "%H:%M:%S"
5360     t_fmt        "<percent-sign><H><colon><percent-sign><M>\
5361                  <colon><percent-sign><S>"
5362     #
5363     # Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
5364     t_fmt_ampm   "<percent-sign><I><colon><percent-sign><M><colon>\
5365                  <percent-sign><S><space><percent_sign><p>"
5366     #
5367     END LC_TIME

```

5368
5369
5370 XSI
5371
5372
5373
5374
5375
5376
5377
5378
5379
5380
5381
5382
5383
5384
5385
5386
5387
5388
5389
5390
5391

Item	POSIX Locale Value	Item	POSIX Locale Value
D_T_FMT	"%a %b %e %H:%M:%S %Y"	MON_3	"March"
D_FMT	"%m/%d/%y"	MON_4	"April"
T_FMT	"%H:%M:%S"	MON_5	"May"
AM_STR	"AM"	MON_6	"June"
PM_STR	"PM"	MON_7	"July"
T_FMT_AMP	"%I:%M:%S %p"	MON_8	"August"
DAY_1	"Sunday"	MON_9	"September"
DAY_2	"Monday"	MON_10	"October"
DAY_3	"Tuesday"	MON_11	"November"
DAY_4	"Wednesday"	MON_12	"December"
DAY_5	"Thursday"	ABMON_1	"Jan"
DAY_6	"Friday"	ABMON_2	"Feb"
DAY_7	"Saturday"	ABMON_3	"Mar"
ABDAY_1	"Sun"	ABMON_4	"Apr"
ABDAY_2	"Mon"	ABMON_5	"May"
ABDAY_3	"Tue"	ABMON_6	"Jun"
ABDAY_4	"Wed"	ABMON_7	"Jul"
ABDAY_5	"Thu"	ABMON_8	"Aug"
ABDAY_6	"Fri"	ABMON_9	"Sep"
ABDAY_7	"Sat"	ABMON_10	"Oct"
MON_1	"January"	ABMON_11	"Nov"
MON_2	"February"	ABMON_12	"Dec"

5392 **7.3.6 LC_MESSAGES**

5393 The *LC_MESSAGES* category shall define the format and values for affirmative and negative
5394 responses.

5395 XSI The message catalog used by the standard utilities and selected by the *catopen()* function shall be
5396 determined by the setting of *NLSPATH*; see Chapter 8 (on page 187). The *LC_MESSAGES*
5397 category can be specified as part of an *NLSPATH* substitution field.

5398 XSI The following keywords shall be recognized as part of the locale definition file. The
5399 *nl_langinfo()* function accepts uppercase versions of the first four keywords.

5400 **copy** Specify the name of an existing locale to be used as the definition of this category.
5401 If this keyword is specified, no other keyword can be specified.

5402 **yesexpr** The operand consists of an extended regular expression (see Section 9.4 (on page
5403 203)) that describes the acceptable affirmative response to a question expecting an
5404 affirmative or negative response.

5405 **noexpr** The operand consists of an extended regular expression that describes the
5406 acceptable negative response to a question expecting an affirmative or negative
5407 response.

5408 The format and values for affirmative and negative responses of the POSIX locale follow; the
5409 code listing depicting the *localedef* input, the table representing the same information with the
5410 XSI addition of *nl_langinfo()* constants.

```
5411 LC_MESSAGES
5412 # This is the POSIX locale definition for
5413 # the LC_MESSAGES category.
5414 #
```

```

5415     yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
5416     #
5417     noexpr  "<circumflex><left-square-bracket><n><N><right-square-bracket>"
5418     #
5419 XSI   yesstr  "yes"
5420     nostr   "no"
5421     END LC_MESSAGES

```

	localedef Keyword	langinfo Constant	POSIX Locale Value
5422	yesexpr	YESEXPR	"^[yY]"
5423	noexpr	NOEXPR	"^[nN]"
5424			
5425	XSI yesstr	YESSTR	"yes" (LEGACY)
5426	XSI nostr	NOSTR	"no" (LEGACY)

5427 **7.3.6.1** *LC_MESSAGES Application Usage*

5428 XSI The **yesstr** and **nostr** locale keywords and the YESSTR and NOSTR *langinfo* items were formerly
5429 used to match user affirmative and negative responses. In IEEE Std. 1003.1-200x, the **yesexpr**,
5430 **noexpr**, YESEXPR, and NOEXPR extended regular expressions have replaced them. However,
5431 they have been retained for backward compatibility to allow an application to include a sample
5432 desired response in a prompting message. They are marked LEGACY. Applications should use
5433 the general locale-based messaging facilities to issue such prompting messages.

5434 7.4 Locale Definition Grammar

5435 The grammar and lexical conventions in this section shall together describe the syntax for the
 5436 locale definition source. The general conventions for this style of grammar are described in the
 5437 Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 1.10, Grammar Conventions. The
 5438 grammar shall take precedence over the text in this chapter.

5439 7.4.1 Locale Lexical Conventions

5440 The lexical conventions for the locale definition grammar are described in this section.

5441 The following tokens shall be processed (in addition to those string constants shown in the
 5442 grammar):

5443	LOC_NAME	A string of characters representing the name of a locale.
5444	CHAR	Any single character.
5445	NUMBER	A decimal number, represented by one or more decimal digits.
5446	COLLSYMBOL	A symbolic name, enclosed between angle brackets. The string cannot duplicate any charmap symbol defined in the current charmap (if any), or a COLLELEMENT symbol.
5447		
5448		
5449	COLLELEMENT	A symbolic name, enclosed between angle brackets, which cannot duplicate either any charmap symbol or a COLLSYMBOL symbol.
5450		
5451	CHARCLASS	A string of alphanumeric characters from the portable character set, the first of which is not a digit, consisting of at least one and at most {CHARCLASS_NAME_MAX} bytes, and optionally surrounded by double-quotes.
5452		
5453		
5454		
5455	CHARSYMBOL	A symbolic name, enclosed between angle brackets, from the current charmap (if any).
5456		
5457	OCTAL_CHAR	One or more octal representations of the encoding of each byte in a single character. The octal representation consists of an escape character (normally a backslash) followed by two or more octal digits.
5458		
5459		
5460		
5461	HEX_CHAR	One or more hexadecimal representations of the encoding of each byte in a single character. The hexadecimal representation consists of an escape character followed by the constant x and two or more hexadecimal digits.
5462		
5463		
5464		
5465	DECIMAL_CHAR	One or more decimal representations of the encoding of each byte in a single character. The decimal representation consists of an escape character followed by a character 'd' and two or more decimal digits.
5466		
5467		
5468		
5469	ELLIPSIS	The string " . . . ".
5470	EXTENDED_REG_EXP	An extended regular expression as defined in the grammar in Section 9.5 (on page 206).
5471		
5472	EOL	The line termination character newline.

5473 **7.4.2 Locale Grammar**

5474 This section presents the grammar for the locale definition.

```

5475 %token          LOC_NAME
5476 %token          CHAR
5477 %token          NUMBER
5478 %token          COLLSYMBOL COLLELEMENT
5479 %token          CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
5480 %token          ELLIPSIS
5481 %token          EXTENDED_REG_EXP
5482 %token          EOL
5483 %start          locale_definition
5484 %%
5485 locale_definition : global_statements locale_categories
5486                  | locale_categories
5487                  ;
5488 global_statements : global_statements symbol_redefine
5489                  | symbol_redefine
5490                  ;
5491 symbol_redefine   : 'escape_char' CHAR EOL
5492                  | 'comment_char' CHAR EOL
5493                  ;
5494 locale_categories : locale_categories locale_category
5495                  | locale_category
5496                  ;
5497 locale_category  : lc_ctype | lc_collate | lc_messages
5498                  | lc_monetary | lc_numeric | lc_time
5499                  ;
5500 /* The following grammar rules are common to all categories */
5501 char_list        : char_list char_symbol
5502                  | char_symbol
5503                  ;
5504 char_symbol      : CHAR | CHARSYMBOL
5505                  | OCTAL_CHAR | HEX_CHAR | DECIMAL_CHAR
5506                  ;
5507 elem_list        : elem_list char_symbol
5508                  | elem_list COLLSYMBOL
5509                  | elem_list COLLELEMENT
5510                  | char_symbol
5511                  | COLLSYMBOL
5512                  | COLLELEMENT
5513                  ;
5514 symb_list        : symb_list COLLSYMBOL
5515                  | COLLSYMBOL
5516                  ;

```

```

5517     locale_name      : LOC_NAME
5518                       | ''' LOC_NAME '''
5519                       ;

5520     /* The following is the LC_CTYPE category grammar */

5521     lc_ctype           : ctype_hdr ctype_keywords      ctype_tlr
5522                       | ctype_hdr 'copy' locale_name EOL ctype_tlr
5523                       ;

5524     ctype_hdr          : 'LC_CTYPE' EOL
5525                       ;

5526     ctype_keywords     : ctype_keywords ctype_keyword
5527                       | ctype_keyword
5528                       ;

5529     ctype_keyword      : charclass_keyword charclass_list EOL
5530                       | charconv_keyword charconv_list EOL
5531                       | 'charclass' charclass_namelist EOL
5532                       ;

5533     charclass_namelist : charclass_namelist ';' CHARCLASS
5534                       | CHARCLASS
5535                       ;

5536     charclass_keyword  : 'upper' | 'lower' | 'alpha' | 'digit'
5537                       | 'punct' | 'xdigit' | 'space' | 'print'
5538                       | 'graph' | 'blank' | 'cntrl' | 'alnum'
5539                       | CHARCLASS
5540                       ;

5541     charclass_list     : charclass_list ';' char_symbol
5542                       | charclass_list ';' ELLIPSIS ';' char_symbol
5543                       | char_symbol
5544                       ;

5545     charconv_keyword   : 'toupper'
5546                       | 'tolower'
5547                       ;

5548     charconv_list      : charconv_list ';' charconv_entry
5549                       | charconv_entry
5550                       ;

5551     charconv_entry     : '(' char_symbol ',' char_symbol ')'
5552                       ;

5553     ctype_tlr          : 'END' 'LC_CTYPE' EOL
5554                       ;

5555     /* The following is the LC_COLLATE category grammar */

5556     lc_collate         : collate_hdr collate_keywords   collate_tlr
5557                       | collate_hdr 'copy' locale_name EOL collate_tlr
5558                       ;

5559     collate_hdr        : 'LC_COLLATE' EOL
5560                       ;

```

```

5561     collate_keywords      :           order_statements
5562                           | opt_statements order_statements
5563                           ;

5564     opt_statements         : opt_statements collating_symbols
5565                           | opt_statements collating_elements
5566                           | collating_symbols
5567                           | collating_elements
5568                           ;

5569     collating_symbols      : 'collating-symbol' COLLSYMBOL EOL
5570                           ;

5571     collating_elements     : 'collating-element' COLLELEMENT
5572                           | 'from' ''' elem_list ''' EOL
5573                           ;

5574     order_statements       : order_start collation_order order_end
5575                           ;

5576     order_start            : 'order_start' EOL
5577                           | 'order_start' order_opts EOL
5578                           ;

5579     order_opts              : order_opts ';' order_opt
5580                           | order_opt
5581                           ;

5582     order_opt              : order_opt ',' opt_word
5583                           | opt_word
5584                           ;

5585     opt_word                : 'forward' | 'backward' | 'position'
5586                           ;

5587     collation_order        : collation_order collation_entry
5588                           | collation_entry
5589                           ;

5590     collation_entry        : COLLSYMBOL EOL
5591                           | collation_element weight_list EOL
5592                           | collation_element                EOL
5593                           ;

5594     collation_element      : char_symbol
5595                           | COLLELEMENT
5596                           | ELLIPSIS
5597                           | 'UNDEFINED'
5598                           ;

5599     weight_list            : weight_list ';' weight_symbol
5600                           | weight_list ';'
5601                           | weight_symbol
5602                           ;

5603     weight_symbol          : /* empty */
5604                           | char_symbol
5605                           | COLLSYMBOL
5606                           | ''' elem_list '''

```

```

5607         | ''' symb_list '''
5608         | ELLIPSIS
5609         | 'IGNORE'
5610         ;
5611 order_end      : 'order_end' EOL
5612         ;
5613 collate_tlr    : 'END' 'LC_COLLATE' EOL
5614         ;
5615 /* The following is the LC_MESSAGES category grammar */
5616 lc_messages    : messages_hdr messages_keywords      messages_tlr
5617         | messages_hdr 'copy' locale_name EOL messages_tlr
5618         ;
5619 messages_hdr   : 'LC_MESSAGES' EOL
5620         ;
5621 messages_keywords : messages_keywords messages_keyword
5622         | messages_keyword
5623         ;
5624 messages_keyword : 'yesexpr' ''' EXTENDED_REG_EXP ''' EOL
5625         | 'noexpr' ''' EXTENDED_REG_EXP ''' EOL
5626         | 'yesstr' ''' char_list ''' EOL
5627         | 'nostr' ''' char_list ''' EOL
5628         ;
5629 messages_tlr   : 'END' 'LC_MESSAGES' EOL
5630         ;
5631 /* The following is the LC_MONETARY category grammar */
5632 lc_monetary    : monetary_hdr monetary_keywords      monetary_tlr
5633         | monetary_hdr 'copy' locale_name EOL monetary_tlr
5634         ;
5635 monetary_hdr   : 'LC_MONETARY' EOL
5636         ;
5637 monetary_keywords : monetary_keywords monetary_keyword
5638         | monetary_keyword
5639         ;
5640 monetary_keyword : mon_keyword_string mon_string EOL
5641         | mon_keyword_char NUMBER EOL
5642         | mon_keyword_char '-1' EOL
5643         | mon_keyword_grouping mon_group_list EOL
5644         ;
5645 mon_keyword_string : 'int_curr_symbol' | 'currency_symbol'
5646         | 'mon_decimal_point' | 'mon_thousands_sep'
5647         | 'positive_sign' | 'negative_sign'
5648         ;
5649 mon_string      : ''' char_list '''
5650         | ''''
5651         ;

```



```

5652     mon_keyword_char      : 'int_frac_digits' | 'frac_digits'
5653                           | 'p_cs_precedes' | 'p_sep_by_space'
5654                           | 'n_cs_precedes' | 'n_sep_by_space'
5655                           | 'p_sign_posn' | 'n_sign_posn'
5656                           ;

5657     mon_keyword_grouping  : 'mon_grouping'
5658                           ;

5659     mon_group_list        : NUMBER
5660                           | mon_group_list ';' NUMBER
5661                           ;

5662     monetary_tlr         : 'END' 'LC_MONETARY' EOL
5663                           ;

5664     /* The following is the LC_NUMERIC category grammar */
5665     lc_numeric            : numeric_hdr numeric_keywords      numeric_tlr
5666                           | numeric_hdr 'copy' locale_name EOL numeric_tlr
5667                           ;

5668     numeric_hdr           : 'LC_NUMERIC' EOL
5669                           ;

5670     numeric_keywords      : numeric_keywords numeric_keyword
5671                           | numeric_keyword
5672                           ;

5673     numeric_keyword       : num_keyword_string num_string EOL
5674                           | num_keyword_grouping num_group_list EOL
5675                           ;

5676     num_keyword_string    : 'decimal_point'
5677                           | 'thousands_sep'
5678                           ;

5679     num_string            : '"' char_list '"'
5680                           | '""'
5681                           ;

5682     num_keyword_grouping  : 'grouping'
5683                           ;

5684     num_group_list        : NUMBER
5685                           | num_group_list ';' NUMBER
5686                           ;

5687     numeric_tlr          : 'END' 'LC_NUMERIC' EOL
5688                           ;

5689     /* The following is the LC_TIME category grammar */
5690     lc_time               : time_hdr time_keywords          time_tlr
5691                           | time_hdr 'copy' locale_name EOL time_tlr
5692                           ;

5693     time_hdr              : 'LC_TIME' EOL
5694                           ;

```

```
5695     time_keywords      : time_keywords time_keyword
5696                          | time_keyword
5697                          ;
5698     time_keyword         : time_keyword_name time_list EOL
5699                          | time_keyword_fmt time_string EOL
5700                          | time_keyword_opt time_list EOL
5701                          ;
5702     time_keyword_name    : 'abday' | 'day' | 'abmon' | 'mon'
5703                          ;
5704     time_keyword_fmt     : 'd_t_fmt' | 'd_fmt' | 't_fmt'
5705                          | 'am_pm' | 't_fmt_ampm'
5706                          ;
5707     time_keyword_opt     : 'era' | 'era_d_fmt' | 'era_t_fmt'
5708                          | 'era_d_t_fmt' | 'alt_digits'
5709                          ;
5710     time_list            : time_list ';' time_string
5711                          | time_string
5712                          ;
5713     time_string          : '"' char_list '"'
5714                          ;
5715     time_tlr             : 'END' 'LC_TIME' EOL
5716                          ;
```

5717 **7.5 Locale Definition Example**

5718 The following is an example of a locale definition file that could be used as input to the *localedef*
 5719 utility. It assumes that the utility is executed with the *-f* option, naming a *charmap* file with (at
 5720 least) the following content:

```
5721     CHARMAP
5722     <space>      \x20
5723     <dollar>     \x24
5724     <A>         \101
5725     <a>         \141
5726     <A-acute>   \346
5727     <a-acute>   \365
5728     <A-grave>   \300
5729     <a-grave>   \366
5730     <b>         \142
5731     <C>         \103
5732     <c>         \143
5733     <c-cedilla> \347
5734     <d>         \x64
5735     <H>         \110
5736     <h>         \150
5737     <eszet>    \xb7
5738     <s>         \x73
5739     <z>         \x7a
5740     END CHARMAP
```

5741 It should not be taken as complete or to represent any actual locale, but only to illustrate the
 5742 syntax.

```
5743     #
5744     LC_CTYPE
5745     lower  <a>;<b>;<c>;<c-cedilla>;<d>;...;<z>
5746     upper  A;B;C;Ç;...;Z
5747     space  \x20;\x09;\x0a;\x0b;\x0c;\x0d
5748     blank  \040;\011
5749     toupper (<a>,<A>);(b,B);(c,C);(ç,Ç);(d,D);(z,Z)
5750     END LC_CTYPE
5751     #
5752     LC_COLLATE
5753     #
5754     # The following example of collation is based on
5755     # Canadian standard Z243.4.1-1998, "Canadian Alphanumeric
5756     # Ordering Standard For Character sets of CSA Z234.4 Standard".
5757     # (Other parts of this example locale definition file do not
5758     # purport to relate to Canada, or to any other real culture.)
5759     # The proposed standard defines a 4-weight collation, such that
5760     # in the first pass, characters are compared without regard to
5761     # case or accents; in second pass, backwards compare without
5762     # regard to case; in the third pass, forward compare without
5763     # regard to diacriticals. In the 3 first passes, non-alphabetic
5764     # characters are ignored; in the fourth pass, only special
5765     # characters are considered, such that "The string that has a
5766     # special character in the lowest position comes first. If two
```

```

5767      # strings have a special character in the same position, the
5768      # collation value of the special character determines ordering.
5769      #
5770      # Only a subset of the character set is used here; mostly to
5771      # illustrate the set-up.
5772      #
5773      collating-symbol <NULL>
5774      collating-symbol <LOW_VALUE>
5775      collating-symbol <LOWER-CASE>
5776      collating-symbol <SUBSCRIPT-LOWER>
5777      collating-symbol <SUPERSCRIPT-LOWER>
5778      collating-symbol <UPPER-CASE>
5779      collating-symbol <NO-ACCENT>
5780      collating-symbol <PECULIAR>
5781      collating-symbol <LIGATURE>
5782      collating-symbol <ACUTE>
5783      collating-symbol <GRAVE>
5784      # Further collating-symbols follow.
5785      #
5786      # Properly, the standard does not include any multi-character
5787      # collating elements; the one below is added for completeness.
5788      #
5789      collating_element <ch> from "<c><h>"
5790      collating_element <CH> from "<C><H>"
5791      collating_element <Ch> from "<C><h>"
5792      #
5793      order_start forward;backward;forward;forward,position
5794      #
5795      # Collating symbols are specified first in the sequence to allocate
5796      # basic collation values to them, lower than that of any character.
5797      <NULL>
5798      <LOW_VALUE>
5799      <LOWER-CASE>
5800      <SUBSCRIPT-LOWER>
5801      <SUPERSCRIPT-LOWER>
5802      <UPPER-CASE>
5803      <NO-ACCENT>
5804      <PECULIAR>
5805      <LIGATURE>
5806      <ACUTE>
5807      <GRAVE>
5808      <RING-ABOVE>
5809      <DIAERESIS>
5810      <TILDE>
5811      # Further collating symbols are given a basic collating value here.
5812      #
5813      # Here follow special characters.
5814      <space>          IGNORE;IGNORE;IGNORE;<space>
5815      # Other special characters follow here.
5816      #
5817      # Here follow the regular characters.
5818      <a>              <a>;<NO-ACCENT>;<LOWER-CASE>;IGNORE

```

```

5819      <A>          <a>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
5820      <a-acute>    <a>;<ACUTE>;<LOWER-CASE>;IGNORE
5821      <A-acute>    <a>;<ACUTE>;<UPPER-CASE>;IGNORE
5822      <a-grave>    <a>;<GRAVE>;<LOWER-CASE>;IGNORE
5823      <A-grave>   <a>;<GRAVE>;<UPPER-CASE>;IGNORE
5824      <ae>        "<a><e>";"<LIGATURE><LIGATURE>";\
5825                "<LOWER-CASE><LOWER-CASE>";IGNORE
5826      <AE>        "<a><e>";"<LIGATURE><LIGATURE>";\
5827                "<UPPER-CASE><UPPER-CASE>";IGNORE
5828      <b>          <b>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
5829      <B>          <b>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
5830      <c>          <c>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
5831      <C>          <c>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
5832      <ch>        <ch>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
5833      <Ch>        <ch>;<NO-ACCENT>;<PECULIAR>;IGNORE
5834      <CH>        <ch>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
5835      #
5836      # As an example, the strings "Bach" and "bach" could be encoded (for
5837      # compare purposes) as:
5838      # "Bach"    <b>;<a>;<ch>;<LOW_VALUE>;<NO_ACCENT>;<NO_ACCENT>;\
5839                <NO_ACCENT>;<LOW_VALUE>;<UPPER-CASE>;<LOWER-CASE>;\
5840                <LOWER-CASE>;<NULL>
5841      # "bach"   <b>;<a>;<ch>;<LOW_VALUE>;<NO_ACCENT>;<NO_ACCENT>;\
5842                <NO_ACCENT>;<LOW_VALUE>;<LOWER-CASE>;<LOWER-CASE>;\
5843                <LOWER-CASE>;<NULL>
5844      #
5845      # The two strings are equal in pass 1 and 2, but differ in pass 3.
5846      #
5847      # Further characters follow.
5848      #
5849      UNDEFINED    IGNORE;IGNORE;IGNORE;IGNORE
5850      #
5851      order_end
5852      #
5853      END LC_COLLATE
5854      #
5855      LC_MONETARY
5856      int_curr_symbol    "USD "
5857      currency_symbol    "$"
5858      mon_decimal_point  "."
5859      mon_grouping       3;0
5860      positive_sign      ""
5861      negative_sign      "- "
5862      p_cs_precedes      1
5863      n_sign_posn        0
5864      END LC_MONETARY
5865      #
5866      LC_NUMERIC
5867      copy "US_en.ASCII"
5868      END LC_NUMERIC
5869      #
5870      LC_TIME

```

```
5871      abday  "Sun";"Mon";"Tue";"Wed";"Thu";"Fri";"Sat"
5872      #
5873      day    "Sunday";"Monday";"Tuesday";"Wednesday";\
5874      "Thursday";"Friday";"Saturday"
5875      #
5876      abmon  "Jan";"Feb";"Mar";"Apr";"May";"Jun";\
5877      "Jul";"Aug";"Sep";"Oct";"Nov";"Dec"
5878      #
5879      mon    "January";"February";"March";"April";\
5880      "May";"June";"July";"August";"September";\
5881      "October";"November";"December"
5882      #
5883      d_t_fmt "%a %b %d %T %Z %Y\n"
5884      END LC_TIME
5885      #
5886      LC_MESSAGES
5887      yesexpr "^[yY][[:alpha:]]*"|(OK)"
5888      #
5889      noexpr  "^[nN][[:alpha:]]*"
5890      END LC_MESSAGES
```

Environment Variables

5891

5892 8.1 Environment Variable Definition

5893 Environment variables defined in this chapter affect the operation of multiple utilities, functions,
 5894 and applications. There are other environment variables that are of interest only to specific
 5895 utilities. Environment variables that apply to a single utility only are defined as part of the
 5896 utility description. See the ENVIRONMENT VARIABLES section of the utility descriptions in
 5897 the Shell and Utilities volume of IEEE Std. 1003.1-200x for information on environment variable
 5898 usage.

5899 The value of an environment variable is a string of characters. For a C-language program, an
 5900 array of strings called the environment is made available when a process begins. The array is
 5901 pointed to by the external variable *environ*, which is defined as:

```
5902     extern char **environ;
```

5903 These strings have the form *name=value*; *names* do not contain the character '='. For values to
 5904 be portable across systems conforming to IEEE Std. 1003.1-200x, the value shall be composed of
 5905 characters from the portable character set (except NUL and as indicated below). There is no
 5906 meaning associated with the order of strings in the environment. If more than one string in a
 5907 process' environment has the same *name*, the consequences are undefined.

5908 Environment variable names used by the utilities in the Shell and Utilities volume of
 5909 IEEE Std. 1003.1-200x shall consist solely of uppercase letters, digits, and the '_' (underscore)
 5910 from the characters defined in Table 6-1 (on page 133). Other characters may be permitted by an
 5911 implementation; applications shall tolerate the presence of such names. Uppercase and
 5912 lowercase letters retain their unique identities and are not folded together. The name space of
 5913 environment variable names containing lowercase letters is reserved for applications.
 5914 Applications can define any environment variables with names from this name space without
 5915 modifying the behavior of the standard utilities.

5916 The *values* that the environment variables may be assigned are not restricted except that they are
 5917 considered to end with a null byte and the total space used to store the environment and the
 5918 arguments to the process is limited to {ARG_MAX} bytes.

5919 Other *name=value* pairs may be placed in the environment by, for example, calling any of the
 5920 XSI *setenv()*, *unsetenv()*, or *putenv()* functions, manipulating the *environ* variable, or by using *envp*
 5921 arguments when creating a process; see *exec* in the System Interfaces volume of
 5922 IEEE Std. 1003.1-200x.

5923 It is unwise to conflict with certain variables that are frequently exported by widely used
 5924 command interpreters and applications:

5925
5926
5927
5928
5929
5930
5931
5932
5933
5934
5935
5936
5937
5938
5939
5940
5941
5942
5943
5944
5945

<i>ARFLAGS</i>	<i>IFS</i>	<i>MAILPATH</i>	<i>PS1</i>
<i>CC</i>	<i>LANG</i>	<i>MAILRC</i>	<i>PS2</i>
<i>CDPATH</i>	<i>LC_ALL</i>	<i>MAKEFLAGS</i>	<i>PS3</i>
<i>CFLAGS</i>	<i>LC_COLLATE</i>	<i>MAKESHELL</i>	<i>PS4</i>
<i>CHARSET</i>	<i>LC_CTYPE</i>	<i>MANPATH</i>	<i>PWD</i>
<i>COLUMNS</i>	<i>LC_MESSAGES</i>	<i>MBOX</i>	<i>RANDOM</i>
<i>DATMSK</i>	<i>LC_MONETARY</i>	<i>MORE</i>	<i>SECONDS</i>
<i>DEAD</i>	<i>LC_NUMERIC</i>	<i>MSGVERB</i>	<i>SHELL</i>
<i>EDITOR</i>	<i>LC_TIME</i>	<i>NLSPATH</i>	<i>TERM</i>
<i>ENV</i>	<i>LDFLAGS</i>	<i>NPROC</i>	<i>TERMCAP</i>
<i>EXINIT</i>	<i>LEX</i>	<i>OLDPWD</i>	<i>TERMINFO</i>
<i>FC</i>	<i>LFLAGS</i>	<i>OPTARG</i>	<i>TMPDIR</i>
<i>FCEDIT</i>	<i>LINENO</i>	<i>OPTERR</i>	<i>TZ</i>
<i>FFLAGS</i>	<i>LINES</i>	<i>OPTIND</i>	<i>USER</i>
<i>GET</i>	<i>LISTER</i>	<i>PAGER</i>	<i>VISUAL</i>
<i>GFLAGS</i>	<i>LOGNAME</i>	<i>PATH</i>	<i>YACC</i>
<i>HISTFILE</i>	<i>LPDEST</i>	<i>PPID</i>	<i>YFLAGS</i>
<i>HISTORY</i>	<i>MAIL</i>	<i>PRINTER</i>	
<i>HISTSIZE</i>	<i>MAILCHECK</i>	<i>PROCLANG</i>	
<i>HOME</i>	<i>MAILER</i>	<i>PROJECTDIR</i>	

5946
5947
5948
5949
5950
5951
5952
5953
5954
5955
5956

If the variables in the following two sections are present in the environment during the execution of an application or utility, they are given the meaning described below. Some are placed into the environment by the implementation at the time the user logs in; all can be added or changed by the user or any ancestor of the current process. The implementation adds or changes environment variables named in IEEE Std. 1003.1-200x only as specified in IEEE Std. 1003.1-200x. If they are defined in the application's environment, the utilities in the Shell and Utilities volume of IEEE Std. 1003.1-200x and the functions in the System Interfaces volume of IEEE Std. 1003.1-200x assume they have the specified meaning. Conforming applications shall not set these environment variables to have meanings other than as described. See *getenv()* and the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.13, Shell Execution Environment for methods of accessing these variables.

5957 **8.2 Internationalization Variables**

5958 This section describes environment variables that are relevant to the operation of
 5959 internationalized interfaces described in the System Interfaces volume of IEEE Std. 1003.1-200x
 5960 and the Shell and Utilities volume of IEEE Std. 1003.1-200x.

5961 Users may use the following environment variables to announce specific localization
 5962 requirements to applications. Applications shall retrieve this information using the *setlocale()*
 5963 function to initialize the correct behavior of the internationalized interfaces. The descriptions of
 5964 the internationalization environment variables describe the resulting behavior only when the
 5965 application locale is initialized in this way.

5966 **LANG** This variable shall determine the locale category for native language, local
 5967 customs, and coded character set in the absence of the *LC_ALL* and other *LC_**
 5968 (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*,
 5969 *LC_TIME*) environment variables. This can be used by applications to
 5970 determine the language to use for error messages and instructions, collating
 5971 sequences, date formats, and so on.

5972 **LC_ALL** This variable shall determine the values for all locale categories. The value of
 5973 the *LC_ALL* environment variable has precedence over any of the other
 5974 environment variables starting with *LC_*(*LC_COLLATE*, *LC_CTYPE*,
 5975 *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) and the *LANG*
 5976 environment variable.

5977 **LC_COLLATE** This variable shall determine the locale category for character collation. It
 5978 determines collation information for regular expressions and sorting,
 5979 including equivalence classes and multi-character collating elements, in
 5980 various utilities and the *strcoll()* and *strxfrm()* functions. Additional semantics
 5981 of this variable, if any, are implementation-defined.

5982 **LC_CTYPE** This variable shall determine the locale category for character handling
 5983 functions, such as *tolower()*, *toupper()*, and *isalpha()*. This environment
 5984 variable determines the interpretation of sequences of bytes of text data as
 5985 characters (for example, single as opposed to multi-byte characters), the
 5986 classification of characters (for example, alpha, digit, graph), and the behavior
 5987 of character classes. Additional semantics of this variable, if any, are
 5988 implementation-defined.

5989 **LC_MESSAGES** This variable shall determine the locale category for processing affirmative
 5990 and negative responses and the language and cultural conventions in which
 5991 XSI messages should be written. It also affects the behavior of the *catopen()*
 5992 function in determining the message catalog. Additional semantics of this
 5993 variable, if any, are implementation-defined. The language and cultural
 5994 conventions of diagnostic and informative messages whose format is
 5995 unspecified by IEEE Std. 1003.1-200x should be affected by the setting of
 5996 *LC_MESSAGES*.

5997 **LC_MONETARY** This variable shall determine the locale category for monetary-related numeric
 5998 formatting information. Additional semantics of this variable, if any, are
 5999 implementation-defined.

6000 **LC_NUMERIC** This variable shall determine the locale category for numeric formatting (for
 6001 example, thousands separator and radix character) information in various
 6002 utilities as well as the formatted I/O operations in *printf()* and *scanf()* and the
 6003 string conversion functions in *strtod()*. Additional semantics of this variable,
 6004 if any, are implementation-defined.

6005		LC_TIME	This variable shall determine the locale category for date and time formatting information. It affects the behavior of the time functions in <i>strftime()</i> . Additional semantics of this variable, if any, are implementation-defined.
6006			
6007			
6008	XSI	NLSPATH	This variable shall contain a sequence of templates that the <i>catopen()</i> function uses when attempting to locate message catalogs. Each template consists of an optional prefix, one or more substitution fields, a file name, and an optional suffix.
6009			
6010			
6011			
6012			For example:
6013			<code>NLSPATH="/system/nlslib/%N.cat"</code>
6014			defines that <i>catopen()</i> should look for all message catalogs in the directory <code>/system/nlslib</code> , where the catalog name should be constructed from the <i>name</i> parameter passed to <i>catopen()</i> (<i>%N</i>), with the suffix <code>.cat</code> .
6015			
6016			
6017			Substitution fields consist of a <code>'%'</code> symbol, followed by a single-letter keyword. The following keywords are currently defined:
6018			
6019			<code>%N</code> The value of the <i>name</i> parameter passed to <i>catopen()</i> .
6020			<code>%L</code> The value of the <i>LC_MESSAGES</i> category.
6021			<code>%l</code> The <i>language</i> element from the <i>LC_MESSAGES</i> category.
6022			<code>%t</code> The <i>territory</i> element from the <i>LC_MESSAGES</i> category.
6023			<code>%c</code> The <i>codeset</i> element from the <i>LC_MESSAGES</i> category.
6024			<code>%%</code> A single <code>'%'</code> character.
6025			An empty string is substituted if the specified value is not currently defined. The separators underscore (<code>'_'</code>) and period (<code>'.'</code>) are not included in <code>%t</code> and <code>%c</code> substitutions.
6026			
6027			
6028			Templates defined in <i>NLSPATH</i> are separated by colons (<code>':'</code>). A leading or two adjacent colons <code>": : "</code> is equivalent to specifying <code>%N</code> . For example:
6029			
6030			<code>NLSPATH=" : %N.cat : /nlslib/%L/%N.cat "</code>
6031			indicates to <i>catopen()</i> that it should look for the requested message catalog in <i>name</i> , <i>name.cat</i> , and <code>/nlslib/category/name.cat</code> , where <i>category</i> is the value of the <i>LC_MESSAGES</i> category of the current locale.
6032			
6033			
6034			Users should not set the <i>NLSPATH</i> variable unless they have a specific reason to override the default system path. Doing so causes undefined behavior in the standard utilities.
6035			
6036			
6037			The environment variables <i>LANG</i> , <i>LC_ALL</i> , <i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> ,
6038	XSI		<i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i> , and <i>NLSPATH</i> provide for the support of
6039			internationalized applications. The standard utilities shall make use of these environment
6040			variables as described in this section and the individual ENVIRONMENT VARIABLES sections
6041			for the utilities. If these variables specify locale categories that are not based upon the same
6042			underlying codeset, the results are unspecified.
6043			The values of locale categories shall be determined by a precedence order; the first condition met
6044			below determines the value:
6045			1. If the <i>LC_ALL</i> environment variable is defined and is not null, the value of <i>LC_ALL</i> shall be
6046			used.

- 6047 2. If the *LC_** environment variable (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*,
6048 *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) is defined and is not null, the value of the
6049 environment variable shall be used to initialize the category that corresponds to the
6050 environment variable.
- 6051 3. If the *LANG* environment variable is defined and is not null, the value of the *LANG*
6052 environment variable shall be used.
- 6053 4. If the *LANG* environment variable is not set or is set to the empty string, the
6054 implementation-defined default locale shall be used.

6055 If the locale value is "C" or "POSIX", the POSIX locale shall be used and the standard utilities
6056 behave in accordance with the rules in Section 7.2 (on page 144) for the associated category.

6057 If the locale value begins with a slash, it shall be interpreted as the path name of a file that was
6058 created in the output format used by the *localedf* utility; see OUTPUT FILES under *localedf*.
6059 Referencing such a path name results in that locale being used for the indicated category.

6060 XSI If the locale value has the form:

```
6061 language[_territory][.codeset]
```

6062 it refers to an implementation-provided locale, where settings of language, territory, and codeset
6063 are implementation-defined.

6064 *LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, and *LC_TIME* are
6065 defined to accept an additional field *@modifier*, which allows the user to select a specific instance
6066 of localization data within a single category (for example, for selecting the dictionary as opposed
6067 to the character ordering of data). The syntax for these environment variables is thus defined as:

```
6068 [language[_territory][.codeset][@modifier]]
```

6069 For example, if a user wanted to interact with the system in French, but required to sort German
6070 text files, *LANG* and *LC_COLLATE* could be defined as:

```
6071 LANG=Fr_FR  
6072 LC_COLLATE=De_DE
```

6073 This could be extended to select dictionary collation (say) by use of the *@modifier* field; for
6074 example:

```
6075 LC_COLLATE=De_DE@dict
```

6076

6077 An implementation may support other formats.

6078 If the locale value is not recognized by the implementation, the behavior is unspecified.

6079 At runtime, these values are bound to a program's locale by calling the *setlocale()* function.

6080 Additional criteria for determining a valid locale name are implementation-defined.

6081 **8.3 Other Environment Variables**

6082	<i>COLUMNS</i>	This variable shall represent a decimal integer >0 used to indicate the user's preferred width in column positions for the terminal screen or window; see Section 3.106 (on page 59). If this variable is unset or null, the implementation determines the number of columns, appropriate for the terminal or window, in an unspecified manner. When <i>COLUMNS</i> is set, any terminal-width information implied by <i>TERM</i> is overridden. Users and portable applications should not set <i>COLUMNS</i> unless they wish to override the system selection and produce output unrelated to the terminal characteristics.
6083		
6084		
6085		
6086		
6087		
6088		
6089		
6090		Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window.
6091		
6092		
6093	XSI <i>DATEMSK</i>	Indicates the path name of the template file used by <i>getdate()</i> .
6094	<i>HOME</i>	The system initializes this variable at the time of login to be a path name of the user's home directory. See < <i>pwd.h</i> >.
6095		
6096	<i>LINES</i>	This variable shall represent a decimal integer >0 used to indicate the user's preferred number of lines on a page or the vertical screen or window size in lines. A line in this case is a vertical measure large enough to hold the tallest character in the character set being displayed. If this variable is unset or null, the implementation determines the number of lines, appropriate for the terminal or window (size, terminal baud rate, and so on), in an unspecified manner. When <i>LINES</i> is set, any terminal-height information implied by <i>TERM</i> is overridden. Users and portable applications should not set <i>LINES</i> unless they wish to override the system selection and produce output unrelated to the terminal characteristics.
6097		
6098		
6099		
6100		
6101		
6102		
6103		
6104		
6105		
6106		Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window.
6107		
6108		
6109	<i>LOGNAME</i>	The system initializes this variable at the time of login to be the user's login name. See < <i>pwd.h</i> >. For a value of <i>LOGNAME</i> to be portable across implementations of IEEE Std. 1003.1-200x, the value should be composed of characters from the portable file name character set.
6110		
6111		
6112		
6113	XSI <i>MSGVERB</i>	Describes which message components shall be used in writing messages by <i>fmtmsg()</i> .
6114		
6115	<i>PATH</i>	This variable shall represent the sequence of path prefixes that certain functions and utilities apply in searching for an executable file known only by a file name. The prefixes are separated by a colon (':'). When a non-zero-length prefix is applied to this file name, a slash is inserted between the prefix and the file name. A zero-length prefix is a legacy feature that indicates the current working directory. It appears as two adjacent colons ("::"), as an initial colon preceding the rest of the list, or as a trailing colon following the rest of the list. A portable application shall use an actual path name (such as <i>.</i>) to represent the current working directory in <i>PATH</i> . The list is searched from beginning to end, applying the file name to each prefix, until an executable file with the specified name and appropriate execution permissions is found. If the path name being sought contains a slash, the search through the path prefixes is not performed. If the path name begins with a slash, the specified path is resolved (see Section 4.9 (on page 123)). If <i>PATH</i> is unset or is set to
6116		
6117		
6118		
6119		
6120		
6121		
6122		
6123		
6124		
6125		
6126		
6127		
6128		

6129		null, the path search is implementation-defined.
6130	<i>PWD</i>	This variable shall represent an absolute path name of the current working directory. It shall not contain any file name components of dot or dot-dot. The value is set by the <i>cd</i> utility.
6131		
6132		
6133	<i>SHELL</i>	This variable shall represent a path name of the user's preferred command language interpreter. If this interpreter does not conform to the Shell Command Language in the Shell and Utilities volume of IEEE Std. 1003.1-200x, Chapter 2, Shell Command Language, utilities may behave differently from those described in IEEE Std. 1003.1-200x.
6134		
6135		
6136		
6137		
6138	<i>TMPDIR</i>	This variable shall represent a path name of a directory made available for programs that need a place to create temporary files.
6139		
6140	<i>TERM</i>	This variable shall represent the terminal type for which output is to be prepared. This information is used by utilities and application programs wishing to exploit special capabilities specific to a terminal. The format and allowable values of this environment variable are unspecified.
6141		
6142		
6143		
6144	<i>TZ</i>	This variable shall represent timezone information. The contents of the environment variable named <i>TZ</i> shall be used by the <i>ctime()</i> , <i>localtime()</i> , <i>strftime()</i> , and <i>mktime()</i> functions, and by various utilities, to override the default timezone. The value of <i>TZ</i> has one of the two forms (spaces inserted for clarity):
6145		
6146		
6147		
6148		
6149		<i>:characters</i>
6150		or:
6151		<i>std offset dst offset, rule</i>
6152		If <i>TZ</i> is of the first format (that is, if the first character is a colon), the characters following the colon are handled in an implementation-defined manner.
6153		
6154		
6155		The expanded format (for all <i>TZs</i> whose value does not have a colon as the first character) is as follows:
6156		
6157		<i>stdoffset[dst[offset][,start[/time],end[/time]]]</i>
6158		Where:
6159	<i>std</i> and <i>dst</i>	Indicate no less than three, nor more than {TZNAME_MAX}, bytes that are the designation for the standard (<i>std</i>) or the alternative (<i>dst</i> —such as Daylight Savings Time) timezone. Only <i>std</i> is required; if <i>dst</i> is missing, then the alternative time does not apply in this locale.
6160		
6161		
6162		
6163		
6164		Each of these fields may occur in either of two formats quoted or unquoted:
6165		
6166		— In the quoted form, the first character shall be the less-than ('<') character and the last character shall be the greater-than ('>') character. All characters between these quoting characters shall be alphanumeric characters in the current locale, the plus-sign ('+') character, or the minus-sign ('-') character. The <i>std</i> and <i>dst</i> fields in this case do not include the quoting characters.
6167		
6168		
6169		
6170		
6171		
6172		

6173		— In the unquoted form, all characters in these fields shall be
6174		alphanumeric characters in the current locale.
6175		The interpretation of these fields is unspecified if either field is
6176		less than three bytes (except for the case when <i>dst</i> is missing),
6177		more than {TZNAME_MAX} bytes, or if they contain characters
6178		other than those specified.
6179	<i>offset</i>	Indicates the value added to the local time to arrive at
6180		Coordinated Universal Time. The <i>offset</i> has the form:
6181		<i>hh</i> [: <i>mm</i> [: <i>ss</i>]]
6182		The minutes (<i>mm</i>) and seconds (<i>ss</i>) are optional. The hour (<i>hh</i>)
6183		shall be required and may be a single digit. The <i>offset</i> following
6184		<i>std</i> shall be required. If no <i>offset</i> follows <i>dst</i> , the alternative time
6185		is assumed to be one hour ahead of standard time. One or more
6186		digits may be used; the value is always interpreted as a decimal
6187		number. The hour shall be between zero and 24, and the minutes
6188		(and seconds)—if present—between zero and 59. The result of
6189		using values outside of this range is unspecified. If preceded by
6190		a '-', the timezone shall be east of the Prime Meridian;
6191		otherwise, it shall be west (which may be indicated by an
6192		optional preceding '+').
6193	<i>rule</i>	Indicates when to change to and back from the alternative time.
6194		The <i>rule</i> has the form:
6195		<i>date</i> [/ <i>time</i>], <i>date</i> [/ <i>time</i>]
6196		where the first <i>date</i> describes when the change from standard to
6197		alternative time occurs and the second <i>date</i> describes when the
6198		change back happens. Each <i>time</i> field describes when, in current
6199		local time, the change to the other time is made.
6200		The format of <i>date</i> is one of the following:
6201	<i>Jn</i>	The Julian day <i>n</i> ($1 \leq n \leq 365$). Leap days shall not be
6202		counted. That is, in all years—including leap years—
6203		February 28 is day 59 and March 1 is day 60. It is
6204		impossible to refer explicitly to the occasional February
6205		29.
6206	<i>n</i>	The zero-based Julian day ($0 \leq n \leq 365$). Leap days shall
6207		be counted, and it is possible to refer to February 29.
6208	<i>Mm.n.d</i>	The <i>d</i> 'th day ($0 \leq d \leq 6$) of week <i>n</i> of month <i>m</i> of the
6209		year ($1 \leq n \leq 5$, $1 \leq m \leq 12$, where week 5 means "the
6210		last <i>d</i> day in month <i>m</i> " which may occur in either the
6211		fourth or the fifth week). Week 1 is the first week in
6212		which the <i>d</i> 'th day occurs. Day zero is Sunday.
6213		The <i>time</i> has the same format as <i>offset</i> except that no leading sign
6214		('-' or '+') is allowed. The default, if <i>time</i> is not given, shall be
6215		02:00:00.
6216		

Regular Expressions

6217

6218 *Regular Expressions* (REs) provide a mechanism to select specific strings from a set of character
6219 strings.

6220 Regular expressions are a context-independent syntax that can represent a wide variety of
6221 character sets and character set orderings, where these character sets are interpreted according
6222 to the current locale. While many regular expressions can be interpreted differently depending
6223 on the current locale, many features, such as character class expressions, provide for contextual
6224 invariance across locales.

6225 The Basic Regular Expression (BRE) notation and construction rules in Section 9.3 (on page 198)
6226 shall apply to most utilities supporting regular expressions. Some utilities, instead, support the
6227 Extended Regular Expressions (ERE) described in Section 9.4 (on page 203); any exceptions for
6228 both cases are noted in the descriptions of the specific utilities using regular expressions. Both
6229 BREs and EREs are supported by the Regular Expression Matching interface in the System
6230 Interfaces volume of IEEE Std. 1003.1-200x under *regcomp()*, *regexec()*, and related functions. |

6231 **9.1 Regular Expression Definitions**

6232 For the purposes of this section, the following definitions shall apply:

6233 **entire regular expression**6234 The concatenated set of one or more BREs or EREs that make up the pattern specified for
6235 string selection.6236 **matched**6237 A sequence of zero or more characters shall be said to be matched by a BRE or ERE when
6238 the characters in the sequence correspond to a sequence of characters defined by the
6239 pattern.6240 Matching shall be based on the bit pattern used for encoding the character, not on the
6241 graphic representation of the character. This means that if a character set contains two or
6242 more encodings for a graphic symbol, or if the strings searched contain text encoded in
6243 more than one codeset, no attempt is made to search for any other representation of the
6244 encoded symbol. If that is required, the user can specify equivalence classes containing all
6245 variations of the desired graphic symbol.6246 The search for a matching sequence starts at the beginning of a string and stops when the
6247 first sequence matching the expression is found, where *first* is defined to mean “begins
6248 earliest in the string”. If the pattern permits a variable number of matching characters and
6249 thus there is more than one such sequence starting at that point, the longest such sequence
6250 is matched. For example: the BRE "bb*" matches the second to fourth characters of *abbbc*,
6251 and the ERE *(wee | week)(knights | night)* matches all ten characters of *weeknights*.6252 Consistent with the whole match being the longest of the leftmost matches, each subpattern,
6253 from left to right, shall match the longest possible string. For this purpose, a null string shall
6254 be considered to be longer than no match at all. For example, matching the BRE
6255 "\(.*\).*" against "abcdef", the subexpression "(\1)" is "abcdef", and matching
6256 the BRE "\(a*\).*" against "bc", the subexpression "(\1)" is the null string.6257 When a multi-character collating element in a bracket expression (see Section 9.3.5 (on page
6258 199)) is involved, the longest sequence shall be measured in characters consumed from the
6259 string to be matched; that is, the collating element counts not as one element, but as the
6260 number of characters it matches.6261 **BRE (ERE) matching a single character**

6262 A BRE or ERE that shall match either a single character or a single collating element.

6263 Only a BRE or ERE of this type that includes a bracket expression (see Section 9.3.5 (on page
6264 199)) can match a collating element.6265 **BRE (ERE) matching multiple characters**

6266 A BRE or ERE that shall match a concatenation of single characters or collating elements.

6267 Such a BRE or ERE is made up from a BRE (ERE) matching a single character and BRE (ERE)
6268 special characters.6269 **invalid**6270 This section uses the term *invalid* for certain constructs or conditions. Invalid REs shall
6271 cause the utility or function using the RE to generate an error condition. When *invalid* is not
6272 used, violations of the specified syntax or semantics for REs produce undefined results: this
6273 may entail an error, enabling an extended syntax for that RE, or using the construct in error
6274 as literal characters to be matched. For example, the BRE construct "\{1,2,3\}" does not
6275 comply with the grammar. A portable application cannot rely on it producing an error nor
6276 matching the literal characters "\{1,2,3\}".

6277 9.2 Regular Expression General Requirements

6278 The requirements in this section shall apply to both basic and extended regular expressions.

6279 The use of regular expressions is generally associated with text processing. REs (BREs and EREs)
6280 operate on text strings; that is, zero or more characters followed by an end-of-string delimiter
6281 (typically NUL). Some utilities employing regular expressions limit the processing to lines; that
6282 is, zero or more characters followed by a <newline> character. In the regular expression
6283 processing described in IEEE Std. 1003.1-200x, the <newline> character is regarded as an
6284 ordinary character and both a period and a non-matching list can match one. The Shell and
6285 Utilities volume of IEEE Std. 1003.1-200x specifies within the individual descriptions of those
6286 standard utilities employing regular expressions whether they permit matching of <newline>
6287 characters; if not stated otherwise, the use of literal <newline> characters or any escape sequence
6288 equivalent produces undefined results. Those utilities (like *grep*) that do not allow <newline>
6289 characters to match are responsible for eliminating any <newline> character from strings before
6290 matching against the RE. The *regcomp()* function in the System Interfaces volume of
6291 IEEE Std. 1003.1-200x, however, can provide support for such processing without violating the
6292 rules of this section.

6293 The interfaces specified in IEEE Std. 1003.1-200x do not permit the inclusion of a NUL character
6294 in an RE or in the string to be matched. If during the operation of a standard utility a NUL is
6295 included in the text designated to be matched, that NUL may designate the end of the text string
6296 for the purposes of matching.

6297 When a standard utility or function that uses regular expressions specifies that pattern matching
6298 shall be performed without regard to the case (uppercase or lowercase) of either data or
6299 patterns, then when each character in the string is matched against the pattern, not only the
6300 character, but also its case counterpart (if any), shall be matched. This definition of case-
6301 insensitive processing is intended to allow matching of multi-character collating elements as
6302 well as characters. For example, as each character in the string is matched using both its cases,
6303 the RE "[[.Ch.]]" when matched against the string "char", is in reality matched against
6304 "ch", "Ch", "cH", and "CH".

6305 The implementation shall support any regular expression that does not exceed 256 bytes in
6306 length.

6307 **9.3 Basic Regular Expressions**6308 **9.3.1 BREs Matching a Single Character or Collating Element**

6309 A BRE ordinary character, a special character preceded by a backslash or a period, shall match a
 6310 single character. A bracket expression shall match a single character or a single collating
 6311 element.

6312 **9.3.2 BRE Ordinary Characters**

6313 An ordinary character is a BRE that matches itself: any character in the supported character set,
 6314 except for the BRE special characters listed in Section 9.3.3.

6315 The interpretation of an ordinary character preceded by a backslash ('**') is undefined, except
 6316 for:

- 6317 • The characters '**', '*(*', '*{*', and '*}*'
- 6318 • The digits 1 to 9 inclusive (see Section 9.3.6 (on page 201))
- 6319 • A character inside a bracket expression

6320 **9.3.3 BRE Special Characters**

6321 A *BRE special character* has special properties in certain contexts. Outside those contexts, or when
 6322 preceded by a backslash, such a character is a BRE that matches the special character itself. The
 6323 BRE special characters and the contexts in which they have their special meaning are as follows:

6324 . [** The period, left-bracket, and backslash shall be special except when used in a bracket
 6325 expression (see Section 9.3.5 (on page 199)). An expression containing a '*[*' that is not
 6326 preceded by a backslash and is not part of a bracket expression produces undefined
 6327 results.

6328 * The asterisk shall be special except when used:

- 6329 • In a bracket expression
- 6330 • As the first character of an entire BRE (after an initial '*^*', if any)
- 6331 • As the first character of a subexpression (after an initial '*^*', if any); see Section
 6332 9.3.6 (on page 201)

6333 ^ The circumflex shall be special when used as:

- 6334 • An anchor (see Section 9.3.8 (on page 202))
- 6335 • The first character of a bracket expression (see Section 9.3.5 (on page 199))

6336 \$ The dollar sign shall be special when used as an anchor.

6337 **9.3.4 Periods in BREs**

6338 A period ('*.*'), when used outside a bracket expression, is a BRE that shall match any character
 6339 in the supported character set except NUL.

6340 **9.3.5 RE Bracket Expression**

6341 A bracket expression (an expression enclosed in square brackets, "[]") is an RE that matches a
 6342 single collating element contained in the non-empty set of collating elements represented by the
 6343 bracket expression.

6344 The following rules and definitions apply to bracket expressions:

6345 1. A *bracket expression* is either a matching list expression or a non-matching list expression. It
 6346 consists of one or more expressions: collating elements, collating symbols, equivalence
 6347 classes, character classes, or range expressions. Portable applications shall not use range
 6348 expressions, even though all implementations shall support them. The right-bracket (']')
 6349 shall lose its special meaning and represents itself in a bracket expression if it occurs first in
 6350 the list (after an initial circumflex ('^'), if any). Otherwise, it shall terminate the bracket
 6351 expression, unless it appears in a collating symbol (such as "[.].]") or is the ending
 6352 right-bracket for a collating symbol, equivalence class, or character class. The special
 6353 characters '.', '*', '[', and '\' (period, asterisk, left-bracket, and backslash,
 6354 respectively) shall lose their special meaning within a bracket expression.

6355 The character sequences "[.", "[=", and "[:" (left-bracket followed by a period, equals-
 6356 sign, or colon) shall be special inside a bracket expression and are used to delimit collating
 6357 symbols, equivalence class expressions, and character class expressions. These symbols
 6358 shall be followed by a valid expression and the matching terminating sequence ".]",
 6359 " =]", or " :]", as described in the following items.

6360 2. A *matching list* expression specifies a list that shall match any one of the expressions
 6361 represented in the list. The first character in the list shall not be the circumflex; for
 6362 example, "[abc]" is an RE that matches any of the characters 'a', 'b', or 'c'.

6363 3. A *non-matching list* expression begins with a circumflex ('^'), and specifies a list that shall
 6364 match any character or collating element except for the expressions represented in the list
 6365 after the leading circumflex. For example, "[^abc]" is an RE that matches any character
 6366 or collating element except the characters 'a', 'b', or 'c'. The circumflex shall have this
 6367 special meaning only when it occurs first in the list, immediately following the left-bracket.

6368 4. A *collating symbol* is a collating element enclosed within bracket-period ("[.]" and ".]")
 6369 delimiters. Collating elements are defined as described in Section 7.3.2.4 (on page 158).
 6370 Portable applications shall represent multi-character collating elements as collating
 6371 symbols when it is necessary to distinguish them from a list of the individual characters
 6372 that make up the multi-character collating element. For example, if the string "ch" is a
 6373 collating element in the current collation sequence with the associated collating symbol
 6374 <ch>, the expression "[[.ch.]]" shall be treated as an RE matching the character
 6375 sequence 'ch', while "[ch]" shall be treated as an RE matching 'c' or 'h'. Collating
 6376 symbols are recognized only inside bracket expressions. This implies that the RE
 6377 "[[.ch.]]*c" shall match the first to fifth character in the string "chchch". If the string
 6378 is not a collating element in the current collating sequence definition, or if the collating
 6379 element has no characters associated with it (for example, see the symbol <HIGH> in the
 6380 example collation definition shown in Section 7.3.2.2 (on page 157)), the symbol shall be
 6381 treated as an invalid expression.

6382 5. An *equivalence class expression* shall represent the set of collating elements belonging to an
 6383 equivalence class, as described in Section 7.3.2.4 (on page 158). Only primary equivalence
 6384 classes shall be recognized. The class shall be expressed by enclosing any one of the
 6385 collating elements in the equivalence class within bracket-equal ("[=" and "=]")
 6386 delimiters. For example, if 'a', 'â', and 'ã' belong to the same equivalence class, then
 6387 "[[=a=]b]", "[[=â=]b]", and "[[=ã=]b]" are each equivalent to "[aâãb]". If the

6388 collating element does not belong to an equivalence class, the equivalence class expression
6389 shall be treated as a *collating symbol*.

6390 6. A *character class expression* shall represent the set of characters belonging to a character
6391 class, as defined in the *LC_CTYPE* category in the current locale. All character classes
6392 specified in the current locale shall be recognized. A character class expression is expressed
6393 as a character class name enclosed within bracket-colon ("[:]" and "[:]") delimiters.

6394 The following character class expressions shall be supported in all locales:

```
6395      [:alnum:]      [:cntrl:]      [:lower:]      [:space:]
6396      [:alpha:]     [:digit:]     [:print:]     [:upper:]
6397      [:blank:]     [:graph:]     [:punct:]     [:xdigit:]
```

6398 XSI In addition, character class expressions of the form:

```
6399      [:name:]
```

6400 are recognized in those locales where the *name* keyword has been given a **charclass**
6401 definition in the *LC_CTYPE* category.

6402 7. A range expression represents the set of collating elements that fall between two elements
6403 in the collating element order of the current locale, inclusive. A range expression shall be
6404 expressed as the starting point and the ending point separated by a hyphen ('-').

6405 Range expressions shall not be used in portable applications because their behavior is
6406 dependent on the collating sequence.

6407 In the following, all examples assume the collation sequence specified for the POSIX locale,
6408 unless another collation sequence is specifically defined.

6409 The starting range point and the ending range point shall be a collating element or
6410 collating symbol. An equivalence class expression used as a starting or ending point of a
6411 range expression produces unspecified results. An equivalence class can be used portably
6412 within a bracket expression, but only outside the range. For example, the unspecified
6413 expression "[[=e=]-f]" should be given as "[[=e=]e-f]". The ending range point
6414 shall collate equal to or higher than the starting range point; otherwise, the expression is
6415 treated as invalid. The order used is the order in which the collating elements are specified
6416 in the current collation definition. One-to-many mappings (see the description of
6417 *LC_COLLATE* in Section 7.3.2 (on page 155)) are not performed. For example, assuming
6418 that the character eszet ('ß') is placed in the collation sequence after 'r' and 's', but
6419 before 't' and that it maps to the sequence "ss" for collation purposes, then the
6420 expression "[r-s]" matches only 'r' and 's', but the expression "[s-t]" matches
6421 's', 'ß', or 't'.

6422 The interpretation of range expressions where the ending range point is also the starting
6423 range point of a subsequent range expression (for example, "[a-m-o]") is undefined.

6424 The hyphen character shall be treated as itself if it occurs first (after an initial '^', if any)
6425 or last in the list, or as an ending range point in a range expression. As examples, the
6426 expressions "[ac]" and "[ac-]" are equivalent and match any of the characters 'a',
6427 'c', or '-'; "[^ac]" and "[^ac-]" are equivalent and match any characters except
6428 'a', 'c', or '-'; the expression "[%-]" matches any of the characters between '%' and
6429 '-' inclusive; the expression "[--@]" matches any of the characters between '-' and
6430 '@' inclusive; and the expression "[a--@]" is invalid, because the letter 'a' follows the
6431 symbol '-' in the POSIX locale. To use a hyphen as the starting range point, it shall either
6432 come first in the bracket expression or be specified as a collating symbol; for example,
6433 "[] [.-.] -0]", which matches either a right bracket or any character or collating element

6434 that collates between hyphen and 0, inclusive.

6435 If a bracket expression specifies both '-' and ']', the ']' shall be placed first (after the
6436 '^', if any) and the '-' last within the bracket expression.

6437 9.3.6 BREs Matching Multiple Characters

6438 The following rules can be used to construct BREs matching multiple characters from BREs
6439 matching a single character:

6440 1. The concatenation of BREs shall match the concatenation of the strings matched by each
6441 component of the BRE.

6442 2. A *subexpression* can be defined within a BRE by enclosing it between the character pairs
6443 "\" and "\". Such a subexpression shall match whatever it would have matched
6444 without the "\" and "\" , except that anchoring within subexpressions is optional
6445 behavior; see Section 9.3.8 (on page 202). Subexpressions can be arbitrarily nested.

6446 3. The *back-reference* expression '\n' shall match the same (possibly empty) string of
6447 characters as was matched by a subexpression enclosed between "\" and "\"
6448 preceding the '\n'. The character 'n' shall be a digit from 1 through 9, specifying the
6449 *n*th subexpression (the one that begins with the *n*th "\" from the beginning of the
6450 pattern and ends with the corresponding paired "\"). The expression is invalid if less
6451 than *n* subexpressions precede the '\n'. For example, the expression "\(.*)\1\$" |
6452 matches a line consisting of two adjacent appearances of the same string, and the
6453 expression "\(a\)*\1" fails to match 'a'. When the referenced subexpression matched
6454 more than one string, the back-referenced expression shall refer to the last matched string.
6455 If the subexpression referenced by the back-reference matches more than one string
6456 because of an asterisk ('*') or an interval expression (see item (5)), the back-reference
6457 shall match the last (rightmost) of these strings.

6458 4. When a BRE matching a single character, a subexpression, or a back-reference is followed
6459 by the special character asterisk ('*'), together with that asterisk it shall match what zero
6460 or more consecutive occurrences of the BRE would match. For example, "[ab]*" and
6461 "[ab][ab]" are equivalent when matching the string "ab".

6462 5. When a BRE matching a single character, a subexpression, or a back-reference is followed
6463 by an *interval expression* of the format "{m}", "{m,}", or "{m,n}", together with
6464 that interval expression it shall match what repeated consecutive occurrences of the BRE
6465 would match. The values of *m* and *n* are decimal integers in the range 0
6466 $\leq m \leq n \leq \{RE_DUP_MAX\}$, where *m* specifies the exact or minimum number of occurrences
6467 and *n* specifies the maximum number of occurrences. The expression "{m}" shall match
6468 exactly *m* occurrences of the preceding BRE, "{m,}" shall match at least *m* occurrences,
6469 and "{m,n}" shall match any number of occurrences between *m* and *n*, inclusive.

6470 For example, in the string "abababcccccd" the BRE "c{3}" is matched by
6471 characters '7' to '9', the BRE "\(ab)\{4,}" is not matched at all, and the BRE
6472 "c{1,3}d" is matched by characters ten to thirteen.

6473 The behavior of multiple adjacent duplication symbols ('*' and intervals) produces undefined
6474 results.

6475 A subexpression repeated by an asterisk ('*') or an interval expression shall not match a null
6476 expression unless this is the only match for the repetition or it is necessary to satisfy the exact or
6477 minimum number of occurrences for the interval expression.

6478 **9.3.7 BRE Precedence**

6479 The order of precedence shall be as shown in the following table:

BRE Precedence (from high to low)	
6480	
6481	Collation-related bracket symbols [=] [: :] [. .]
6482	Escaped characters \<special character>
6483	Bracket expression []
6484	Subexpressions/back-references \ (\) \ n
6485	Single-character-BRE duplication * \ { m , n \ }
6486	Concatenation
6487	Anchoring ^ \$

6488 **9.3.8 BRE Expression Anchoring**

6489 A BRE can be limited to matching strings that begin or end a line; this is called *anchoring*. The
 6490 circumflex and dollar sign special characters shall be considered BRE anchors in the following
 6491 contexts:

- 6492 1. A circumflex (' ^ ') shall be an anchor when used as the first character of an entire BRE.
 6493 The implementation may treat the circumflex as an anchor when used as the first character
 6494 of a subexpression. The circumflex shall anchor the expression (or optionally
 6495 subexpression) to the beginning of a string; only sequences starting at the first character of
 6496 a string shall be matched by the BRE. For example, the BRE " ^ ab " matches " ab " in the
 6497 string " abcdef ", but fails to match in the string " cdefab ". The BRE " \ (^ ab \) " may
 6498 match the former string. A portable BRE shall escape a leading circumflex in a
 6499 subexpression to match a literal circumflex.
- 6500 2. A dollar sign (' \$ ') shall be an anchor when used as the last character of an entire BRE.
 6501 The implementation may treat a dollar sign as an anchor when used as the last character of
 6502 a subexpression. The dollar sign shall anchor the expression (or optionally subexpression)
 6503 to the end of the string being matched; the dollar sign can be said to match the end-of-
 6504 string following the last character.
- 6505 3. A BRE anchored by both ' ^ ' and ' \$ ' shall match only an entire string. For example, the
 6506 BRE " ^ abcdef \$ " matches strings consisting only of " abcdef ".

6507 **9.4 Extended Regular Expressions**

6508 The *extended regular expression* (ERE) notation and construction rules shall apply to utilities
 6509 defined as using extended regular expressions; any exceptions to the following rules are noted in
 6510 the descriptions of the specific utilities using EREs.

6511 **9.4.1 EREs Matching a Single Character or Collating Element**

6512 An ERE ordinary character, a special character preceded by a backslash, or a period shall match
 6513 a single character. A bracket expression shall match a single character or a single collating
 6514 element. An *ERE matching a single character* enclosed in parentheses shall match the same as the
 6515 ERE without parentheses would have matched.

6516 **9.4.2 ERE Ordinary Characters**

6517 An *ordinary character* is an ERE that matches itself. An ordinary character is any character in the
 6518 supported character set, except for the ERE special characters listed in Section 9.4.3. The
 6519 interpretation of an ordinary character preceded by a backslash ('**') is undefined.

6520 **9.4.3 ERE Special Characters**

6521 An *ERE special character* has special properties in certain contexts. Outside those contexts, or
 6522 when preceded by a backslash, such a character shall be an ERE that matches the special
 6523 character itself. The extended regular expression special characters and the contexts in which
 6524 they shall have their special meaning are as follows:

6525 . [** (The period, left-bracket, backslash, and left-parenthesis shall be special except when
 6526 used in a bracket expression (see Section 9.3.5 (on page 199)). Outside a bracket
 6527 expression, a left-parenthesis immediately followed by a right-parenthesis produces
 6528 undefined results.

6529) The right-parenthesis shall be special when matched with a preceding left-parenthesis,
 6530 both outside a bracket expression.

6531 * + ? { The asterisk, plus-sign, question-mark, and left-brace shall be special except when used
 6532 in a bracket expression (see Section 9.3.5 (on page 199)). Any of the following uses
 6533 produce undefined results:

- 6534 • If these characters appear first in an ERE, or immediately following a vertical-line,
 6535 circumflex, or left-parenthesis

- 6536 • If a left-brace is not part of a valid interval expression (see Section 9.4.6 (on page
 6537 204))

6538 | The vertical-line is special except when used in a bracket expression (see Section 9.3.5
 6539 (on page 199)). A vertical-line appearing first or last in an ERE, or immediately
 6540 following a vertical-line or a left-parenthesis, or immediately preceding a right-
 6541 parenthesis, produces undefined results.

6542 ^ The circumflex shall be special when used as:

- 6543 • An anchor (see Section 9.4.9 (on page 205))

- 6544 • The first character of a bracket expression (see Section 9.3.5 (on page 199))

6545 \$ The dollar sign shall be special when used as an anchor.

6546 **9.4.4 Periods in EREs**

6547 A period ('.'), when used outside a bracket expression, is an ERE that shall match any
6548 character in the supported character set except NUL.

6549 **9.4.5 ERE Bracket Expression**

6550 The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see Section
6551 9.3.5 (on page 199).

6552 **9.4.6 EREs Matching Multiple Characters**

6553 The following rules shall be used to construct EREs matching multiple characters from EREs
6554 matching a single character:

- 6555 1. A *concatenation of EREs* shall match the concatenation of the character sequences matched
6556 by each component of the ERE. A concatenation of EREs enclosed in parentheses shall
6557 match whatever the concatenation without the parentheses matches. For example, both the
6558 ERE "cd" and the ERE "(cd)" are matched by the third and fourth character of the string
6559 "abcdefabcdef".
- 6560 2. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6561 the special character plus-sign ('+'), together with that plus-sign it shall match what one
6562 or more consecutive occurrences of the ERE would match. For example, the ERE
6563 "b+(bc)" matches the fourth to seventh characters in the string "acabbbbcde". And,
6564 "[ab]+" and "[ab][ab]*" are equivalent.
- 6565 3. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6566 the special character asterisk ('*'), together with that asterisk it shall match what zero or
6567 more consecutive occurrences of the ERE would match. For example, the ERE "b*c"
6568 matches the first character in the string "cabbbbcde", and the ERE "b*cd" matches the
6569 third to seventh characters in the string "cabbbbcdebbbbbbbcdbbc". And, "[ab]*" and
6570 "[ab][ab]" are equivalent when matching the string "ab".
- 6571 4. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6572 the special character question-mark ('?'), together with that question-mark it shall match
6573 what zero or one consecutive occurrences of the ERE would match. For example, the ERE
6574 "b?c" matches the second character in the string "acabbbbcde".
- 6575 5. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6576 an *interval expression* of the format "{m}", "{m,}", or "{m,n}", together with that
6577 interval expression it shall match what repeated consecutive occurrences of the ERE would
6578 match. The values of *m* and *n* are decimal integers in the range $0 \leq m \leq n \leq \text{RE_DUP_MAX}$,
6579 where *m* specifies the exact or minimum number of occurrences and *n* specifies the
6580 maximum number of occurrences. The expression "{m}" matches exactly *m* occurrences
6581 of the preceding ERE, "{m,}" matches at least *m* occurrences, and "{m,n}" matches any
6582 number of occurrences between *m* and *n*, inclusive.

6583 For example, in the string "abababcccccd" the ERE "c{3}" is matched by characters
6584 '7' to '9' and the ERE "(ab){2,}" is matched by characters one to six.

6585 The behavior of multiple adjacent duplication symbols ('+', '*', '?', and intervals) produces
6586 undefined results.

6587 An ERE matching a single character repeated by an '*', '?', or an interval expression shall not
6588 match a null expression unless this is the only match for the repetition or it is necessary to satisfy
6589 the exact or minimum number of occurrences for the interval expression.

6590 **9.4.7 ERE Alternation**

6591 Two EREs separated by the special character vertical-line ('|') shall match a string that is
 6592 matched by either. For example, the ERE "a((bc)|d)" matches the string "abc" and the string
 6593 "ad". Single characters, or expressions matching single characters, separated by the vertical bar
 6594 and enclosed in parentheses, shall be treated as an ERE matching a single character.

6595 **9.4.8 ERE Precedence**

6596 The order of precedence shall be as shown in the following table:

ERE Precedence (from high to low)	
6597 Collation-related bracket symbols	[==] [::] [..]
6598 Escaped characters	\<special character>
6600 Bracket expression	[]
6601 Grouping	()
6602 Single-character-ERE duplication	* + ? {m,n}
6603 Concatenation	
6604 Anchoring	^ \$
6605 Alternation	

6606 For example, the ERE "abba|cde" matches either the string "abba" or the string "cde"
 6607 (rather than the string "abbade" or "abbcde", because concatenation has a higher order of
 6608 precedence than alternation).

6609 **9.4.9 ERE Expression Anchoring**

6610 An ERE can be limited to matching strings that begin or end a line; this is called *anchoring*. The
 6611 circumflex and dollar sign special characters shall be considered ERE anchors when used
 6612 anywhere outside a bracket expression. This shall have the following effects:

- 6613 1. A circumflex ('^') outside a bracket expression shall anchor the expression or
 6614 subexpression it begins to the beginning of a string; such an expression or subexpression
 6615 can match only a sequence starting at the first character of a string. For example, the EREs
 6616 "^ab" and "(^ab)" match "ab" in the string "abcdef", but fail to match in the string
 6617 "cdefab", and the ERE "a^b" is valid, but can never match because the 'a' prevents the
 6618 expression "^b" from matching starting at the first character.
- 6619 2. A dollar sign ('\$') outside a bracket expression shall anchor the expression or
 6620 subexpression it ends to the end of a string; such an expression or subexpression can
 6621 match only a sequence ending at the last character of a string. For example, the EREs
 6622 "ef\$" and "(ef\$)" match "ef" in the string "abcdef", but fail to match in the string
 6623 "cdefab", and the ERE "e\$f" is valid, but can never match because the 'f' prevents the
 6624 expression "e\$" from matching ending at the last character.

6625 **9.5 Regular Expression Grammar**

6626 Grammars describing the syntax of both basic and extended regular expressions are presented in
 6627 this section. The grammar takes precedence over the text. See the Shell and Utilities volume of
 6628 IEEE Std. 1003.1-200x, Section 1.10, Grammar Conventions.

6629 **9.5.1 BRE/ERE Grammar Lexical Conventions**

6630 The lexical conventions for regular expressions are as described in this section.

6631 Except as noted, the longest possible token or delimiter beginning at a given point is recognized.

6632 The following tokens are processed (in addition to those string constants shown in the
 6633 grammar):

6634 **COLL_ELEM** Any single-character collating element, unless it is a META_CHAR.

6635 **BACKREF** Applicable only to basic regular expressions. The character string
 6636 consisting of '\ ' followed by a single-digit numeral, '1' to '9'.

6637 **DUP_COUNT** Represents a numeric constant. It shall be an integer in the range 0
 6638 ≤DUP_COUNT ≤{RE_DUP_MAX}. This token is only recognized when
 6639 the context of the grammar requires it. At all other times, digits not
 6640 preceded by '\ ' are treated as ORD_CHAR.

6641 **META_CHAR** One of the characters:

6642 ^ When found first in a bracket expression

6643 - When found anywhere but first (after an initial '^', if any) or
 6644 last in a bracket expression, or as the ending range point in a
 6645 range expression

6646] When found anywhere but first (after an initial '^', if any) in a
 6647 bracket expression

6648 **L_ANCHOR** Applicable only to basic regular expressions. The character '^' when it
 6649 appears as the first character of a basic regular expression and when not
 6650 QUOTED_CHAR. The '^' may be recognized as an anchor elsewhere;
 6651 see Section 9.3.8 (on page 202).

6652 **ORD_CHAR** A character, other than one of the special characters in SPEC_CHAR.

6653 **QUOTED_CHAR** In a BRE, one of the character sequences:

6654 \^ \. * \[\\$ \\

6655 In an ERE, one of the character sequences:

6656 \^ \. \[\\$ \(\) \|
 6657 * \+ \? \{ \\

6658 **R_ANCHOR** (Applicable only to basic regular expressions.) The character '\$' when it
 6659 appears as the last character of a basic regular expression and when not
 6660 QUOTED_CHAR. The '\$' may be recognized as an anchor elsewhere;
 6661 see Section 9.3.8 (on page 202).

6662 **SPEC_CHAR** For basic regular expressions, one of the following special characters:

6663 . Anywhere outside bracket expressions

6664 \ Anywhere outside bracket expressions

```

6665      [      Anywhere outside bracket expressions
6666      ^      When used as an anchor (see Section 9.3.8 (on page 202)) or
6667            when first in a bracket expression
6668      $      When used as an anchor
6669      *      Anywhere except first in an entire RE, anywhere in a bracket
6670            expression, directly following "\(", directly following an
6671            anchoring '^'
6672
6673      For extended regular expressions, shall be one of the following special
6674            characters found anywhere outside bracket expressions:
6675
6676            ^      .      [      $      (      )      |
6677            *      +      ?      {      \

```

(The close-parenthesis shall be considered special in this context only if matched with a preceding open-parenthesis.)

6678 9.5.2 RE and Bracket Expression Grammar

6679 This section presents the grammar for basic regular expressions, including the bracket
6680 expression grammar that is common to both BREs and EREs.

```

6681 %token   ORD_CHAR QUOTED_CHAR DUP_COUNT
6682 %token   BACKREF L_ANCHOR R_ANCHOR
6683 %token   Back_open_paren  Back_close_paren
6684 /*      '\('           '\)'           */
6685 %token   Back_open_brace  Back_close_brace
6686 /*      '\{'           '\}'           */
6687 /* The following tokens are for the Bracket Expression
6688    grammar common to both REs and EREs. */
6689 %token   COLL_ELEM META_CHAR
6690 %token   Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close
6691 /*      '['           '='           '[' '.'           '.']'           '[' ':'           ':']' */
6692 %token   class_name
6693 /* class_name is a keyword to the LC_CTYPE locale category */
6694 /* (representing a character class) in the current locale */
6695 /* and is only recognized between [: and :] */
6696 %start   basic_reg_exp
6697 %%
6698 /* -----
6699    Basic Regular Expression
6700    -----
6701 */
6702 basic_reg_exp :      RE_expression
6703              | L_ANCHOR
6704              |      R_ANCHOR
6705              | L_ANCHOR      R_ANCHOR
6706              | L_ANCHOR RE_expression
6707              |      RE_expression R_ANCHOR

```

```

6708         | L_ANCHOR RE_expression R_ANCHOR
6709         ;
6710 RE_expression :          simple_RE
6711         | RE_expression simple_RE
6712         ;
6713 simple_RE      : nondupl_RE
6714         | nondupl_RE RE_dupl_symbol
6715         ;
6716 nondupl_RE     : one_character_RE
6717         | Back_open_paren RE_expression Back_close_paren
6718         | BACKREF
6719         ;
6720 one_character_RE : ORD_CHAR
6721         | QUOTED_CHAR
6722         | '.'
6723         | bracket_expression
6724         ;
6725 RE_dupl_symbol : '*'
6726         | Back_open_brace DUP_COUNT          Back_close_brace
6727         | Back_open_brace DUP_COUNT ','      Back_close_brace
6728         | Back_open_brace DUP_COUNT ',' DUP_COUNT Back_close_brace
6729         ;

6730 /* -----
6731    Bracket Expression
6732    -----
6733 */
6734 bracket_expression : '[' matching_list ']'
6735         | '[' nonmatching_list ']'
6736         ;
6737 matching_list      : bracket_list
6738         ;
6739 nonmatching_list   : '^' bracket_list
6740         ;
6741 bracket_list       : follow_list
6742         | follow_list '-'
6743         ;
6744 follow_list        :          expression_term
6745         | follow_list expression_term
6746         ;
6747 expression_term    : single_expression
6748         | range_expression
6749         ;
6750 single_expression  : end_range
6751         | character_class
6752         | equivalence_class
6753         ;
6754 range_expression   : start_range end_range
6755         | start_range '-'
6756         ;
6757 start_range        : end_range '-'
6758         ;
6759 end_range          : COLL_ELEM

```

```

6760         | collating_symbol
6761         ;
6762 collating_symbol : Open_dot COLL_ELEM Dot_close
6763         | Open_dot META_CHAR Dot_close
6764         ;
6765 equivalence_class : Open_equal COLL_ELEM Equal_close
6766         ;
6767 character_class : Open_colon class_name Colon_close
6768         ;

```

6769 The BRE grammar does not permit L_ANCHOR or R_ANCHOR inside "\(" and "\)" (which
6770 implies that '^' and '\$' are ordinary characters). This reflects the semantic limits on the
6771 application, as noted in Section 9.3.8 (on page 202). Implementations are permitted to extend the
6772 language to interpret '^' and '\$' as anchors in these locations, and as such, portable
6773 applications cannot use unescaped '^' and '\$' in positions inside "\(" and "\)" that might
6774 be interpreted as anchors.

6775 9.5.3 ERE Grammar

6776 This section presents the grammar for extended regular expressions, excluding the bracket
6777 expression grammar.

6778 **Note:** The bracket expression grammar and the associated %token lines are identical
6779 between BREs and EREs. It has been omitted from the ERE section to avoid
6780 unnecessary editorial duplication.

```

6781 %token ORD_CHAR QUOTED_CHAR DUP_COUNT
6782 %start extended_reg_exp
6783 %%
6784 /* -----
6785    Extended Regular Expression
6786    -----
6787 */
6788 extended_reg_exp : ERE_branch
6789         | extended_reg_exp '|' ERE_branch
6790         ;
6791 ERE_branch : ERE_expression
6792         | ERE_branch ERE_expression
6793         ;
6794 ERE_expression : one_character_ERE
6795         | '^'
6796         | '$'
6797         | '(' extended_reg_exp ')'
6798         | ERE_expression ERE_dupl_symbol
6799         ;
6800 one_character_ERE : ORD_CHAR
6801         | QUOTED_CHAR
6802         | '.'
6803         | bracket_expression
6804         ;
6805 ERE_dupl_symbol : '*'
6806         | '+'
6807         | '?'
6808         | '{' DUP_COUNT '}'

```

```

6809         | '{' DUP_COUNT ',' '}'
6810         | '{' DUP_COUNT ',' DUP_COUNT '}'
6811         ;

```

6812 The ERE grammar does not permit several constructs that previous sections specify as having
 6813 undefined results:

- 6814 • ORD_CHAR preceded by '\'
- 6815 • One or more *ERE_dupl_symbols* appearing first in an ERE, or immediately following '|',
 6816 '^', or '('
- 6817 • '{' not part of a valid *ERE_dupl_symbol*
- 6818 • '|' appearing first or last in an ERE, or immediately following '|' or '(', or immediately
 6819 preceding ')'

6820 Implementations are permitted to extend the language to allow these. Portable applications |
 6821 cannot use such constructs.

Directory Structure and Devices

6822

6823 10.1 Directory Structure and Files

6824 The following directories shall exist on conforming systems and portable applications shall
6825 make use of them only as described. Portable applications shall not assume the ability to create
6826 files in any of these directories, unless specified below.

6827 / The root directory.

6828 /dev Contains /dev/console, /dev/null, and /dev/tty, described below.

6829 The following directory shall exist on conforming systems and shall be used as described.

6830 /tmp A directory made available for programs that need a place to create temporary
6831 files. Applications are allowed to create files in this directory, but cannot assume
6832 that such files are preserved between invocations of the application.

6833 The following files shall exist on conforming systems and shall be both readable and writable.

6834 /dev/null An infinite data source and data sink. Data written to /dev/null shall be discarded.
6835 Reads from /dev/null shall always return end-of-file (EOF).

6836 /dev/tty In each process, a synonym for the controlling terminal associated with the process
6837 group of that process, if any. It is useful for programs or shell procedures that wish
6838 to be sure of writing messages to or reading data from the terminal no matter how
6839 output has been redirected. It can also be used for programs that demand the name
6840 of a file for output, when typed output is desired and it is tiresome to find out
6841 what terminal is currently in use.

6842 The following file shall exist on conforming systems and need not be readable or writable:

6843 /dev/console The /dev/console file is a generic name given to the system console. It is usually
6844 linked to a particular machine-dependent special file. It shall provide a basic I/O
6845 interface to the system console.

6846 10.2 Output Devices and Terminal Types

6847 The utilities in the Shell and Utilities volume of IEEE Std. 1003.1-200x historically have been
6848 implemented on a wide range of terminal types, but a conforming implementation need not
6849 support all features of all utilities on every conceivable terminal. IEEE Std. 1003.1-200x states
6850 which features are optional for certain classes of terminals in the individual utility description
6851 sections. The implementation shall document which terminal types it supports and which of
6852 these features and utilities are not supported by each terminal.

6853 When a feature or utility is not supported on a specific terminal type, as allowed by
6854 IEEE Std. 1003.1-200x, and the implementation considers such a condition to be an error
6855 preventing use of the feature or utility, the implementation shall indicate such conditions
6856 through diagnostic messages or exit status values or both (as appropriate to the specific utility
6857 description) that inform the user that the terminal type lacks the appropriate capability.

6858 IEEE Std. 1003.1-200x uses a notational convention based on historical practice that identifies
6859 some of the control characters defined in Section 7.3.1 (on page 147) in a manner easily

6860 remembered by users on many terminals. The correspondence between this “<control>-char”
 6861 notation and the actual control characters is shown in the following table. When
 6862 IEEE Std. 1003.1-200x refers to a character by its <control>- name, it is referring to the actual
 6863 control character shown in the Value column of the table, which is not necessarily the exact
 6864 control key sequence on all terminals. Some terminals have keyboards that do not allow the
 6865 direct transmission of all the non-alphanumeric characters shown. In such cases, the system
 6866 documentation shall describe which data sequences transmitted by the terminal are interpreted
 6867 by the system as representing the special characters.

6868 **Table 10-1** Control Character Names

Name	Value	Symbolic Name	Name	Value	Symbolic Name
<control>-A	<SOH>	<SOH>	<control>-Q	<DC1>	<DC1>
<control>-B	<STX>	<STX>	<control>-R	<DC2>	<DC2>
<control>-C	<ETX>	<ETX>	<control>-S	<DC3>	<DC3>
<control>-D	<EOT>	<EOT>	<control>-T	<DC4>	<DC4>
<control>-E	<ENQ>	<ENQ>	<control>-U	<NAK>	<NAK>
<control>-F	<ACK>	<ACK>	<control>-V	<SYN>	<SYN>
<control>-G	<BEL>	<alert>	<control>-W	<ETB>	<ETB>
<control>-H	<BS>	<backspace>	<control>-X	<CAN>	<CAN>
<control>-I	<HT>	<tab>	<control>-Y		
<control>-J	<LF>	<linefeed>	<control>-Z	<SUB>	<SUB>
<control>-K	<VT>	<vertical-tab>	<control>-[<ESC>	<ESC>
<control>-L	<FF>	<form-feed>	<control>-\ <control>-]	<FS>	<FS>
<control>-M	<CR>	<carriage-return>	<control>-^	<GS>	<GS>
<control>-N	<SO>	<SO>	<control>-_	<RS>	<RS>
<control>-O	<SI>	<SI>	<control>-_	<US>	<US>
<control>-P	<DLE>	<DLE>	<control>-?		

6886 **Note:** The notation uses uppercase letters for arbitrary editorial reasons. There is no
 6887 implication that the keystrokes represent control-shift-letter sequences.

General Terminal Interface

6888

6889 This chapter describes a general terminal interface that shall be provided. It shall be supported
6890 on any asynchronous communications ports if the implementation provides them. It is
6891 implementation-defined whether it supports network connections or synchronous ports, or
6892 both.

6893 11.1 Interface Characteristics

6894 11.1.1 Opening a Terminal Device File

6895 When a terminal device file is opened, it normally causes the thread to wait until a connection is
6896 established. In practice, application programs seldom open these files; they are opened by
6897 special programs and become an application's standard input, output, and error files.

6898 As described in *open()*, opening a terminal device file with the *O_NONBLOCK* flag clear shall
6899 cause the thread to block until the terminal device is ready and available. If *CLOCAL* mode is
6900 not set, this means blocking until a connection is established. If *CLOCAL* mode is set in the
6901 terminal, or the *O_NONBLOCK* flag is specified in the *open()*, the *open()* function shall return a
6902 file descriptor without waiting for a connection to be established.

6903 11.1.2 Process Groups

6904 A terminal may have a foreground process group associated with it. This foreground process
6905 group plays a special role in handling signal-generating input characters, as discussed in Section
6906 11.1.9 (on page 217).

6907 A command interpreter process supporting job control can allocate the terminal to different jobs,
6908 or process groups, by placing related processes in a single process group and associating this
6909 process group with the terminal. A terminal's foreground process group may be set or examined
6910 by a process, assuming the permission requirements are met; see *tcgetpgrp()* and *tcsetpgrp()*. The
6911 terminal interface aids in this allocation by restricting access to the terminal by processes that are
6912 not in the current process group; see Section 11.1.4 (on page 214).

6913 When there is no longer any process whose process ID or process group ID matches the process
6914 group ID of the foreground process group, the terminal shall have no foreground process group.
6915 It is unspecified whether the terminal has a foreground process group when there is a process
6916 whose process ID matches the foreground process ID, but whose process group ID does not. No
6917 actions defined in IEEE Std. 1003.1-200x, other than allocation of a controlling terminal or a
6918 successful call to *tcsetpgrp()*, cause a process group to become the foreground process group of
6919 the terminal.

6920 11.1.3 The Controlling Terminal

6921 A terminal may belong to a process as its controlling terminal. Each process of a session that has
6922 a controlling terminal has the same controlling terminal. A terminal may be the controlling
6923 terminal for at most one session. The controlling terminal for a session is allocated by the session
6924 leader in an implementation-defined manner. If a session leader has no controlling terminal, and
6925 opens a terminal device file that is not already associated with a session without using the
6926 O_NOCTTY option (see *open()*), it is implementation-defined whether the terminal becomes the
6927 controlling terminal of the session leader. If a process which is not a session leader opens a
6928 terminal file, or the O_NOCTTY option is used on *open()*, then that terminal shall not become
6929 the controlling terminal of the calling process. When a controlling terminal becomes associated
6930 with a session, its foreground process group shall be set to the process group of the session
6931 leader.

6932 The controlling terminal is inherited by a child process during a *fork()* function call. A process
6933 relinquishes its controlling terminal when it creates a new session with the *setsid()* function;
6934 other processes remaining in the old session that had this terminal as their controlling terminal
6935 continue to have it. Upon the close of the last file descriptor in the system (whether or not it is in
6936 the current session) associated with the controlling terminal, it is unspecified whether all
6937 processes that had that terminal as their controlling terminal cease to have any controlling
6938 terminal. Whether and how a session leader can reacquire a controlling terminal after the
6939 controlling terminal has been relinquished in this fashion is unspecified. A process does not
6940 relinquish its controlling terminal simply by closing all of its file descriptors associated with the
6941 controlling terminal if other processes continue to have it open.

6942 When a controlling process terminates, the controlling terminal is dissociated from the current
6943 session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by
6944 other processes in the earlier session may be denied, with attempts to access the terminal treated
6945 as if a modem disconnect had been sensed.

6946 11.1.4 Terminal Access Control

6947 If a process is in the foreground process group of its controlling terminal, read operations shall
6948 be allowed, as described in Section 11.1.5 (on page 215). Any attempts by a process in a
6949 background process group to read from its controlling terminal cause its process group to be
6950 sent a SIGTTIN signal unless one of the following special cases applies: if the reading process is
6951 ignoring or blocking the SIGTTIN signal, or if the process group of the reading process is
6952 orphaned, the *read()* returns -1 , with *errno* set to [EIO] and no signal is sent. The default action of
6953 the SIGTTIN signal is to stop the process to which it is sent. See <**signal.h**>.

6954 If a process is in the foreground process group of its controlling terminal, write operations shall
6955 be allowed as described in Section 11.1.8 (on page 217). Attempts by a process in a background
6956 process group to write to its controlling terminal shall cause the process group to be sent a
6957 SIGTTOU signal unless one of the following special cases applies: if TOSTOP is not set, or if
6958 TOSTOP is set and the process is ignoring or blocking the SIGTTOU signal, the process is
6959 allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, and the
6960 process group of the writing process is orphaned, and the writing process is not ignoring or
6961 blocking the SIGTTOU signal, the *write()* returns -1 , with *errno* set to [EIO] and no signal is sent.

6962 Certain calls that set terminal parameters are treated in the same fashion as *write()*, except that
6963 TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set
6964 (see Section 11.2.5 (on page 223), *tcdrain()*, *tcfLOW()*, *tcfLush()*, *tcsendbreak()*, *tcsetattr()*, and
6965 *tcsetpgrp()*).

6966 11.1.5 Input Processing and Reading Data

6967 A terminal device associated with a terminal device file may operate in full-duplex mode, so that
 6968 data may arrive even while output is occurring. Each terminal device file has an *input queue*,
 6969 associated with it, into which incoming data is stored by the system before being read by a
 6970 process. The system may impose a limit, {MAX_INPUT}, on the number of bytes that may be
 6971 stored in the input queue. The behavior of the system when this limit is exceeded is
 6972 implementation-defined.

6973 Two general kinds of input processing are available, determined by whether the terminal device
 6974 file is in canonical mode or non-canonical mode. These modes are described in Section 11.1.6 and
 6975 Section 11.1.7 (on page 216). Additionally, input characters are processed according to the
 6976 *c_iflag* (see Section 11.2.2 (on page 219)) and *c_lflag* (see Section 11.2.5 (on page 223)) fields.
 6977 Such processing can include *echoing*, which in general means transmitting input characters
 6978 immediately back to the terminal when they are received from the terminal. This is useful for
 6979 terminals that can operate in full-duplex mode.

6980 The manner in which data is provided to a process reading from a terminal device file is
 6981 dependent on whether the terminal file is in canonical or non-canonical mode, and on whether
 6982 or not the O_NONBLOCK flag is set by *open()* or *fcntl()*.

6983 If the O_NONBLOCK flag is clear, then the read request shall be blocked until data is available
 6984 or a signal has been received. If the O_NONBLOCK flag is set, then the read request shall be
 6985 completed, without blocking, in one of three ways:

- 6986 1. If there is enough data available to satisfy the entire request, the *read()* shall complete
 6987 successfully and shall return the number of bytes read.
- 6988 2. If there is not enough data available to satisfy the entire request, the *read()* shall complete
 6989 successfully, having read as much data as possible, and shall return the number of bytes it
 6990 was able to read.
- 6991 3. If there is no data available, the *read()* shall return -1 , with *errno* set to [EAGAIN].

6992 When data is available depends on whether the input processing mode is canonical or non-
 6993 canonical. The following sections, Section 11.1.6 and Section 11.1.7 (on page 216), describe each
 6994 of these input processing modes.

6995 11.1.6 Canonical Mode Input Processing

6996 In canonical mode input processing, terminal input is processed in units of lines. A line is
 6997 delimited by a newline character (NL), an end-of-file character (EOF), or an end-of-line (EOL)
 6998 character. See Section 11.1.9 (on page 217) for more information on EOF and EOL. This means
 6999 that a read request shall not return until an entire line has been typed or a signal has been
 7000 received. Also, no matter how many bytes are requested in the *read()* call, at most one line shall
 7001 be returned. It is not, however, necessary to read a whole line at once; any number of bytes, even
 7002 one, may be requested in a *read()* without losing information.

7003 If {MAX_CANON} is defined for this terminal device, it is a limit on the number of bytes in a
 7004 line. The behavior of the system when this limit is exceeded is implementation-defined. If
 7005 {MAX_CANON} is not defined, there is no such limit; see *pathconf()*.

7006 Erase and kill processing occur when either of two special characters, the ERASE and KILL
 7007 characters (see Section 11.1.9 (on page 217)), is received. This processing affects data in the input
 7008 queue that has not yet been delimited by a newline (NL), EOF, or EOL character. This un-
 7009 delimited data makes up the current line. The ERASE character deletes the last character in the
 7010 current line, if there is one. The KILL character deletes all data in the current line, if there are any.
 7011 The ERASE and KILL characters have no effect if there is no data in the current line. The ERASE

7012 and KILL characters themselves are not placed in the input queue.

7013 11.1.7 Non-Canonical Mode Input Processing

7014 In non-canonical mode input processing, input bytes are not assembled into lines, and erase and
7015 kill processing do not occur. The values of the MIN and TIME members of the `c_cc` array are
7016 used to determine how to process the bytes received. The IEEE Std. 1003.1-200x does not specify
7017 whether the setting of `O_NONBLOCK` takes precedence over MIN or TIME settings. Therefore,
7018 if `O_NONBLOCK` is set, `read()` may return immediately, regardless of the setting of MIN or
7019 TIME. Also, if no data is available, `read()` may either return 0, or return -1 with `errno` set to
7020 `[EAGAIN]`.

7021 MIN represents the minimum number of bytes that should be received when the `read()` function
7022 returns successfully. TIME is a timer of 0.1 second granularity that is used to time out bursty and
7023 short-term data transmissions. If MIN is greater than `{MAX_INPUT}`, the response to the request
7024 is undefined. The four possible values for MIN and TIME and their interactions are described
7025 below.

7026 Case A: MIN>0, TIME>0

7027 In case A, TIME serves as an inter-byte timer and is activated after the first byte is received. Since
7028 it is an inter-byte timer, it is reset after a byte is received. The interaction between MIN and
7029 TIME is as follows. As soon as one byte is received, the inter-byte timer is started. If MIN bytes
7030 are received before the inter-byte timer expires (remember that the timer is reset upon receipt of
7031 each byte), the read is satisfied. If the timer expires before MIN bytes are received, the characters
7032 received to that point are returned to the user. Note that if TIME expires at least one byte is
7033 returned because the timer would not have been enabled unless a byte was received. In this case
7034 (MIN>0, TIME>0) the read shall block until the MIN and TIME mechanisms are activated by the
7035 receipt of the first byte, or a signal is received. If the data is in the buffer at the time of the `read()`,
7036 the result shall be as if the data has been received immediately after the `read()`.

7037 Case B: MIN>0, TIME=0

7038 In case B, since the value of TIME is zero, the timer plays no role and only MIN is significant. A
7039 pending read is not satisfied until MIN bytes are received (that is, the pending read shall block
7040 until MIN bytes are received), or a signal is received. A program that uses case B to read record-
7041 based terminal I/O may block indefinitely in the read operation.

7042 Case C: MIN=0, TIME>0

7043 In case C, since MIN=0, TIME no longer represents an inter-byte timer. It now serves as a read
7044 timer that is activated as soon as the `read()` function is processed. A read is satisfied as soon as a
7045 single byte is received or the read timer expires. Note that in case C if the timer expires, no bytes
7046 are returned. If the timer does not expire, the only way the read can be satisfied is if a byte is
7047 received. If bytes are not received, the read shall not block indefinitely waiting for a byte; if no
7048 byte is received within $TIME * 0.1$ seconds after the read is initiated, the `read()` returns a value of
7049 zero, having read no data. If the data is in the buffer at the time of the `read()`, the timer shall be
7050 started as if the data has been received immediately after the `read()`.

7051 **Case D: MIN=0, TIME=0**

7052 The minimum of either the number of bytes requested or the number of bytes currently available
 7053 shall be returned without waiting for more bytes to be input. If no characters are available, *read()*
 7054 shall return a value of zero, having read no data.

7055 **11.1.8 Writing Data and Output Processing**

7056 When a process writes one or more bytes to a terminal device file, they are processed according
 7057 to the *c_oflag* field (see Section 11.2.3 (on page 220)). The implementation may provide a
 7058 buffering mechanism; as such, when a call to *write()* completes, all of the bytes written have
 7059 been scheduled for transmission to the device, but the transmission has not necessarily
 7060 completed. See *write()* for the effects of *O_NONBLOCK* on *write()*.

7061 **11.1.9 Special Characters**

7062 Certain characters have special functions on input or output or both. These functions are
 7063 summarized as follows:

7064 **INTR** Special character on input, which is recognized if the *ISIG* flag is set. Generates a
 7065 *SIGINT* signal which is sent to all processes in the foreground process group for which
 7066 the terminal is the controlling terminal. If *ISIG* is set, the *INTR* character is discarded
 7067 when processed.

7068 **QUIT** Special character on input, which is recognized if the *ISIG* flag is set. Generates a
 7069 *SIGQUIT* signal which is sent to all processes in the foreground process group for
 7070 which the terminal is the controlling terminal. If *ISIG* is set, the *QUIT* character is
 7071 discarded when processed.

7072 **ERASE** Special character on input, which is recognized if the *ICANON* flag is set. Erases the
 7073 last character in the current line; see Section 11.1.6 (on page 215). It shall not erase
 7074 beyond the start of a line, as delimited by an *NL*, *EOF*, or *EOL* character. If *ICANON* is
 7075 set, the *ERASE* character is discarded when processed.

7076 **KILL** Special character on input, which is recognized if the *ICANON* flag is set. Deletes the
 7077 entire line, as delimited by an *NL*, *EOF*, or *EOL* character. If *ICANON* is set, the *KILL*
 7078 character is discarded when processed.

7079 **EOF** Special character on input, which is recognized if the *ICANON* flag is set. When
 7080 received, all the bytes waiting to be read are immediately passed to the process without
 7081 waiting for a newline, and the *EOF* is discarded. Thus, if there are no bytes waiting
 7082 (that is, the *EOF* occurred at the beginning of a line), a byte count of zero shall be
 7083 returned from the *read()*, representing an end-of-file indication. If *ICANON* is set, the
 7084 *EOF* character is discarded when processed.

7085 **NL** Special character on input, which is recognized if the *ICANON* flag is set. It is the line
 7086 delimiter newline. It cannot be changed.

7087 **EOL** Special character on input, which is recognized if the *ICANON* flag is set. It is an
 7088 additional line delimiter, like *NL*.

7089 **SUSP** If the *ISIG* flag is set, receipt of the *SUSP* character causes a *SIGTSTP* signal to be sent
 7090 to all processes in the foreground process group for which the terminal is the
 7091 controlling terminal, and the *SUSP* character is discarded when processed.

7092 **STOP** Special character on both input and output, which is recognized if the *IXON* (output
 7093 control) or *IXOFF* (input control) flag is set. Can be used to suspend output
 7094 temporarily. It is useful with CRT terminals to prevent output from disappearing

7095 before it can be read. If IXON is set, the STOP character is discarded when processed.

7096 START Special character on both input and output, which is recognized if the IXON (output
7097 control) or IXOFF (input control) flag is set. Can be used to resume output that has
7098 been suspended by a STOP character. If IXON is set, the START character is discarded
7099 when processed.

7100 CR Special character on input, which is recognized if the ICANON flag is set; it is the
7101 carriage-return character. When ICANON and ICRNL are set and IGNCR is not set,
7102 this character is translated into an NL, and has the same effect as an NL character.

7103 The NL and CR characters cannot be changed. It is implementation-defined whether the START
7104 and STOP characters can be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and
7105 SUSP shall be changeable to suit individual tastes. Special character functions associated with
7106 changeable special control characters can be disabled individually.

7107 If two or more special characters have the same value, the function performed when that
7108 character is received is undefined.

7109 A special character is recognized not only by its value, but also by its context; for example, an
7110 implementation may support multi-byte sequences that have a meaning different from the
7111 meaning of the bytes when considered individually. Implementations may also support
7112 additional single-byte functions. These implementation-defined multi-byte or single-byte
7113 functions are recognized only if the IEXTEN flag is set; otherwise, data is received without
7114 interpretation, except as required to recognize the special characters defined in this section.

7115 XSI If IEXTEN is set, the ERASE, KILL, and EOF characters can be escaped by a preceding '\'
7116 character, in which case no special function occurs.

7117 11.1.10 Modem Disconnect

7118 If a modem disconnect is detected by the terminal interface for a controlling terminal, and if
7119 CLOCAL is not set in the `c_cflag` field for the terminal (see Section 11.2.4 (on page 222)), the
7120 SIGHUP signal is sent to the controlling process for which the terminal is the controlling
7121 terminal. Unless other arrangements have been made, this causes the controlling process to
7122 terminate (see `exit()`). Any subsequent read from the terminal device shall return the value of
7123 zero, indicating end-of-file; see `read()`. Thus, processes that read a terminal file and test for end-
7124 of-file can terminate appropriately after a disconnect. If the EIO condition as specified in `read()`
7125 also exists, it is unspecified whether on EOF condition or the [EIO] is returned. Any subsequent
7126 `write()` to the terminal device returns `-1`, with `errno` set to [EIO], until the device is closed.

7127 11.1.11 Closing a Terminal Device File

7128 The last process to close a terminal device file shall cause any output to be sent to the device and
7129 any input to be discarded. If HUPCL is set in the control modes and the communications port
7130 supports a disconnect function, the terminal device shall perform a disconnect.

7131 11.2 Parameters that Can be Set

7132 11.2.1 The termios Structure

7133 Routines that need to control certain terminal I/O characteristics shall do so by using the
7134 **termios** structure as defined in the `<termios.h>` header. The members of this structure include
7135 (but are not limited to):

Member Type	Array Size	Member Name	Description
<code>tcflag_t</code>		<code>c_iflag</code>	Input modes.
<code>tcflag_t</code>		<code>c_oflag</code>	Output modes.
<code>tcflag_t</code>		<code>c_cflag</code>	Control modes.
<code>tcflag_t</code>		<code>c_lflag</code>	Local modes.
<code>cc_t</code>	NCCS	<code>c_cc[]</code>	Control characters.

7143 The types `tcflag_t` and `cc_t` are defined in the `<termios.h>` header. They shall be unsigned
7144 integer types.

7145 11.2.2 Input Modes

7146 Values of the `c_iflag` field describe the basic terminal input control, and are composed of the
7147 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
7148 symbols in this table are defined in `<termios.h>`:

Mask Name	Description
BRKINT	Signal interrupt on break.
ICRNL	Map CR to NL on input.
IGNBRK	Ignore break condition.
IGNCR	Ignore CR.
IGNPAR	Ignore characters with parity errors.
INLCR	Map NL to CR on input.
INPCK	Enable input parity check.
ISTRIP	Strip character.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IXON	Enable start/stop output control.
PARMRK	Mark parity errors.

7163 In the context of asynchronous serial data transmission, a break condition is defined as a
7164 sequence of zero-valued bits that continues for more than the time to send one byte. The entire
7165 sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a
7166 time equivalent to more than one byte. In contexts other than asynchronous serial data
7167 transmission, the definition of a break condition is implementation-defined.

7168 If IGNBRK is set, a break condition detected on input is ignored; that is, not put on the input
7169 queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the break
7170 condition shall flush the input and output queues, and if the terminal is the controlling terminal
7171 of a foreground process group, the break condition shall generate a single SIGINT signal to that
7172 foreground process group. If neither IGNBRK nor BRKINT is set, a break condition is read as a
7173 single 0x00, or if PARMRK is set, as 0xff 0x00 0x00.

7174 If IGNPAR is set, a byte with a framing or parity error (other than break) is ignored.

7175 If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than
7176 break) is given to the application as the three-byte sequence 0xff 0x00 X, where 0xff 0x00 is a
7177 two-byte flag preceding each sequence and X is the data of the byte received in error. To avoid
7178 ambiguity in this case, if ISTRIP is not set, a valid byte of 0xff is given to the application as 0xff
7179 0xff. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) is given
7180 to the application as a single byte 0x00.

7181 If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is
7182 disabled, allowing output parity generation without input parity errors. Note that whether input
7183 parity checking is enabled or disabled is independent of whether parity detection is enabled or
7184 disabled (see Section 11.2.4 (on page 222)). If parity detection is enabled but input parity
7185 checking is disabled, the hardware to which the terminal is connected shall recognize the parity
7186 bit, but the terminal special file shall not check whether or not this bit is correctly set.

7187 If ISTRIP is set, valid input bytes are first stripped to seven bits; otherwise, all eight bits are
7188 processed.

7189 If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a
7190 received CR character is ignored (not read). If IGNCR is not set and ICRNL is set, a received CR
7191 character is translated into an NL character.

7192 XSI If IXANY is set, any input character shall restart output that has been suspended.

7193 If IXON is set, start/stop output control is enabled. A received STOP character shall suspend
7194 output and a received START character shall restart output. When IXON is set, START and
7195 STOP characters are not read, but merely perform flow control functions. When IXON is not set,
7196 the START and STOP characters are read.

7197 If IXOFF is set, start/stop input control is enabled. The system shall transmit STOP characters,
7198 which are intended to cause the terminal device to stop transmitting data, as needed to prevent
7199 the input queue from overflowing and causing implementation-defined behavior, and shall
7200 transmit START characters, which are intended to cause the terminal device to resume
7201 transmitting data, as soon as the device can continue transmitting data without risk of
7202 overflowing the input queue. The precise conditions under which STOP and START characters
7203 are transmitted are implementation-defined.

7204 The initial input control value after *open()* is implementation-defined.

7205 11.2.3 Output Modes

7206 The **c_oflag** field specifies the terminal interface's treatment of output, and is composed of the
7207 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
7208 symbols in this table are defined in `<termios.h>`:

7209
7210
7211
7212 XSI
7213
7214
7215
7216
7217
7218
7219
7220
7221
7222
7223
7224
7225
7226
7227
7228
7229
7230
7231
7232
7233
7234
7235
7236
7237
7238
7239

Mask Name	Description
OPOST	Perform output processing.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NUL.
NLDLY	Select newline delays:
NL0	Newline character type 0.
NL1	Newline character type 1.
CRDLY	Select carriage-return delays:
CR0	Carriage-return delay type 0.
CR1	Carriage-return delay type 1.
CR2	Carriage-return delay type 2.
CR3	Carriage-return delay type 3.
TABDLY	Select horizontal-tab delays:
TAB0	Horizontal-tab delay type 0.
TAB1	Horizontal-tab delay type 1.
TAB2	Horizontal-tab delay type 2.
TAB3	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	Backspace-delay type 0.
BS1	Backspace-delay type 1.
VTDLY	Select vertical-tab delays:
VT0	Vertical-tab delay type 0.
VT1	Vertical-tab delay type 1.
FFDLY	Select form-feed delays:
FF0	Form-feed delay type 0.
FF1	Form-feed delay type 1.

7240
7241
7242

7243 XSI
7244
7245
7246
7247
7248
7249

7250
7251
7252
7253
7254

7255

7256
7257

If OPOST is set, output data is post-processed as described below, so that lines of text are modified to appear appropriately on the terminal device; otherwise, characters are transmitted without change.

If ONLCR is set, the NL character shall be transmitted as the CR-NL character pair. If OCRNL is set, the CR character shall be transmitted as the NL character. If ONOCR is set, no CR character shall be transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer shall be set to 0 and the delays specified for CR shall be used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer shall also be set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay. If OFILL is set, fill characters are transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL; otherwise, NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the newline delays. If OFILL is set, two fill characters are transmitted.

7258 Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10
7259 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill
7260 characters, and type 2, four fill characters.

7261 Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10
7262 seconds. Type 3 specifies that tabs shall be expanded into spaces. If OFILL is set, two fill
7263 characters are transmitted for any delay.

7264 Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character is transmitted.

7265 The actual delays depend on line speed and system load.

7266 The initial output control value after *open()* is implementation-defined.

7267 11.2.4 Control Modes

7268 The **c_cflag** field describes the hardware control of the terminal, and is composed of the
7269 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
7270 symbols in this table are defined in `<termios.h>`; not all values specified are required to be
7271 supported by the underlying hardware:

Mask Name	Description
CLOCAL	Ignore modem status lines.
CREAD	Enable receiver.
CSIZE	Number of bits transmitted or received per byte:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits.
CSTOPB	Send two stop bits, else one.
HUPCL	Hang up on last close.
PARENB	Parity enable.
PARODD	Odd parity, else even.

7284 In addition, the input and output baud rates are stored in the **termios** structure. The following
7285 values are supported:

Name	Description	Name	Description
B0	Hang up	B600	600 baud
B50	50 baud	B1200	1200 baud
B75	75 baud	B1800	1800 baud
B110	110 baud	B2400	2400 baud
B134	134.5 baud	B4800	4800 baud
B150	150 baud	B9600	9600 baud
B200	200 baud	B19200	19200 baud
B300	300 baud	B38400	38400 baud

7295 The following functions are provided for getting and setting the values of the input and output
7296 baud rates in the **termios** structure: *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, and *cfsetospeed()*.
7297 The effects on the terminal device do not become effective and not all errors are detected until
7298 the *tcsetattr()* function is successfully called.

7299 The CSIZE bits specify the number of transmitted or received bits per byte. If ISTRIP is not set,
7300 the value of all the other bits is unspecified. If ISTRIP is set, the value of all but the 7 low-order
7301 bits is zero, but the value of any other bits beyond CSIZE is unspecified when read. CSIZE does
7302 not include the parity bit, if any. If CSTOPB is set, two stop bits are used; otherwise, one stop

7303 bit. For example, at 110 baud, two stop bits are normally used.

7304 If CREAD is set, the receiver is enabled; otherwise, no characters shall be received.

7305 If PARENB is set, parity generation and detection is enabled and a parity bit is added to each
7306 byte. If parity is enabled, PARODD specifies odd parity if set; otherwise, even parity is used.

7307 If HUPCL is set, the modem control lines for the port are lowered when the last process with the
7308 port open closes the port or the process terminates. The modem connection shall be broken.

7309 If CLOCAL is set, a connection does not depend on the state of the modem status lines. If
7310 CLOCAL is clear, the modem status lines shall be monitored.

7311 Under normal circumstances, a call to the *open()* function shall wait for the modem connection
7312 to complete. However, if the O_NONBLOCK flag is set (see *open()*) or if CLOCAL has been set,
7313 the *open()* function shall return immediately without waiting for the connection.

7314 If the object for which the control modes are set is not an asynchronous serial connection, some
7315 of the modes may be ignored; for example, if an attempt is made to set the baud rate on a
7316 network connection to a terminal on another host, the baud rate may or may not be set on the
7317 connection between that terminal and the machine to which it is directly connected.

7318 The initial hardware control value after *open()* is implementation-defined.

7319 **11.2.5 Local Modes**

7320 The *c_lflag* field of the argument structure is used to control various functions. It is composed
7321 of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
7322 symbols in this table are defined in *<termios.h>*; not all values specified are required to be
7323 supported by the underlying hardware:

7324

7325

Mask Name	Description
ECHO	Enable echo.
ECHOE	Echo ERASE as an error correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo <newline>.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended (implementation-defined) functions.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt, quit or suspend.
TOSTOP	Send SIGTTOU for background output.

7326

7327

7328

7329

7330

7331

7332

7333

7334

7335 If ECHO is set, input characters are echoed back to the terminal. If ECHO is clear, input
7336 characters are not echoed.

7337 If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase, if
7338 possible, the last character in the current line from the display. If there were no character to
7339 erase, an implementation might echo an indication that this was the case, or do nothing.

7340 If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase the
7341 line from the display or shall echoe the newline character after the KILL character.

7342 If ECHONL and ICANON are set, the newline character shall be echoed even if ECHO is not set.

7343 If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions,
7344 and the assembly of input characters into lines delimited by NL, EOF, and EOL, as described in
7345 Section 11.1.6 (on page 215).

7346 If ICANON is not set, read requests are satisfied directly from the input queue. A read shall not
 7347 be satisfied until at least MIN bytes have been received or the timeout value TIME expired
 7348 between bytes. The time value represents tenths of a second. See Section 11.1.7 (on page 216) for
 7349 more details.

7350 If IEXTEN is set, implementation-defined functions are recognized from the input data. It is
 7351 implementation-defined how IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF.
 7352 If IEXTEN is not set, implementation-defined functions shall not be recognized and the
 7353 corresponding input characters are processed as described for ICANON, ISIG, IXON, and
 7354 IXOFF.

7355 If ISIG is set, each input character is checked against the special control characters INTR, QUIT,
 7356 and SUSP. If an input character matches one of these control characters, the function associated
 7357 with that character is performed. If ISIG is not set, no checking is done. Thus these special input
 7358 functions are possible only if ISIG is set.

7359 If NOFLSH is set, the normal flush of the input and output queues associated with the INTR,
 7360 QUIT, and SUSP characters shall not be done.

7361 If TOSTOP is set, the signal SIGTTOU is sent to the process group of a process that tries to write
 7362 to its controlling terminal if it is not in the foreground process group for that terminal. This
 7363 signal, by default, stops the members of the process group. Otherwise, the output generated by
 7364 that process is output to the current output stream. Processes that are blocking or ignoring
 7365 SIGTTOU signals are excepted and allowed to produce output, and the SIGTTOU signal is not
 7366 sent.

7367 The initial local control value after *open()* is implementation-defined.

7368 11.2.6 Special Control Characters

7369 The special control characters values are defined by the array *c_cc*. The subscript name and
 7370 description for each element in both canonical and non-canonical modes are as follows:

7371

7372

7373

7374

7375

7376

7377

7378

7379

7380

7381

7382

7383

7384

7385

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF		EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR	VINTR	INTR character
VKILL		KILL character
	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character

7386 The subscript values are unique, except that the VMIN and VTIME subscripts may have the
 7387 same values as the VEOF and VEOL subscripts, respectively.

7388 Implementations that do not support changing the START and STOP characters may ignore the
 7389 character values in the *c_cc* array indexed by the VSTART and VSTOP subscripts when
 7390 *tcsetattr()* is called, but shall return the value in use when *tcgetattr()* is called.

7391 The initial values of all control characters are implementation-defined. |

7392 If the value of one of the changeable special control characters (see Section 11.1.9 (on page 217))
7393 is `_POSIX_VDISABLE`, that function shall be disabled; that is, no input data is recognized as the
7394 disabled special character. If `ICANON` is not set, the value of `_POSIX_VDISABLE` has no special
7395 meaning for the `VMIN` and `VTIME` entries of the `c_cc` array.

7397

7398 **12.1 Utility Argument Syntax**

7399 This section describes the argument syntax of the standard utilities and introduces terminology
 7400 used throughout IEEE Std. 1003.1-200x for describing the arguments processed by the utilities.

7401 Within IEEE Std. 1003.1-200x, a special notation is used for describing the syntax of a utility's
 7402 arguments. Unless otherwise noted, all utility descriptions use this notation, which is illustrated
 7403 by this example (see the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.9.1, Simple
 7404 Commands):

```
7405 utility_name[-a][-b][-c option_argument]
7406 [-d|-e][-foption_argument][operand...]
```

7407 The notation used for the SYNOPSIS sections imposes requirements on the implementors of the
 7408 standard utilities and provides a simple reference for the application developer or system user.

7409 1. The utility in the example is named *utility_name*. It is followed by *options*, *option-*
 7410 *arguments*, and *operands*. The arguments that consist of hyphens and single letters or
 7411 digits, such as 'a', are known as *options* (or, historically, *flags*). Certain options are
 7412 followed by an *option-argument*, as shown with [-c *option_argument*]. The arguments
 7413 following the last options and option-arguments are named *operands*.

7414 2. Option-arguments are sometimes shown separated from their options by <blank>
 7415 characters, sometimes directly adjacent. This reflects the situation that in some cases an
 7416 option-argument is included within the same argument string as the option; in most cases
 7417 it is the next argument. The Utility Syntax Guidelines in Section 12.2 (on page 229) require
 7418 that the option be a separate argument from its option-argument, but there are some
 7419 exceptions in IEEE Std. 1003.1-200x to ensure continued operation of historical
 7420 applications:

7421 a. If the SYNOPSIS of a standard utility shows a space character between an option and
 7422 option-argument (as with [-c *option_argument*] in the example), a portable
 7423 application shall use separate arguments for that option and its option-argument.

7424 b. If a space character is not shown (as with [-*foption_argument*] in the example), a
 7425 portable application shall place an option and its option-argument directly adjacent
 7426 in the same argument string, without intervening <blank> characters.

7427 c. Notwithstanding the preceding requirements on portable applications, a conforming
 7428 system shall permit, but shall not require, an application to specify options and
 7429 option-arguments as separate arguments whether or not a space character is shown
 7430 XSI on the synopsis line, except in those cases (marked with the XSI portability warning)
 7431 where an option-argument is optional and no separation can be used.

7432 d. A standard utility may also be implemented to operate correctly when the required
 7433 separation into multiple arguments is violated by a non-portable application.

7434 In summary, the following table shows allowable combinations:

7435
7436
7437
7438
7439
7440
7441
7442
7443
7444
7445
7446
7447
7448
7449
7450
7451
7452
7453
7454
7455
7456
7457
7458
7459
7460
7461
7462
7463
7464
7465
7466
7467
7468
7469
7470
7471
7472
7473
7474
7475
7476
7477
7478

	SYNOPSIS Shows:		
	<i>-a arg</i>	<i>-barg</i>	<i>-c[arg]</i>
Portable application shall use:	<i>-a arg</i>	<i>-barg</i>	N/A
System shall support:	<i>-a arg</i>	<i>-barg</i>	<i>-carg</i> or <i>-c</i>
System may support:	<i>-aarg</i>	<i>-b arg</i>	

3. Options are usually listed in alphabetical order unless this would make the utility description more confusing. There are no implied relationships between the options based upon the order in which they appear, unless otherwise stated in the OPTIONS section, or unless the exception in Guideline 11 of Section 12.2 (on page 229) applies. If an option that does not have option-arguments is repeated, the results are undefined, unless otherwise stated.
4. Frequently, names of parameters that require substitution by actual values are shown with embedded underscores. Alternatively, parameters are shown as follows:

<parameter name>

The angle brackets are used for the symbolic grouping of a phrase representing a single parameter and portable applications shall not include them in data submitted to the utility.

5. When a utility has only a few permissible options, they are sometimes shown individually, as in the example. Utilities with many flags generally show all of the individual flags (that do not take option-arguments) grouped, as in:

utility_name [-abcDxyz][-p arg*][*operand*]*

Utilities with very complex arguments may be shown as follows:

*utility_name [options][*operands*]*

6. Unless otherwise specified, whenever an operand or option-argument is, or contains, a numeric value:
 - The number is interpreted as a decimal integer.
 - Numerals in the range 0 to 2 147 483 647 are syntactically recognized as numeric values.
 - When the utility description states that it accepts negative numbers as operands or option-arguments, numerals in the range -2 147 483 647 to 2 147 483 647 are syntactically recognized as numeric values.
 - Ranges greater than those listed here are allowed.

This does not mean that all numbers within the allowable range are necessarily semantically correct. A standard utility that accepts an option-argument or operand that is to be interpreted as a number, and for which a range of values smaller than that shown above is permitted by the IEEE Std. 1003.1-200x, describes that smaller range along with the description of the option-argument or operand. If an error is generated, the utility's diagnostic message shall indicate that the value is out of the supported range, not that it is syntactically incorrect.

7. Arguments or option-arguments enclosed in the '[' and ']' notation are optional and can be omitted. Portable applications shall not include the '[' and ']' symbols in data submitted to the utility.
8. Arguments separated by the '|' vertical bar notation are mutually-exclusive. Portable applications shall not include the '|' symbol in data submitted to the utility. Alternatively, mutually-exclusive options and operands may be listed with multiple

7479 synopsis lines. For example:

```
7480     utility_name -d[-a][[-c option_argument]][operand...]
7481     utility_name[-a][[-b]][operand...]
```

7482 When multiple synopsis lines are given for a utility, it is an indication that the utility has
 7483 mutually-exclusive arguments. These mutually-exclusive arguments alter the functionality
 7484 of the utility so that only certain other arguments are valid in combination with one of the
 7485 mutually-exclusive arguments. Only one of the mutually-exclusive arguments is allowed
 7486 for invocation of the utility. Unless otherwise stated in an accompanying OPTIONS
 7487 section, the relationships between arguments depicted in the SYNOPSIS sections are
 7488 mandatory requirements placed on portable applications. The use of conflicting mutually-
 7489 exclusive arguments produces undefined results, unless a utility description specifies
 7490 otherwise. When an option is shown without the '[' and ']' brackets, it means that
 7491 option is required for that version of the SYNOPSIS. However, it is not required to be the
 7492 first argument, as shown in the example above, unless otherwise stated.

7493 9. Ellipses ("...") are used to denote that one or more occurrences of an option or operand
 7494 are allowed. When an option or an operand followed by ellipses is enclosed in brackets,
 7495 zero or more options or operands can be specified. The forms:

```
7496     utility_name -f option_argument...[operand...]  

7497     utility_name [-g option_argument]...[operand...]
```

7498 indicate that multiple occurrences of the option and its option-argument preceding the
 7499 ellipses are valid, with semantics as indicated in the OPTIONS section of the utility. (See
 7500 also Guideline 11 in Section 12.2.) In the first example, each option-argument requires a
 7501 preceding `-f` and at least one `-f option_argument` must be given.

7502 10. When the synopsis line is too long to be printed on a single line in the Shell and Utilities
 7503 volume of IEEE Std. 1003.1-200x, the indented lines following the initial line are
 7504 continuation lines. An actual use of the command would appear on a single logical line.

7505 12.2 Utility Syntax Guidelines

7506 The following guidelines are established for the naming of utilities and for the specification of
 7507 options, option-arguments, and operands. The `getopt()` function in the System Interfaces volume
 7508 of IEEE Std. 1003.1-200x assists utilities in handling options and operands that conform to these
 7509 guidelines.

7510 Operands and option-arguments can contain characters not specified in the portable character
 7511 set.

7512 The guidelines are intended to provide guidance to the authors of future utilities, such as those
 7513 written specific to a local system or that are components of a larger application. Some of the
 7514 standard utilities do not conform to all of these guidelines; in those cases, the OPTIONS sections
 7515 describe the deviations.

7516 **Guideline 1:** Utility names should be between two and nine characters, inclusive.

7517 **Guideline 2:** Utility names should include lowercase letters (the **lower** character
 7518 classification) and digits only from the portable character set.

7519 **Guideline 3:** Each option name should be a single alphanumeric character (the **alnum**
 7520 character classification) from the portable character set.

- 7521 Multi-digit options are not allowed.
- 7522 **Guideline 4:** All options should be preceded by the '-' delimiter character.
- 7523 **Guideline 5:** Options without option-arguments should be accepted when grouped behind
7524 one '-' delimiter.
- 7525 **Guideline 6:** Each option and option-argument should be a separate argument, except as
7526 noted in Section 12.1 (on page 227), item (2).
- 7527 **Guideline 7:** Option-arguments should not be optional.
- 7528 **Guideline 8:** When multiple option-arguments are specified to follow a single option, they
7529 should be presented as a single argument, using commas within that
7530 argument or <blank> characters within that argument to separate them.
- 7531 **Guideline 9:** All options should precede operands on the command line.
- 7532 **Guideline 10:** The argument -- should be accepted as a delimiter indicating the end of
7533 options. Any following arguments should be treated as operands, even if they
7534 begin with the '-' character. The -- argument should not be used as an
7535 option or as an operand.
- 7536 **Guideline 11:** The order of different options relative to one another should not matter,
7537 unless the options are documented as mutually-exclusive and such an option
7538 is documented to override any incompatible options preceding it. If an option
7539 that has option-arguments is repeated, the option and option-argument
7540 combinations should be interpreted in the order specified on the command
7541 line.
- 7542 **Guideline 12:** The order of operands may matter and position-related interpretations should
7543 be determined on a utility-specific basis.
- 7544 **Guideline 13:** For utilities that use operands to represent files to be opened for either reading
7545 or writing, the '-' operand should be used only to mean standard input (or
7546 standard output when it is clear from context that an output file is being
7547 specified).
- 7548 The utilities in the Shell and Utilities volume of IEEE Std. 1003.1-200x that claim conformance to
7549 these guidelines shall conform completely to these guidelines as if these guidelines contained the
7550 term "shall" instead of "should". On some systems, the utilities accept usage in violation of
7551 these guidelines for backward compatibility as well as accepting the required form.
- 7552 It is recommended that all future utilities and applications use these guidelines to enhance user
7553 portability. The fact that some historical utilities could not be changed (to avoid breaking
7554 existing applications) should not deter this future goal.

7555

7556 This chapter describes the contents of headers.

7557 Headers contain function prototypes, the definition of symbolic constants, common structures,
7558 preprocessor macros, and defined types. Each function in the System Interfaces volume of
7559 IEEE Std. 1003.1-200x specifies the headers that an application shall include in order to use that
7560 function. In most cases, only one header is required. These headers are present on an application
7561 development system; they need not be present on the target execution system.

7562 **13.1 Format of Entries**

7563 The entries in this chapter are based on a common format as follows. The only sections relating
7564 to conformance are the SYNOPSIS and DESCRIPTION.

7565 **NAME**

7566 This section gives the name or names of the entry and briefly states its purpose.

7567 **SYNOPSIS**

7568 This section summarizes the use of the entry being described.

7569 **DESCRIPTION**

7570 This section describes the functionality of the header.

7571 **APPLICATION USAGE**

7572 This section is non-normative.

7573 This section gives warnings and advice to application writers about the entry. In the
7574 event of conflict between warnings and advice and a normative part of this volume of
7575 IEEE Std. 1003.1-200x, the normative material is to be taken as correct.

7576 **RATIONALE**

7577 This section is non-normative.

7578 This section contains historical information concerning the contents of this volume of
7579 IEEE Std. 1003.1-200x and why features were included or discarded by the standard
7580 developers.

7581 **FUTURE DIRECTIONS**

7582 This section is non-normative.

7583 This section provides comments which should be used as a guide to current thinking;
7584 there is not necessarily a commitment to adopt these future directions.

7585 **SEE ALSO**

7586 This section is non-normative.

7587 This section gives references to related information.

7588 **CHANGE HISTORY**

7589 This section is non-normative.

7590 This section shows the derivation of the entry and any significant changes that have
7591 been made to it.

7592 **NAME**7593 aio.h — asynchronous input and output (**REALTIME**)7594 **SYNOPSIS**

7595 AIO #include <aio.h>

7596

7597 **DESCRIPTION**7598 The <aio.h> header shall define the **aio** structure which shall include at least the following
7599 members:

7600	int	aio_fildes	File descriptor.
7601	off_t	aio_offset	File offset.
7602	volatile void *	aio_buf	Location of buffer.
7603	size_t	aio_nbytes	Length of transfer.
7604	int	aio_reqprio	Request priority offset.
7605	struct sigevent	aio_sigevent	Signal number and value.
7606	int	aio_lio_opcode	Operation to be performed.

7607 This header shall also include the following constants:

7608 **AIO_CANCELED** A return value indicating that all requested operations have been
7609 canceled.7610 **AIO_NOTCANCELED**7611 A return value indicating that some of the requested operations could not
7612 be canceled since they are in progress.7613 **AIO_ALLDONE** A return value indicating that none of the requested operations could be
7614 canceled since they are already complete.7615 **LIO_WAIT** A *lio_listio()* synchronization operation indicating that the calling thread
7616 is to suspend until the *lio_listio()* operation is complete.7617 **LIO_NOWAIT** A *lio_listio()* synchronization operation indicating that the calling thread
7618 is to continue execution while the *lio_listio()* operation is being
7619 performed, and no notification is given when the operation is complete.7620 **LIO_READ** A *lio_listio()* element operation option requesting a read.7621 **LIO_WRITE** A *lio_listio()* element operation option requesting a write.7622 **LIO_NOP** A *lio_listio()* element operation option indicating that no transfer is
7623 requested.7624 The following shall be declared as functions and may also be declared as macros. Function
7625 prototypes shall be provided for use with an ISO C standard compiler.

```

7626 int aio_cancel(int, struct aiocb *);
7627 int aio_error(const struct aiocb *);
7628 int aio_fsync(int, struct aiocb *);
7629 int aio_read(struct aiocb *);
7630 ssize_t aio_return(struct aiocb *);
7631 int aio_suspend(const struct aiocb *const[], int,
7632 const struct timespec *);
7633 int aio_write(struct aiocb *);
7634 int lio_listio(int, struct aiocb *restrict const[restrict], int,
7635 struct sigevent *restrict);

```

7636 Inclusion of the <aio.h> header may make visible symbols defined in the headers <fcntl.h>
7637 <signal.h>, <sys/types.h>, and <time.h>.

7638 **APPLICATION USAGE**

7639 None.

7640 **RATIONALE**

7641 None.

7642 **FUTURE DIRECTIONS**

7643 None.

7644 **SEE ALSO**

7645 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>, the System Interfaces volume of
7646 IEEE Std. 1003.1-200x, *fsync()*, *lseek()*, *read()*, *write()*

7647 **CHANGE HISTORY**

7648 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

7649 **Issue 6**

7650 The <aio.h> header is marked as part of the Asynchronous Input and Output option. |

7651 The description of the constants is expanded. |

7652 The **restrict** keyword is added to the prototype for *lio_listio()*. |

7653 **NAME**

7654 arpa/inet.h — definitions for internet operations

7655 **SYNOPSIS**

7656 #include <arpa/inet.h>

7657 **DESCRIPTION**7658 The **in_port_t** and **in_addr_t** types shall be defined as described in <netinet/in.h>.7659 The **in_addr** structure shall be defined as described in <netinet/in.h>.7660 The INET_ADDRSTRLEN and INET6_ADDRSTRLEN macros shall be defined as described in
7661 <netinet/in.h>.7662 The following shall be declared as functions, defined as macros, or both. If functions are
7663 declared, function prototypes shall be provided for use with an ISO C standard compiler.

7664 uint32_t htonl(uint32_t);

7665 uint16_t htons(uint16_t);

7666 uint32_t ntohl(uint32_t);

7667 uint16_t ntohs(uint16_t);

7668 The **uint32_t** and **uint16_t** types shall be defined as described in <inttypes.h>.7669 The following shall be declared as functions, and may also be defined as macros. Function
7670 prototypes shall be provided for use with an ISO C standard compiler.

7671 in_addr_t inet_addr(const char *);

7672 in_addr_t inet_lnaof(struct in_addr);

7673 struct in_addr inet_makeaddr(in_addr_t, in_addr_t);

7674 in_addr_t inet_netof(struct in_addr);

7675 in_addr_t inet_network(const char *);

7676 char *inet_ntoa(struct in_addr);

7677 IP6 const char *inet_ntop(int, const void *restrict, char *restrict,
7678 socklen_t);

7679 int inet_pton(int, const char *restrict, void *restrict);

7680

7681 Inclusion of the <arpa/inet.h> header may also make visible all symbols from <netinet/in.h>
7682 and <inttypes.h>.7683 **APPLICATION USAGE**

7684 None.

7685 **RATIONALE**

7686 None.

7687 **FUTURE DIRECTIONS**

7688 None.

7689 **SEE ALSO**7690 <netinet/in.h>, <inttypes.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *htonl()*,
7691 *inet_addr()*7692 **CHANGE HISTORY**

7693 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7694 The **restrict** keyword is added to the prototypes for *inet_ntop()* and *inet_pton()*.

7695 **NAME**

7696 assert.h — verify program assertion

7697 **SYNOPSIS**

7698 #include <assert.h>

7699 **DESCRIPTION**

7700 cx The functionality described on this reference page extends the ISO C standard. Applications
7701 shall define the appropriate feature test macro (see the System Interfaces volume of
7702 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
7703 symbols in this header.

7704 The <assert.h> header shall define the *assert()* macro. It refers to the macro NDEBUG which is
7705 not defined in the header. If NDEBUG is defined as a macro name before the inclusion of this
7706 header, the *assert()* macro is defined simply as:

7707 #define assert(ignore)((void) 0)

7708 Otherwise, the macro behaves as described in *assert()*.

7709 The *assert()* macro is redefined according to the current state of NDEBUG each time <assert.h>
7710 is included.

7711 The *assert()* macro is implemented as a macro, not as a function. If the macro definition is
7712 suppressed in order to access an actual function, the behavior is undefined.

7713 **APPLICATION USAGE**

7714 None.

7715 **RATIONALE**

7716 None.

7717 **FUTURE DIRECTIONS**

7718 None.

7719 **SEE ALSO**7720 The System Interfaces volume of IEEE Std. 1003.1-200x, *assert()*7721 **CHANGE HISTORY**

7722 First released in Issue 1. Derived from Issue 1 of the SVID.

7723 **Issue 6**

7724 The definition of the *assert()* macro is changed for alignment with the ISO/IEC 9899:1999
7725 standard.

7726 **NAME**7727 `complex.h` — complex arithmetic7728 **SYNOPSIS**7729 `#include <complex.h>`7730 **DESCRIPTION**

7731 `CX` The functionality described on this reference page extends the ISO C standard. Applications
 7732 shall define the appropriate feature test macro (see the System Interfaces volume of
 7733 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 7734 symbols in this header.

7735 The **<complex.h>** header shall define the following constants:

7736 `complex` Expands to `_Complex`.

7737 `_Complex_I` Expands to a constant expression of type `const float _Complex`, with the value
 7738 of the imaginary unit (that is, a number such that $i^2 = -1$).

7739 `imaginary` Expands to `_Imaginary`.

7740 `_Imaginary_I` Expands to a constant expression of type `const float _Imaginary` with the value
 7741 of the imaginary unit.

7742 `I` Expands to either `_Imaginary_I` or `_Complex_I`. If `_Imaginary_I` is not defined, `I`
 7743 expands to `_Complex_I`.

7744 The constants `imaginary` and `_Imaginary_I` shall be defined if the implementation supports
 7745 imaginary types.

7746 An application may undefine and then, perhaps, redefine the `complex`, `imaginary`, and `I` constants.

7747 The following shall be declared as functions and may also be defined as macros. Function
 7748 prototypes shall be provided for use with an ISO C standard compiler.

```

7749 double      complex cacos(double complex);
7750 float       complex cacosf(float complex);
7751 long double complex cacosl(long double complex);
7752 double      complex casin(double complex);
7753 float       complex casinf(float complex);
7754 long double complex casinl(long double complex);
7755 double      complex catan(double complex);
7756 float       complex catanf(float complex);
7757 long double complex catanl(long double complex);
7758 double      complex ccos(double complex);
7759 float       complex ccosf(float complex);
7760 long double complex ccosl(long double complex);
7761 double      complex csin(double complex);
7762 float       complex csinf(float complex);
7763 long double complex csinl(long double complex);
7764 double      complex ctan(double complex);
7765 float       complex ctanf(float complex);
7766 long double complex ctanl(long double complex);
7767 double      complex cacosh(double complex);
7768 float       complex cacoshf(float complex);
7769 long double complex cacoshl(long double complex);
7770 double      complex casinh(double complex);
7771 float       complex casinhf(float complex);

```



```
7772     long double  complex casinhl(long double complex);
7773     double       complex catanh(double complex);
7774     float        complex catanhf(float complex);
7775     long double  complex catanhl(long double complex);
7776     double       complex ccosh(double complex);
7777     float        complex ccoshf(float complex);
7778     long double  complex ccoshl(long double complex);
7779     double       complex csinh(double complex);
7780     float        complex csinhf(float complex);
7781     long double  complex csinhl(long double complex);
7782     double       complex catanh(double complex);
7783     float        complex catanhf(float complex);
7784     long double  complex catanhl(long double complex);
7785     double       complex cexp(double complex);
7786     float        complex cexpf(float complex);
7787     long double  complex cexpl(long double complex);
7788     double       complex clog(double complex);
7789     float        complex clogf(float complex);
7790     long double  complex clogl(long double complex);
7791     double       complex cabs(double complex);
7792     float        complex cabsf(float complex);
7793     long double  complex cabsl(long double complex);
7794     double       complex cpow(double complex, double complex);
7795     float        complex cpowf(float complex, float complex);
7796     long double  complex cpowl(long double complex, long double complex);
7797     double       complex csqrt(double complex);
7798     float        complex csqrtf(float complex);
7799     long double  complex csqrtl(long double complex);
7800     double       complex carg(double complex);
7801     float        complex cargf(float complex);
7802     long double  complex cargl(long double complex);
7803     double       complex cimag(double complex);
7804     float        complex cimagf(float complex);
7805     long double  complex cimagl(long double complex);
7806     double       complex conj(double complex);
7807     float        complex conjf(float complex);
7808     long double  complex conjl(long double complex);
7809     double       complex cproj(double complex);
7810     float        complex cprojf(float complex);
7811     long double  complex cprojl(long double complex);
7812     double       complex creal(double complex);
7813     float        complex crealf(float complex);
7814     long double  complex creall(long double complex);
```

7815 **APPLICATION USAGE**

7816 Values are interpreted as radians, not degrees. An implementation may set *errno*, but is not
7817 required to.

7818 Some of the complex arithmetic functions have branch cuts, across which the function is
7819 discontinuous. For implementations with a signed zero (including all IEC 60559:1989 standard
7820 implementations), the sign of zero distinguishes one side of a cut from another so the function is
7821 continuous (except for format limitations) as the cut is approached from either side. For
7822 example, for the square root function, which has a branch cut along the negative real axis, the
7823 top of the cut, with imaginary part +0, maps to the positive imaginary axis, and the bottom of
7824 the cut, with imaginary part -0, maps to the negative imaginary axis.

7825 Implementations that do not support a signed zero cannot distinguish the sides of branch cuts.
7826 These implementations shall map a cut so the function is continuous as the cut is approached
7827 coming around the finite endpoint of the cut in a counter-clockwise direction. (Branch cuts for
7828 the functions specified here have just one finite endpoint.) For example, for the square root
7829 function, coming counter-clockwise around the finite endpoint of the cut along the negative real
7830 axis approaches the cut from above, so the cut maps to the positive imaginary axis.

7831 The usual mathematical formulas for complex multiply, divide, and absolute value are
7832 problematic because of their treatment of infinities and because of undue overflow and
7833 underflow. The `CX_LIMITED_RANGE` pragma can be used to inform the implementation that
7834 (where the state is on) the usual mathematical formulas are acceptable. The pragma can occur
7835 either outside external declarations or preceding all explicit declarations and statements inside a
7836 compound statement. When outside external declarations, the pragma takes effect from its
7837 occurrence until another `CX_LIMITED_RANGE` pragma is encountered, or until the end of the
7838 translation unit. When inside a compound statement, the pragma takes effect from its
7839 occurrence until another `CX_LIMITED_RANGE` pragma is encountered (including within a
7840 nested compound statement), or until the end of the compound statement; at the end of a
7841 compound statement the state for the pragma is restored to its condition just before the
7842 compound statement. If this pragma is used in any other context, the behavior is undefined. The
7843 default state for the pragma is off.

7844 **RATIONALE**

7845 The choice of *I* instead of *i* for the imaginary unit concedes to the widespread use of the
7846 identifier *i* for other purposes. The application can use a different identifier, say *j*, for the
7847 imaginary unit by following the inclusion of the `<complex.h>` header with:

```
7848 #undef I  
7849 #define j _Imaginary_I
```

7850 An *I* suffix to designate imaginary constants is not required, as multiplication by *I* provides a
7851 sufficiently convenient and more generally useful notation for imaginary terms. The
7852 corresponding real type for the imaginary unit is `float`, so that use of *I* for algorithmic or
7853 notational convenience will not result in widening types.

7854 On systems with imaginary types, the application has the ability to control whether use of the
7855 macro *I* introduces an imaginary type, by explicitly defining *I* to be `_Imaginary_I` or `_Complex_I`.
7856 Disallowing imaginary types is useful for some applications intended to run on implementations
7857 without support for such types.

7858 The macro `_Imaginary_I` provides a test for whether imaginary types are supported.

7859 The `cis()` function ($\cos(x) + I^*\sin(x)$) was considered but rejected because its implementation is
7860 easy and straightforward, even though some implementations could compute sine and cosine
7861 more efficiently in tandem.

7862 **FUTURE DIRECTIONS**

7863 The following function names and the same names suffixed with *f* or *l* are reserved for future
7864 use, and may be added to the declarations in the <complex.h> header.

7865	<i>cerf()</i>	<i>cexpm1()</i>	<i>clog2()</i>
7866	<i>cerfc()</i>	<i>clog10()</i>	<i>clgamma()</i>
7867	<i>cexp2()</i>	<i>clog1p()</i>	<i>ctgamma()</i>

7868 **SEE ALSO**

7869 The System Interfaces volume of IEEE Std. 1003.1-200x, *cabs()*, *cacos()*, *cacosh()*, *carg()*, *casin()*,
7870 *casinh()*, *catan()*, *catanh()*, *ccos()*, *ccosh()*, *cexp()*, *cimag()*, *clog()*, *conj()*, *cpow()*, *cproj()*, *creal()*,
7871 *csin()*, *csinh()*, *csqrt()*, *ctan()*, *ctanh()*

7872 **CHANGE HISTORY**

7873 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

7874 **NAME**

7875 cpio.h — cpio archive values

7876 **SYNOPSIS**

7877 xSI #include <cpio.h>

7878

7879 **DESCRIPTION**

7880 Values needed by the *c_mode* field of the *cpio* archive format are described as follows:

7881

7882

	Name	Description	Value (Octal)
7883	C_IRUSR	Read by owner.	0000400
7884	C_IWUSR	Write by owner.	0000200
7885	C_IXUSR	Execute by owner.	0000100
7886	C_IRGRP	Read by group.	0000040
7887	C_IWGRP	Write by group.	0000020
7888	C_IXGRP	Execute by group.	0000010
7889	C_IROTH	Read by others.	0000004
7890	C_IWOTH	Write by others.	0000002
7891	C_IXOTH	Execute by others.	0000001
7892	C_ISUID	Set user ID.	0004000
7893	C_ISGID	Set group ID.	0002000
7894	C_ISVTX	On directories, restricted deletion flag.	0001000
7895	C_ISDIR	Directory.	0040000
7896	C_ISFIFO	FIFO.	0010000
7897	C_ISREG	Regular file.	0100000
7898	C_ISBLK	Block special.	0060000
7899	C_ISCHR	Character special.	0020000
7900	C_ISCTG	Reserved.	0110000
7901	C_ISLNK	Symbolic link.	0120000
7902	C_ISSOCK	Socket.	0140000

7903 The header shall define the symbolic constant:

7904 MAGIC "070707"

7905 **APPLICATION USAGE**

7906 None.

7907 **RATIONALE**

7908 None.

7909 **FUTURE DIRECTIONS**

7910 None.

7911 **SEE ALSO**

7912 The Shell and Utilities volume of IEEE Std. 1003.1-200x, *pax*

7913 **CHANGE HISTORY**

7914 First released in Issue 3 of the Headers Interface, Issue 3 specification. Derived from the

7915 POSIX.1-1988 standard.

7916 **Issue 4, Version 2**

7917 Descriptions for C_ISLNK and C_ISSOCK are provided; formerly, these were listed as

7918 "Reserved".

7919 **Issue 6**

7920

7921

The SEE ALSO is updated to refer to *pax*, since the *cpio* utility is not included in the Shell and Utilities volume of IEEE Std. 1003.1-200x.

7922 **NAME**

7923 ctype.h — character types

7924 **SYNOPSIS**

7925 #include <ctype.h>

7926 **DESCRIPTION**

7927 **CX** The functionality described on this reference page extends the ISO C standard. Applications
7928 shall define the appropriate feature test macro (see the System Interfaces volume of
7929 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
7930 symbols in this header.

7931 The **<ctype.h>** header shall declare the following as functions and may also define them as
7932 macros. Function prototypes shall be provided for use with an ISO C standard compiler.

7933 int isalnum(int);

7934 int isalpha(int);

7935 **XSI** int isascii(int);

7936 int isblank(int);

7937 int iscntrl(int);

7938 int isdigit(int);

7939 int isgraph(int);

7940 int islower(int);

7941 int isprint(int);

7942 int ispunct(int);

7943 int isspace(int);

7944 int isupper(int);

7945 int isxdigit(int);

7946 **XSI** int toascii(int);

7947 int tolower(int);

7948 int toupper(int);

7949 The following are defined as macros:

7950 **XSI** int _toupper(int);

7951 int _tolower(int);

7952

7953 **APPLICATION USAGE**

7954 None.

7955 **RATIONALE**

7956 None.

7957 **FUTURE DIRECTIONS**

7958 None.

7959 **SEE ALSO**

7960 **<locale.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, *isalnum()*, *isalpha()*, *isascii()*,
7961 *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *mblen()*,
7962 *mbstowcs()*, *mbtowc()*, *setlocale()*, *toascii()*, *tolower()*, *_tolower()*, *toupper()*, *_toupper()*, *wcstombs()*,
7963 *wctomb()*

7964 **CHANGE HISTORY**

7965 First released in Issue 1. Derived from Issue 1 of the SVID.

7966 **Issue 4**

7967 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 7968
 - The function declarations in this header are expanded to full ISO C standard prototypes.

7969 **Issue 6**

7970 Extensions beyond the ISO C standard are now marked.

7971 **NAME**7972 `dirent.h` — format of directory entries7973 **SYNOPSIS**7974 `#include <dirent.h>`7975 **DESCRIPTION**

7976 The internal format of directories is unspecified.

7977 The **<dirent.h>** header shall define the following data type through **typedef**:7978 **DIR** A type representing a directory stream.7979 It shall also define the structure **dirent** which shall include the following members:7980 XSI `ino_t d_ino` File serial number.7981 `char d_name[]` Name of entry.7982 XSI The type **ino_t** shall be defined as described in **<sys/types.h>**.7983 The character array *d_name* is of unspecified size, but the number of bytes preceding the
7984 terminating null byte does not exceed {NAME_MAX}.7985 The following shall be declared as functions and may also be defined as macros. Function
7986 prototypes shall be provided for use with an ISO C standard compiler.7987 `int closedir(DIR *);`7988 `DIR *opendir(const char *);`7989 `struct dirent *readdir(DIR *);`7990 TSF `int readdir_r(DIR *restrict, struct dirent *restrict,
7991 struct dirent **restrict);`7992 `void rewinddir(DIR *);`7993 XSI `void seekdir(DIR *, long);`7994 `long telldir(DIR *);`

7995

7996 **APPLICATION USAGE**

7997 None.

7998 **RATIONALE**7999 Information similar to that in the **<dirent.h>** header is contained in a file **<sys/dir.h>** in 4.2 BSD
8000 and 4.3 BSD. The equivalent in these implementations of **struct dirent** from this volume of
8001 IEEE Std. 1003.1-200x is **struct direct**. The file name was changed because the name **<sys/dir.h>**
8002 was also used in earlier implementations to refer to definitions related to the older access
8003 method; this produced name conflicts. The name of the structure was changed because this
8004 volume of IEEE Std. 1003.1-200x does not completely define what is in the structure, so it could
8005 be different on some implementations from **struct direct**.8006 The name of an array of **char** of an unspecified size should not be used as an **lvalue**. Use of:8007 `sizeof(d_name)`

8008 is incorrect; use:

8009 `strlen(d_name)`

8010 instead.

8011 The array of **char** *d_name* is not a fixed size. Implementations may need to declare **struct dirent**
8012 with an array size for *d_name* of 1, but the actual number of characters provided matches (or
8013 only slightly exceeds) the length of the file name.

8014 **FUTURE DIRECTIONS**

8015 None.

8016 **SEE ALSO**

8017 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *closedir()*, *opendir()*,
8018 *readdir()*, *readdir_r()*, *rewinddir()*, *seekdir()*, *telldir()*

8019 **CHANGE HISTORY**

8020 First released in Issue 2.

8021 **Issue 4**

8022 Reference to type **ino_t** is marked as an extension, as are references to the *seekdir()* and *telldir()*
8023 functions.

8024 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 8025 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 8026 • A statement is added to the DESCRIPTION indicating that the internal format of directories
8027 is unspecified. Also in the description of the *d_name* field, the text is changed to indicate
8028 “bytes” rather than (possibly multi-byte) “characters”.

8029 **Issue 5**

8030 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8031 **Issue 6**

8032 The Open Group corrigenda item U026/7 has been applied, correcting the prototype for
8033 *readdir_r()*.

8034 The **restrict** keyword is added to the prototype for *readdir_r()*.

8035 **NAME**8036 `dlfcn.h` — dynamic linking8037 **SYNOPSIS**8038 XSI `#include <dlfcn.h>`

8039

8040 **DESCRIPTION**8041 The `<dlfcn.h>` header shall define at least the following macros for use in the construction of a
8042 `dlopen()` *mode* argument:8043 `RTLD_LAZY` Relocations are performed at an implementation-defined time.8044 `RTLD_NOW` Relocations are performed when the object is loaded.8045 `RTLD_GLOBAL` All symbols are available for relocation processing of other modules.8046 `RTLD_LOCAL` All symbols are not made available for relocation processing by other
8047 modules.8048 The `<dlfcn.h>` header shall declare the following functions which may also be defined as
8049 macros. Function prototypes shall be provided for use with an ISO C standard compiler.8050 `int dlclose(void *);`8051 `char *dlerror(void);`8052 `void *dlopen(const char *, int);`8053 `void *dlsym(void *restrict, const char *restrict);`8054 **APPLICATION USAGE**

8055 None.

8056 **RATIONALE**

8057 None.

8058 **FUTURE DIRECTIONS**

8059 None.

8060 **SEE ALSO**8061 The System Interfaces volume of IEEE Std. 1003.1-200x, `dlopen()`, `dlclose()`, `dlsym()`, `dlerror()`8062 **CHANGE HISTORY**

8063 First released in Issue 5.

8064 **Issue 6**8065 The `restrict` keyword is added to the prototype for `dlsym()`.

8066 **NAME**

8067 errno.h — system error numbers

8068 **SYNOPSIS**

8069 #include <errno.h>

8070 **DESCRIPTION**

8071 **cx** The functionality described on this reference page extends the ISO C standard. Applications
 8072 shall define the appropriate feature test macro (see the System Interfaces volume of
 8073 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 8074 symbols in this header.

8075 The <errno.h> header provides a declaration for *errno* and gives non-zero values for the
 8076 following symbolic constants. Their values are unique except as noted below:

8077	[E2BIG]	Argument list too long.
8078	[EACCES]	Permission denied.
8079	[EADDRINUSE]	Address in use.
8080	[EADDRNOTAVAIL]	Address not available.
8081	[EAFNOSUPPORT]	Address family not supported.
8082	[EAGAIN]	Resource unavailable, try again (may be the same value as
8083		[EWOULDBLOCK]).
8084	[EALREADY]	Connection already in progress.
8085	[EBADF]	Bad file descriptor.
8086	[EBADMSG]	Bad message.
8087	[EBUSY]	Device or resource busy.
8088	[ECANCELED]	Operation canceled.
8089	[ECHILD]	No child processes.
8090	[ECONNABORTED]	Connection aborted.
8091	[ECONNREFUSED]	Connection refused.
8092	[ECONNRESET]	Connection reset.
8093	[EDEADLK]	Resource deadlock would occur.
8094	[EDESTADDRREQ]	Destination address required.
8095	[EDOM]	Mathematics argument out of domain of function.
8096	[EDQUOT]	Reserved.
8097	[EEXIST]	File exists.
8098	[EFAULT]	Bad address.
8099	[EFBIG]	File too large.
8100	[EHOSTUNREACH]	Host is unreachable.
8101	[EIDRM]	Identifier removed.
8102	[EILSEQ]	Illegal byte sequence.

8103	[EINPROGRESS]	Operation in progress.	
8104	[EINTR]	Interrupted function.	
8105	[EINVAL]	Invalid argument.	
8106	[EIO]	I/O error.	
8107	[EISCONN]	Socket is connected.	
8108	[EISDIR]	Is a directory.	
8109	[ELOOP]	Too many levels of symbolic links.	
8110	[EMFILE]	Too many open files.	
8111	[EMLINK]	Too many links.	
8112	[EMSGSIZE]	Message too large.	
8113	[EMULTIHOP]	Reserved.	
8114	[ENAMETOOLONG]	File name too long.	
8115	[ENETDOWN]	Network is down.	
8116	[ENETUNREACH]	Network unreachable.	
8117	[ENFILE]	Too many files open in system.	
8118	[ENOBUFS]	No buffer space available.	
8119 XSI	[ENODATA]	No message is available on the STREAM head read queue.	
8120	[ENODEV]	No such device.	
8121	[ENOENT]	No such file or directory.	
8122	[ENOEXEC]	Executable file format error.	
8123	[ENOLCK]	No locks available.	
8124	[ENOLINK]	Reserved.	
8125	[ENOMEM]	Not enough space.	
8126	[ENOMSG]	No message of the desired type.	
8127	[ENOPROTOPT]	Protocol not available.	
8128	[ENOSPC]	No space left on device.	
8129 XSI	[ENOSR]	No STREAM resources.	
8130 XSI	[ENOSTR]	Not a STREAM.	
8131	[ENOSYS]	Function not supported.	
8132	[ENOTCONN]	The socket is not connected.	
8133	[ENOTDIR]	Not a directory.	
8134	[ENOTEMPTY]	Directory not empty.	
8135	[ENOTSOCK]	Not a socket.	
8136	[ENOTSUP]	Not supported.	

8137	[ENOTTY]	Inappropriate I/O control operation.	
8138	[ENXIO]	No such device or address.	
8139	[EOPNOTSUPP]	Operation not supported on socket.	
8140	[EOVERFLOW]	Value too large to be stored in data type.	
8141	[EPERM]	Operation not permitted.	
8142	[EPIPE]	Broken pipe.	
8143	[EPROTO]	Protocol error.	
8144	[EPROTONOSUPPORT]		
8145		Protocol not supported.	
8146	[EPROTOTYPE]	Socket type not supported.	
8147	[ERANGE]	Result too large.	
8148	[EROFS]	Read-only file system.	
8149	[ESPIPE]	Invalid seek.	
8150	[ESRCH]	No such process.	
8151	[ESTALE]	Reserved.	
8152	XS1 [ETIME]	Stream <i>ioctl()</i> timeout.	
8153	[ETIMEDOUT]	Connection timed out.	
8154	[ETXTBSY]	Text file busy.	
8155	[EWOULDBLOCK]	Operation would block (may be the same value as [EAGAIN]).	
8156	[EXDEV]	Cross-device link.	
8157	APPLICATION USAGE		
8158	Additional error numbers may be defined on conforming systems; see the System Interfaces		
8159	volume of IEEE Std. 1003.1-200x.		
8160	RATIONALE		
8161	None.		
8162	FUTURE DIRECTIONS		
8163	None.		
8164	SEE ALSO		
8165	The System Interfaces volume of IEEE Std. 1003.1-200x, Section 2.3, Error Numbers		
8166	CHANGE HISTORY		
8167	First released in Issue 1. Derived from Issue 1 of the SVID.		
8168	Issue 4		
8169	The [EILSEQ] error is added and marked as an EX interface.		
8170	The [ENOTBLK] error is withdrawn.		
8171	Issue 4, Version 2		
8172	The [EADDRINUSE], [EADDRNOTAVAIL], [EAFNOSUPPORT], [EALREADY], [EBADMSG],		
8173	[ECONNABORTED], [ECONNREFUSED], [ECONNRESET], [EDESTADDRREQ], [EDQUOT],		
8174	[EHOSTUNREACH], [EINPROGRESS], [EISCONN], [ELOOP], [EMSGSIZE], [EMULTIHOP],		
8175	[ENETDOWN], [ENETUNREACH], [ENOBUFS], [ENODATA], [ENOLINK],		

8176 [ENOPROTOOPT], [ENOSR], [ENOSTR], [ENOTCONN], [ENOTSOCK], [EOPNOTSUPP],
8177 [EOVERFLOW], [EPROTO], [EPROTONOSUPPORT], [EPROTOTYPE], [ESTALE], [ETIME],
8178 [ETIMEDOUT], and [EWOULDBLOCK] errors are added in the UX context.

8179 **Issue 5**

8180 Updated for alignment with the POSIX Realtime Extension.

8181 **Issue 6**

8182 The following new requirements on POSIX implementations derive from alignment with the
8183 Single UNIX Specification:

- 8184 • The majority of the error conditions previously marked as extensions are now mandatory,
8185 except for the STREAMS-related error conditions.

8186 NAME

8187 fcntl.h — file control options

8188 SYNOPSIS

8189 #include <fcntl.h>

8190 DESCRIPTION

8191 The <fcntl.h> header shall define the following requests and arguments for use by the functions
8192 *fcntl()* and *open()*.8193 Values for *cmd* used by *fcntl()* (the following values are unique) are as follows:

8194 F_DUPFD Duplicate file descriptor.

8195 F_GETFD Get file descriptor flags.

8196 F_SETFD Set file descriptor flags.

8197 F_GETFL Get file status flags and file access modes.

8198 F_SETFL Set file status flags.

8199 F_GETLK Get record locking information.

8200 F_SETLK Set record locking information.

8201 F_SETLKW Set record locking information; wait if blocked.

8202 F_GETOWN Get process or process group ID to receive SIGURG signals.

8203 F_SETOWN Set process or process group ID to receive SIGURG signals.

8204 File descriptor flags used for *fcntl()* are as follows:8205 FD_CLOEXEC Close the file descriptor upon execution of an *exec* family function.8206 Values for *l_type* used for record locking with *fcntl()* (the following values are unique) are as
8207 follows:

8208 F_RDLCK Shared or read lock.

8209 F_UNLCK Unlock.

8210 F_WRLCK Exclusive or write lock.

8211 XSI The values used for *l_whence*, {SEEK_SET}, {SEEK_CUR}, and {SEEK_END} shall be defined as
8212 described in <unistd.h>.8213 The following four sets of values for *oflag* used by *open()* shall be bitwise-distinct:

8214 O_CREAT Create file if it does not exist.

8215 O_EXCL Exclusive use flag.

8216 O_NOCTTY Do not assign controlling terminal.

8217 O_TRUNC Truncate flag.

8218 File status flags used for *open()* and *fcntl()* are as follows:

8219 O_APPEND Set append mode.

8220 SIO O_DSYNC Write according to synchronized I/O data integrity completion.

8221 O_NONBLOCK Non-blocking mode.

8222 SIO O_RSYNC Synchronized read I/O operations.

8223 O_SYNC Write according to synchronized I/O file integrity completion.

8224 Mask for use with file access modes is as follows:

8225 O_ACCMODE Mask for file access modes.

8226 File access modes used for *open()* and *fcntl()* are as follows:

8227 O_RDONLY Open for reading only.

8228 O_RDWR Open for reading and writing.

8229 O_WRONLY Open for writing only.

8230 XSI The symbolic names for file modes for use as values of **mode_t** shall be defined as described in
8231 <sys/stat.h>.

8232 ADV Values for *advice* used by *posix_fadvise()* are as follows:

8233 POSIX_FADV_NORMAL
8234 The application has no advice to give on its behavior with respect to the specified data. It is
8235 the default characteristic if no advice is given for an open file.

8236 POSIX_FADV_SEQUENTIAL
8237 The application expects to access the specified data sequentially from lower offsets to
8238 higher offsets.

8239 POSIX_FADV_RANDOM
8240 The application expects to access the specified data in a random order.

8241 POSIX_FADV_WILLNEED
8242 The application expects to access the specified data in the near future.

8243 POSIX_FADV_DONTNEED
8244 The application expects that it will not access the specified data in the near future.

8245 POSIX_FADV_NOREUSE
8246 The application expects to access the specified data once and then not reuse it thereafter.
8247

8248 The structure **flock** describes a file lock. It shall include the following members:

8249 short l_type Type of lock; F_RDLCK, F_WRLCK, F_UNLCK.

8250 short l_whence Flag for starting offset.

8251 off_t l_start Relative offset in bytes.

8252 off_t l_len Size; if 0 then until EOF.

8253 pid_t l_pid Process ID of the process holding the lock; returned with F_GETLK.

8254 The **mode_t**, **off_t**, and **pid_t** types shall be defined as described in <sys/types.h>.

8255 The following shall be declared as functions and may also be defined as macros. Function
8256 prototypes shall be provided for use with an ISO C standard compiler.

8257 int creat(const char *, mode_t);

8258 int fcntl(int, int, ...);

8259 int open(const char *, int, ...);

8260 ADV int posix_fadvise(int, off_t, size_t, int);

8261 int posix_fallocate(int, off_t, size_t);

8262

- 8263 XSI Inclusion of the <fcntl.h> header may also make visible all symbols from <sys/stat.h> and
8264 <unistd.h>.
- 8265 **APPLICATION USAGE**
8266 None.
- 8267 **RATIONALE**
8268 None.
- 8269 **FUTURE DIRECTIONS**
8270 None.
- 8271 **SEE ALSO**
8272 <sys/stat.h>, <sys/types.h>, <unistd.h>, the System Interfaces volume of IEEE Std. 1003.1-200x,
8273 *creat()*, *exec()*, *fcntl()*, *open()*, *posix_fadvise()*, *posix_fallocate()*, *posix_madvise()*
- 8274 **CHANGE HISTORY**
8275 First released in Issue 1. Derived from Issue 1 of the SVID.
- 8276 **Issue 4**
8277 A reference to <unistd.h> is added for the definition of *l_whence*, {SEEK_SET}, {SEEK_CUR}, and
8278 {SEEK_END}, and marked as an extension.
8279 A reference to <sys/stat.h> is added for the symbolic names of file modes used as values of
8280 **mode_t**, and marked as an extension.
8281 A reference to <sys/types.h> is added for the definition of **mode_t**, **off_t**, and **pid_t**, and marked
8282 as an extension.
8283 A warning is added indicating that inclusion of <fcntl.h> may also make visible all symbols
8284 from <sys/stat.h> and <unistd.h>. This is marked as an extension.
8285 The following change is incorporated for alignment with the ISO POSIX-1 standard:
8286
 - The function declarations in this header are expanded to full ISO C standard prototypes.
- 8287 **Issue 5**
8288 The DESCRIPTION is updated for alignment with POSIX Realtime Extension.
- 8289 **Issue 6**
8290 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:
8291
 - O_DSYNC and O_RSYNC are marked as part of the Synchronized Input and Output option.
8292 The following new requirements on POSIX implementations derive from alignment with the
8293 Single UNIX Specification:
8294
 - The definition of the **mode_t**, **off_t**, and **pid_t** types is mandated.
8295 The F_GETOWN and F_SETOWN values are added for sockets.
8296 The *posix_fadvise()*, *posix_fallocate()*, and *posix_madvise()* functions are added for alignment with
8297 IEEE Std. 1003.1d-1999.
8298 IEEE PASC Interpretation 1003.1 #102 is applied moving the prototype for *posix_madvise()* to
8299 <sys_mman.h>.

8300 **NAME**8301 `fenv.h` — floating-point environment8302 **SYNOPSIS**8303 `#include <fenv.h>`8304 **DESCRIPTION**

8305 `CX` The functionality described on this reference page extends the ISO C standard. Applications
 8306 shall define the appropriate feature test macro (see the System Interfaces volume of
 8307 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 8308 symbols in this header.

8309 The **<fenv.h>** header shall define the following data types through **typedef**:

8310 **fenv_t** Represents the entire floating-point environment. The floating-point environment
 8311 refers collectively to any floating-point status flags and control modes supported
 8312 by the implementation.

8313 **fexcept_t** Represents the floating-point status flags collectively, including any status the
 8314 implementation associates with the flags. A floating-point status flag is a system
 8315 variable whose value is set (but never cleared) when a floating-point exception is
 8316 raised, which occurs as a side effect of exceptional floating-point arithmetic to
 8317 provide auxiliary information. A floating-point control mode is a system variable
 8318 whose value may be set by the user to affect the subsequent behavior of floating-
 8319 point arithmetic.

8320 The **<fenv.h>** header shall define the following constants:8321 `FE_DIVBYZERO`8322 `FE_INEXACT`8323 `FE_INVALID`8324 `FE_OVERFLOW`8325 `FE_UNDERFLOW`

8326 These constants are defined if and only if the implementation supports
 8327 the floating-point exception by means of the floating-point functions
 8328 `fwclearexcept()`, `fegetexceptflag()`, `feraiseexcept()`, `fesetexceptflag()`, and
 8329 `fetestexcept()`. Each expands to an integer constant expression with values
 8330 such that bitwise-inclusive ORs of all combinations of the constants result
 in distinct values.

8331 `FE_ALL_EXCEPT` Simply the bitwise-inclusive OR of all floating-point exception constants
 8332 defined above.

8333 `FE_DOWNWARD`8334 `FE_TONEAREST`8335 `FE_TOWARDZERO`8336 `FE_UPWARD`

8337 These constants are defined if and only if the implementation supports
 8338 getting and setting the represented rounding direction by means of the
 8339 `fegetround()` and `fesetround()` functions. Each expands to an integer
 constant expression whose values are distinct non-negative values.

8340 `FE_DFL_ENV`

8341 Represents the floating-point environment (that is, the one installed at
 8342 program startup) and has type pointer to const-qualified **fenv_t**. It can
 8343 be used as an argument to the functions within the **<fenv.h>** header that
 manage the floating-point environment.

8344 The following shall be declared as functions and may also be defined as macros. Function
 8345 prototypes shall be provided for use with an ISO C standard compiler.

```

8346 void feclearexcept(int);
8347 void fegetexceptflag(fexcept_t *, int);
8348 void feraiseexcept(int);
8349 void fesetexceptflag(const fexcept_t *, int);
8350 int fetestexcept(int);
8351 int fegetround(void);
8352 int fesetround(int);
8353 void fegetenv(fenv_t *);
8354 int feholdexcept(fenv_t *);
8355 void fesetenv(const fenv_t *);
8356 void feupdateenv(const fenv_t *);

```

8357 APPLICATION USAGE

8358 This header is designed to support the floating-point exception status flags and directed-
8359 rounding control modes required by the IEC 60559:1989 standard, and other similar floating-
8360 point state information. Also it is designed to facilitate code portability among all systems.

8361 Certain application programming conventions support the intended model of use for the
8362 floating-point environment:

- 8363 • A function call does not alter its caller's floating-point control modes, clear its caller's
8364 floating-point status flags, nor depend on the state of its caller's floating-point status flags
8365 unless the function is so documented.
- 8366 • A function call is assumed to require default floating-point control modes, unless its
8367 documentation promises otherwise.
- 8368 • A function call is assumed to have the potential for raising floating-point exceptions, unless
8369 its documentation promises otherwise.

8370 With these conventions, an application can safely assume default floating-point control modes
8371 (or be unaware of them). The responsibilities associated with accessing the floating-point
8372 environment fall on the application that does so explicitly.

8373 Even though the rounding direction macros may expand to constants corresponding to the
8374 values of FLT_ROUNDS, they are not required to do so.

8375 The FENV_ACCESS pragma provides a means to inform the implementation when an
8376 application might access the floating-point environment to test floating-point status flags or run
8377 under non-default floating-point control modes. The pragma shall occur either outside external
8378 declarations or preceding all explicit declarations and statements inside a compound statement.
8379 When outside external declarations, the pragma takes effect from its occurrence until another
8380 FENV_ACCESS pragma is encountered, or until the end of the translation unit. When inside a
8381 compound statement, the pragma takes effect from its occurrence until another FENV_ACCESS
8382 pragma is encountered (including within a nested compound statement), or until the end of the
8383 compound statement; at the end of a compound statement the state for the pragma is restored to
8384 its condition just before the compound statement. If this pragma is used in any other context, the
8385 behavior is undefined. If part of an application tests floating-point status flags, sets floating-
8386 point control modes, or runs under non-default mode settings, but was translated with the state
8387 for the FENV_ACCESS pragma off, the behavior is undefined. The default state (on or off) for
8388 the pragma is implementation-defined. (When execution passes from a part of the application
8389 translated with FENV_ACCESS off to a part translated with FENV_ACCESS on, the state of the
8390 floating-point status flags is unspecified and the floating-point control modes have their default
8391 settings.) For example:

```

8392 #include <fenv.h>
8393 void f(double x)

```

```

8394     {
8395         #pragma STDC FENV_ACCESS ON
8396         void g(double);
8397         void h(double);
8398         /* ... */
8399         g(x + 1);
8400         h(x + 1);
8401         /* ... */
8402     }

```

8403 If the function $g()$ might depend on status flags set as a side effect of the first $x+1$, or if the
8404 second $x+1$ might depend on control modes set as a side effect of the call to function $g()$, then
8405 the application shall contain an appropriately placed invocation as follows:

```

8406     #pragma STDC FENV_ACCESS ON

```

8407 **RATIONALE**

8408 The floating-point environment as defined here includes only execution-time modes, not the
8409 myriad of possible translation-time options that can affect an application's results. Each such
8410 option's deviation from IEEE Std. 1003.1-200x should be well documented.

8411 **Dynamic Versus Static Modes**

8412 Dynamic modes are potentially problematic because:

- 8413 1. The application may have to defend against undesirable mode settings, which impose
8414 intellectual as well as time and space overhead.
- 8415 2. The translator may not know which mode settings will be in effect or which functions
8416 change them at execution time, which inhibits optimization.

8417 The ISO/IEC 9899:1999 standard addresses these problems without changing the dynamic
8418 nature of the modes.

8419 An alternate approach would have been to present a model of static modes with explicit
8420 utterances to the translator about what mode settings would be in effect. This would have
8421 avoided any uncertainty due to the global nature of dynamic modes or the dependency on
8422 unenforced conventions. However, some essentially dynamic mechanism still would have been
8423 needed in order to allow functions to inherit (honor) their caller's modes. The IEC 60559:1989
8424 standard requires dynamic rounding direction modes. For the many architectures that maintain
8425 these modes in control registers, implementation of the static model would be more costly. Also,
8426 standard C has no facility, other than pragmas, for supporting static modes.

8427 An implementation on an architecture that provides only static control of modes (for example,
8428 through opword encodings) still could support the dynamic model, by generating multiple code
8429 streams with tests of a private global variable containing the mode setting. Only modules under
8430 an enabling FENV_ACCESS pragma would need such special treatment.

8431 **Translation**

8432 An implementation is not required to provide a facility for altering the modes for translation-
8433 time arithmetic, or for making exception flags from the translation available to the executing
8434 application. The language and library provide facilities to cause floating-point operations to be
8435 done at execution time when they can be subjected to varying dynamic modes and their
8436 exceptions detected. The need does not seem sufficient to require similar facilities for translation.

8437 **The fexcept_t Type**

8438 **fexcept_t** does not have to be an integer type. Its values must be obtained by a call to
 8439 *fegetexceptflag()*, and cannot be created by logical operations from the exception macros. An
 8440 implementation might simply implement **fexcept_** as an **int** and use the representations
 8441 reflected by the exception macros, but is not required to; other representations might contain
 8442 extra information about the exceptions. **fexcept_t** might be a **struct** with a member for each
 8443 exception (that might hold the address of the first or last floating-point instruction that caused
 8444 that exception). The ISO/IEC 9899:1999 standard makes no claims about the internals of an
 8445 **fexcept_t**, and so the user cannot inspect it.

8446 **Exception and Rounding Macros**

8447 Unsupported macros are not defined in order to ensure that their use results in a translation
 8448 error. An application might explicitly define such macros to allow translation of code (perhaps
 8449 never executed) containing the macros. An unsupported exception macro should be defined to
 8450 be 0; for example:

```
8451 #ifndef FE_INEXACT
8452 #define FE_INEXACT 0
8453 #endif
```

8454 so that a bitwise-inclusive OR of macros has a reasonable effect.

8455 **Exceptions**

8456 In previous drafts of IEEE Std. 1003.1-200x, several of the exception functions returned an **int**
 8457 indicating whether the *excepts* argument represented supported exceptions. This facility was
 8458 deemed unnecessary because:

```
8459 excepts & ~FE_ALL_EXCEPT
```

8460 can be used to test invalidity of the *excepts* argument.

8461 **Rounding Precision**

8462 The IEC 60559:1989 standard floating-point standard prescribes rounding precision modes (in
 8463 addition to the rounding direction modes covered by the functions in this reference page) as a
 8464 means for systems whose results are always double or extended to mimic systems that deliver
 8465 results to narrower formats. An implementation of C can meet this goal in any of the following
 8466 ways:

- 8467 1. By supporting the evaluation method indicated by `FLT_EVAL_METHOD` equal to 0
- 8468 2. By providing pragmas or compile options to shorten results by rounding to the
 8469 IEC 60559:1989 standard single or double precision
- 8470 3. By providing functions to dynamically set and get rounding precision modes which
 8471 shorten results by rounding to the IEC 60559:1989 standard single or double precision;
 8472 recommended are functions *fesetprec()* and *fegetprec()* and macros `FE_FLTPREC`,
 8473 `FE_DBLPREC`, and `FE_LDBLPREC`, analogous to the functions and macros for the
 8474 rounding direction modes

8475 IEEE Std. 1003.1-200x does not include a portable interface for precision control because the
 8476 IEC 60559:1989 standard floating-point standard is ambivalent on whether it intends for
 8477 precision control to be dynamic (like the rounding direction modes) or static. Indeed, some
 8478 floating-point architectures provide control modes suitable for a dynamic mechanism, and
 8479 others rely on instructions to deliver single and double-format results suitable only for a static

8480 mechanism.

8481 **FUTURE DIRECTIONS**

8482 None.

8483 **SEE ALSO**

8484 The System Interfaces volume of IEEE Std. 1003.1-200x, *feclearexcept()*, *fegetenv()*,
8485 *fegetexceptflag()*, *fegetround()*, *fehldexcept()*, *feraiseexcept()*, *fesetenv()*, *fesetexceptflag()*,
8486 *fesetround()*, *fetestexcept()*, *feupdateenv()*

8487 **CHANGE HISTORY**

8488 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

8489 **NAME**

8490 float.h — floating types

8491 **SYNOPSIS**

8492 #include <float.h>

8493 **DESCRIPTION**

8494 **cx** The functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of symbols in this header.

8498 The characteristics of floating types are defined in terms of a model that describes a representation of floating-point numbers and values that provide information about an implementation's floating-point arithmetic.

8501 The following parameters are used to define the model for each floating-point type:

8502 *s* Sign (± 1).

8503 *b* Base or radix of exponent representation (an integer > 1).

8504 *e* Exponent (an integer between a minimum e_{\min} and a maximum e_{\max}).

8505 *p* Precision (the number of base-*b* digits in the significand).

8506 f_k Non-negative integers less than *b* (the significand digits).

8507 A normalized floating-point number x ($f_1 > 0$ if $x \neq 0$) is defined by the following model:

8508
$$x = s \times b^e \times \sum_{k=1}^p f_k \times b^{-k}, e_{\min} \leq e \leq e_{\max}$$

8509

8510 FLT_RADIX is a constant expression suitable for use in the #if preprocessing directives. All constants except FLT_RADIX and FLT_ROUNDSS have separate names for all three floating-point types. The floating-point model representation is provided for all macro names except FLT_ROUNDSS.

8514 The rounding mode for floating-point addition is characterized by the value of FLT_ROUNDSS:

8515 -1 Indeterminable.

8516 0 Toward 0.0.

8517 1 To nearest.

8518 2 Toward positive infinity.

8519 3 Toward negative infinity.

8520 All other values for FLT_ROUNDSS characterize implementation-defined rounding behavior.

8521 The values of operations with floating operands and values subject to the usual arithmetic conversions and of floating constants are evaluated to a format whose range and precision may be greater than required by the type. The use of evaluation formats is characterized by the implementation-defined value of FLT_EVAL_METHOD:

8525 -1 Indeterminable.

8526 0 Evaluate all operations and constants just to the range and precision of the type.

8527 1 Evaluate operations and constants of type **float** and **double** to the range and precision of the **double** type, evaluate **long double** operations and constants to the range and precision of the **long double** type.

8530 2 Evaluate all operations and constants to the range and precision of the **long double** type. |
8531 All other negative values for FLT_EVAL_METHOD characterize implementation-defined |
8532 behavior. |
8533 The macro names given in the following list are defined as expressions with values that are |
8534 equal or greater in magnitude (absolute value) to those shown, with the same sign. |

Name	Description	Value
FLT_RADIX	Radix of exponent representation, b .	2
FLT_MANT_DIG DBL_MANT_DIG LDBL_MANT_DIG	Number of base-FLT_RADIX digits in the floating-point significand, p . † †	†
DECIMAL_DIG	Number of decimal digits, n , such that any floating-point number in the widest supported floating type with p_{max} radix b digits can be rounded to a floating-point number with n decimal digits and back again without change to the value. Notes to Reviewers <i>This section with side shading will not appear in the final copy. - Ed.</i> D3, XSH, ERN 146 requires a new equation to be inserted here. However, none of the equations in float.h match the C99 style. This needs looking at again.	10
FLT_DIG DBL_DIG LDBL_DIG	Number of decimal digits, q , such that any floating-point number with q decimal digits can be rounded into a floating-point number with p radix b digits and back again without change to the q decimal digits, $\left\lfloor (p-1) \times \log_{10} b \right\rfloor + \begin{cases} 1 & \text{if } b \text{ is a power of } 10 \\ 0 & \text{otherwise} \end{cases}$	6 10 10
FLT_MIN_EXP DBL_MIN_EXP LDBL_MIN_EXP	Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a normalized floating-point number, e_{min} † †	†
FLT_MIN_10_EXP DBL_MIN_10_EXP LDBL_MIN_10_EXP	Minimum negative integer such that 10 raised to that power is in the range of normalized floating-point numbers, $\left\lfloor \log_{10} b^{e_{min}^{-1}} \right\rfloor$	-37 -37 -37
FLT_MAX_EXP DBL_MAX_EXP LDBL_MAX_EXP	Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable finite floating-point number, e_{max} † †	†

8535	FLT_MAX_10_EXP	Maximum integer such that 10 raised to that power is in the range of representable finite floating-point numbers,	37
		$\left\lfloor \log_{10}((1 - b^{-p}) \times b^{e_{\max}}) \right\rfloor$	
	DBL_MAX_10_EXP		37
	LDBL_MAX_10_EXP		37

8536 † Implementation-defined values.

8537 The macro names given in the following list are defined as expressions with values that are
8538 equal to or greater than those shown.

8539	FLT_MAX	Maximum representable finite floating-point number,	1E+37
		$(1 - b^{-p}) \times b^{e_{\max}}$	
	DBL_MAX		1E+37
	LDBL_MAX		1E+37

8540 The macro names given in the following list are defined as expressions with values that are
8541 equal to or less than those shown.

8542	FLT_EPSILON	The difference between 1.0 and the least value greater than 1.0 that is representable in the given floating-point type,	1E-5
		$b^{(1-p)}$	
	DBL_EPSILON		1E-9
	LDBL_EPSILON		1E-9
	FLT_MIN	Minimum normalized positive floating-point number,	1E-37
		$b^{(e_{\min} - 1)}$	
	DBL_MIN		1E-37
	LDBL_MIN		1E-37

8543 **APPLICATION USAGE**

8544 None.

8545 **RATIONALE**

8546 None.

8547 **FUTURE DIRECTIONS**

8548 None.

8549 **SEE ALSO**

8550 None.

8551 **CHANGE HISTORY**

8552 First released in Issue 4. Derived from the ISO C standard.

8553 **Issue 6**

8554 The description of the operations with floating-point values is updated for alignment with the
8555 ISO/IEC 9899:1999 standard.

8556 **NAME**

8557 fmtmsg.h — message display structures

8558 **SYNOPSIS**

8559 XSI #include <fmtmsg.h>

8560

8561 **DESCRIPTION**

8562 The <fmtmsg.h> header shall define the following macros, which expand to constant integral
8563 expressions:

- 8564 MM_HARD Source of the condition is hardware.
- 8565 MM_SOFT Source of the condition is software.
- 8566 MM_FIRM Source of the condition is firmware.
- 8567 MM_APPL Condition detected by application.
- 8568 MM_UTIL Condition detected by utility.
- 8569 MM_OPSYS Condition detected by operating system.
- 8570 MM_RECOVER Recoverable error.
- 8571 MM_NRECOV Non-recoverable error.
- 8572 MM_HALT Error causing application to halt.
- 8573 MM_ERROR Application has encountered a non-fatal fault.
- 8574 MM_WARNING Application has detected unusual non-error condition.
- 8575 MM_INFO Informative message.
- 8576 MM_NOSEV No severity level provided for the message.
- 8577 MM_PRINT Display message on standard error.
- 8578 MM_CONSOLE Display message on system console.

8579 The table below indicates the null values and identifiers for *fmtmsg()* arguments. The
8580 <fmtmsg.h> header shall define the macros in the **Identifier** column, which expand to constant
8581 expressions that expand to expressions of the type indicated in the **Type** column:

8582

8583

Argument	Type	Null-Value	Identifier
<i>label</i>	char *	(char*)0	MM_NULLLBL
<i>severity</i>	int	0	MM_NULLSEV
<i>class</i>	long	0L	MM_NULLMC
<i>text</i>	char *	(char*)0	MM_NULLTXT
<i>action</i>	char *	(char*)0	MM_NULLACT
<i>tag</i>	char *	(char*)0	MM_NULLTAG

8584

8585

8586

8587

8588

8589

8590 The <fmtmsg.h> header shall also define the following macros for use as return values for
8591 *fmtmsg()*:

- 8592 MM_OK The function succeeded.
- 8593 MM_NOTOK The function failed completely.
- 8594 MM_NOMSG The function was unable to generate a message on standard error, but
8595 otherwise succeeded.

8596 MM_NOCON The function was unable to generate a console message, but otherwise
8597 succeeded.

8598 The following shall be declared as a function and may also be defined as a macro. A function
8599 prototype shall be provided for use with an ISO C standard compiler.

```
8600           int fmtmsg(long, const char *, int,  
8601                const char *, const char *, const char *);
```

8602 **APPLICATION USAGE**

8603 None.

8604 **RATIONALE**

8605 None.

8606 **FUTURE DIRECTIONS**

8607 None.

8608 **SEE ALSO**

8609 The System Interfaces volume of IEEE Std. 1003.1-200x, *fmtmsg()*

8610 **CHANGE HISTORY**

8611 First released in Issue 4, Version 2.

8612 **NAME**

8613 fnmatch.h — file name-matching types

8614 **SYNOPSIS**

8615 #include <fnmatch.h>

8616 **DESCRIPTION**8617 The <fnmatch.h> header shall define the flags and return value used by the *fnmatch()* function.

8618 The following constants are defined:

8619 FNM_NOMATCH The string does not match the specified pattern.

8620 FNM_PATHNAME Slash in *string* only matches slash in *pattern*.8621 FNM_PERIOD Leading period in *string* must be exactly matched by period in *pattern*.

8622 FNM_NOESCAPE Disable backslash escaping.

8623 FNM_NOSYS The implementation does not support this function. **(LEGACY)**8624 The following shall be declared as a function and may also be declared as a macro. Function
8625 prototypes shall be provided for use with an ISO C standard compiler.

8626 int fnmatch(const char *, const char *, int);

8627 **APPLICATION USAGE**

8628 None.

8629 **RATIONALE**

8630 None.

8631 **FUTURE DIRECTIONS**

8632 None.

8633 **SEE ALSO**8634 The System Interfaces volume of IEEE Std. 1003.1-200x, *fnmatch()*, the Shell and Utilities volume
8635 of IEEE Std. 1003.1-200x8636 **CHANGE HISTORY**

8637 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8638 **Issue 6**

8639 The constant FNM_NOSYS is marked LEGACY.

8640 **NAME**8641 `ftw.h` — file tree traversal8642 **SYNOPSIS**8643 XSI `#include <ftw.h>`

8644

8645 **DESCRIPTION**8646 The `<ftw.h>` header shall define the **FTW** structure that includes at least the following members:8647 `int base`8648 `int level`8649 The `<ftw.h>` header shall define macros for use as values of the third argument to the
8650 application-supplied function that is passed as the second argument to `ftw()` and `nftw()`:8651 `FTW_F` File.8652 `FTW_D` Directory.8653 `FTW_DNR` Directory without read permission.8654 `FTW_DP` Directory with subdirectories visited.8655 `FTW_NS` Unknown type; `stat()` failed.8656 `FTW_SL` Symbolic link.8657 `FTW_SLN` Symbolic link that names a nonexistent file.8658 The `<ftw.h>` header shall define macros for use as values of the fourth argument to `nftw()`:8659 `FTW_PHYS` Physical walk, does not follow symbolic links. Otherwise, `nftw()` follows
8660 links but does not walk down any path that crosses itself.8661 `FTW_MOUNT` The walk does not cross a mount point.8662 `FTW_DEPTH` All subdirectories are visited before the directory itself.8663 `FTW_CHDIR` The walk changes to each directory before reading it.8664 The following shall be declared as functions and may also be defined as macros. Function
8665 prototypes shall be provided for use with an ISO C standard compiler.8666 `int ftw(const char *,`
8667 `int (*)(const char *, const struct stat *, int), int);`
8668 `int nftw(const char *, int (*)`
8669 `(const char *, const struct stat *, int, struct FTW*),`
8670 `int, int);`8671 The `<ftw.h>` header shall define the **stat** structure and the symbolic names for `st_mode` and the
8672 file type test macros as described in `<sys/stat.h>`.8673 Inclusion of the `<ftw.h>` header may also make visible all symbols from `<sys/stat.h>`.

8674 **APPLICATION USAGE**

8675 None.

8676 **RATIONALE**

8677 None.

8678 **FUTURE DIRECTIONS**

8679 None.

8680 **SEE ALSO**8681 <sys/stat.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *ftw()*, *nftw()*8682 **CHANGE HISTORY**

8683 First released in Issue 1. Derived from Issue 1 of the SVID.

8684 **Issue 4**

8685 The function declarations in this header are expanded to full ISO C standard prototypes.

8686 A reference to <sys/stat.h> is added for the definition of the **stat** structure, the symbolic names
8687 for *st_mode*, and the file type test macros.8688 A warning is added indicating that inclusion of <ftw.h> may also make visible all symbols from
8689 <sys/stat.h>.8690 **Issue 4, Version 2**

8691 The following changes are incorporated in the DESCRIPTION for X/OPEN UNIX conformance:

- 8692
- The **FTW** structure is defined.
 - 8693 • The *nftw()* function is declared by the header and is mentioned as one of the functions to
8694 which the first list of macros applies.
 - 8695 • FTW_SL and FTW_SLN are added to the first list of macros to handle symbolic links.
 - 8696 • Macros for use as values of the fourth argument to *nftw()* are defined.

8697 **Issue 5**

8698 A description of FTW_DP is added.

8699 **NAME**8700 `glob.h` — path name pattern-matching types8701 **SYNOPSIS**8702 `#include <glob.h>`8703 **DESCRIPTION**8704 The **<glob.h>** header shall define the structures and symbolic constants used by the `glob()`
8705 function.8706 The structure type **glob_t** shall contain at least the following members:8707 `size_t gl_pathc` Count of paths matched by *pattern*.8708 `char **gl_pathv` Pointer to a list of matched path names.8709 `size_t gl_offs` Slots to reserve at the beginning of *gl_pathv*.8710 The following constants shall be provided as values for the *flags* argument:8711 **GLOB_APPEND** Append generated path names to those previously obtained.8712 **GLOB_DOOFFS** Specify how many null pointers to add to the beginning of *pglob-*
8713 *>gl_pathv*.8714 **GLOB_ERR** Cause `glob()` to return on error.8715 **GLOB_MARK** Each path name that is a directory that matches *pattern* has a slash
8716 appended.8717 **GLOB_NOCHECK** If *pattern* does not match any path name, then return a list consisting of
8718 only *pattern*.8719 **GLOB_NOESCAPE** Disable backslash escaping.8720 **GLOB_NOSORT** Do not sort the path names returned.

8721 The following constants shall be defined as error return values:

8722 **GLOB_ABORTED** The scan was stopped because **GLOB_ERR** was set or `(*errfunc)()`
8723 returned non-zero.8724 **GLOB_NOMATCH** The *pattern* does not match any existing path name, and
8725 **GLOB_NOCHECK** was not set in flags.8726 **GLOB_NOSPACE** An attempt to allocate memory failed.8727 **GLOB_NOSYS** The implementation does not support this function.8728 The following shall be declared as functions and may also be declared as macros. Function
8729 prototypes shall be provided for use with an ISO C standard compiler.8730 `int glob(const char *restrict, int, int (*restrict)(const char *, int),`
8731 `glob_t *restrict);`8732 `void globfree (glob_t *);`8733 The implementation may define additional macros or constants using names beginning with
8734 **GLOB_**.

8735 **APPLICATION USAGE**

8736 None.

8737 **RATIONALE**

8738 None.

8739 **FUTURE DIRECTIONS**

8740 None.

8741 **SEE ALSO**8742 The System Interfaces volume of IEEE Std. 1003.1-200x, *glob()*, the Shell and Utilities volume of
8743 IEEE Std. 1003.1-200x8744 **CHANGE HISTORY**

8745 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8746 **Issue 6**8747 The **restrict** keyword is added to the prototype for *glob()*.

8748 **NAME**

8749 grp.h — group structure

8750 **SYNOPSIS**

8751 #include <grp.h>

8752 **DESCRIPTION**8753 The <grp.h> header shall declare the structure **group** which shall include the following
8754 members:

8755 char *gr_name The name of the group.
 8756 gid_t gr_gid Numerical group ID.
 8757 char **gr_mem Pointer to a null-terminated array of character
 8758 pointers to member names.

8759 The **gid_t** type shall be defined as described in <sys/types.h>.8760 The following shall be declared as functions and may also be defined as macros. Function
8761 prototypes shall be provided for use with an ISO C standard compiler.

```
8762 struct group *getgrgid(gid_t);
8763 struct group *getgrnam(const char *);
8764 TSF int getgrgid_r(gid_t, struct group *, char *,
8765 size_t, struct group **);
8766 int getgrnam_r(const char *, struct group *, char *,
8767 size_t, struct group **);
8768 XSI struct group *getgrent(void);
8769 void endgrent(void);
8770 void setgrent(void);
8771
```

8772 **APPLICATION USAGE**

8773 None.

8774 **RATIONALE**

8775 None.

8776 **FUTURE DIRECTIONS**

8777 None.

8778 **SEE ALSO**8779 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *endgrent()*, *getgrgid()*,
8780 *getgrnam()*8781 **CHANGE HISTORY**

8782 First released in Issue 1.

8783 **Issue 4**8784 A reference to <sys/types.h> is added for the definition of **gid_t** and marked as an extension.

8785 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 8786
- The function declarations in this header are expanded to full ISO C standard prototypes.

8787 **Issue 4, Version 2**8788 For X/OPEN UNIX conformance, the *getgrent()*, *endgrent()*, and *setgrent()* functions are added
8789 to the list of functions declared in this header.

8790 **Issue 5**

8791 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8792 **Issue 6**

8793 The following new requirements on POSIX implementations derive from alignment with the
8794 Single UNIX Specification:

- 8795 • The definition of **gid_t** is mandated.
- 8796 • The *getgrgid_r()* and *getgrnam_r()* functions are marked as part of the Thread-Safe Functions
8797 option.

8798 **NAME**

8799 iconv.h — codeset conversion facility

8800 **SYNOPSIS**8801 XSI `#include <iconv.h>`

8802

8803 **DESCRIPTION**8804 The **<iconv.h>** header shall define the following data type through **typedef**:8805 **iconv_t** Identifies the conversion from one codeset to another.8806 The following shall be declared as functions and may also be declared as macros. Function
8807 prototypes shall be provided for use with an ISO C standard compiler.8808 `iconv_t iconv_open(const char *, const char *);`8809 `size_t iconv(iconv_t, char **restrict, size_t *restrict, char **restrict,`
8810 `size_t *restrict);`8811 `int iconv_close(iconv_t);`8812 **APPLICATION USAGE**

8813 None.

8814 **RATIONALE**

8815 None.

8816 **FUTURE DIRECTIONS**

8817 None.

8818 **SEE ALSO**8819 The System Interfaces volume of IEEE Std. 1003.1-200x, *iconv()*, *iconv_close()*, *iconv_open()*8820 **CHANGE HISTORY**

8821 First released in Issue 4.

8822 **Issue 6**8823 The **restrict** keyword is added to the prototype for *iconv()*.

8824 NAME

8825 inttypes.h — fixed size integer types

8826 SYNOPSIS

8827 XSI `#include <inttypes.h>`

8828

8829 DESCRIPTION

8830 The <inttypes.h> header shall include the <stdint.h> header.

8831 **Notes to Reviewers**8832 *This section with side shading will not appear in the final copy. - Ed.*8833 Reviewers are asked to propose changes to eliminate duplication between inttypes.h and
8834 stdint.h.

8835 The <inttypes.h> header shall include definitions of at least the following types:

8836 **imaxdiv_t** Structure type that is the type of the value returned by the *imaxdiv()* function.8837 **int8_t** 8-bit signed integer type.8838 **int16_t** 16-bit signed integer type.8839 **int32_t** 32-bit signed integer type.8840 **uint8_t** 8-bit unsigned integer type.8841 **uint16_t** 16-bit unsigned integer type.8842 **uint32_t** 32-bit unsigned integer type.8843 **intptr_t** Signed integer type large enough to hold any pointer.8844 **uintptr_t** Unsigned integer type large enough to hold any pointer.

8845 If any of the following are true:

8846 • The implementation supports the `_POSIX_V6_ILP32_OFFBIG` programming environment
8847 and the application is being built in the `_POSIX_V6_ILP32_OFFBIG` programming
8848 environment (see the Shell and Utilities volume of IEEE Std. 1003.1-200x, *c99*, Programming
8849 Environments).8850 • The implementation supports the `_POSIX_V6_LP64_OFF64` programming environment and
8851 the application is being built in the `_POSIX_V6_LP64_OFF64` programming environment.8852 • The implementation supports the `_POSIX_V6_LPBIG_OFFBIG` programming environment
8853 and the application is being built in the `_POSIX_V6_LPBIG_OFFBIG` programming
8854 environment.

8855 then <inttypes.h> also shall include definitions for the following types:

8856 **int64_t** 64-bit signed integer type.8857 **uint64_t** 64-bit unsigned integer type.8858 If `__STDC_FORMAT_MACROS` is defined before <inttypes.h> is included, then the following
8859 object-like macros shall be defined. Each expands to a character string literal containing a
8860 conversion specifier, possibly modified by a length modifier, suitable for use within the *format*
8861 argument of a formatted input/output function when converting the corresponding integer
8862 type. These macro names have the general form of PRI (character string literals for the *fprintf()*
8863 and *fwprintf()* family of functions) or SCN (character string literals for the *fscanf()* and *fwscanf()*)

8864 family of functions), followed by the conversion specifier, followed by a name corresponding to
 8865 a similar type name in <stdint.h>. In these names, N represents the width of the type as
 8866 described in *stdint.h*(. For example, *PRIdFAST32* can be used in a format string to print the
 8867 value of an integer of type *int_fast32_t*.

8868 The *fprintf*() macros for signed integers are:

8869	PRIdN	PRIdLEASTN	PRIdFASTN	PRIdMAX	PRIdPTR
8870	PRiIN	PRiLEASTN	PRiFASTN	PRiMAX	PRiPTR

8871 The *fprintf*() macros for unsigned integers are:

8872	PRIoN	PRIoLEASTN	PRIoFASTN	PRIoMAX	PRIoPTR
8873	PRIoN	PRIoLEASTN	PRIoFASTN	PRIoMAX	PRIoPTR
8874	PRIxN	PRIxLEASTN	PRIxFASTN	PRIxMAX	PRIxPTR
8875	PRIXN	PRIXLEASTN	PRIXFASTN	PRIXMAX	PRIXPTR

8876 The *fscanf*() macros for signed integers are:

8877	SCNdN	SCNdLEASTN	SCNdFASTN	SCNdMAX	SCNdPTR
8878	SCNiN	SCNiLEASTN	SCNiFASTN	SCNiMAX	SCNiPTR

8879 The *fscanf*() macros for unsigned integers are:

8880	SCNoN	SCNoLEASTN	SCNoFASTN	SCNoMAX	SCNoPTR
8881	SCNuN	SCNuLEASTN	SCNuFASTN	SCNuMAX	SCNuPTR
8882	SCNxN	SCNxLEASTN	SCNxFASTN	SCNxMAX	SCNxPTR

8883 For each type that the implementation provides in <stdint.h>, the corresponding *fprintf*()
 8884 macros shall be defined and the corresponding *fscanf*() macros shall be defined unless the
 8885 implementation does not have a suitable *fscanf* length modifier for the type.

8886 The following shall be declared as functions and may also be defined as macros. Function
 8887 prototypes shall be provided for use with an ISO C standard compiler.

```
8888 intmax_t imaxabs(intmax_t);
8889 imaxdiv_t imaxdiv(intmax_t, intmax_t);
8890 intmax_t strtoumax(const char *restrict, char **restrict, int);
8891 uintmax_t strtoumax(const char *restrict, char **restrict, int);
8892 intmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);
8893 uintmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);
```

8894 **EXAMPLES**

```
8895 #include <inttypes.h>
8896 #include <wchar.h>
8897 int main(void)
8898 {
8899     uintmax_t i = UINTMAX_MAX; // This type always exists.
8900     wprintf(L"The largest integer value is %020"
8901           PRIxMAX "\n", i);
8902     return 0;
8903 }
```

8904 **APPLICATION USAGE**

8905 None.

8906 **RATIONALE**

8907 The <inttypes.h> header was derived from the header of the same name found on several
8908 existing 64-bit systems. The C Standard Committee debated other methods for specifying
8909 integer sizes and other characteristics, but in the end decided to standardize existing practice
8910 rather than innovate in this area.

8911 The ISO/IEC 9899:1990 standard specifies that the language should support four signed and
8912 unsigned integer data types—**char**, **short**, **int**, and **long**—but places very little requirement on
8913 their size other than that **int** and **short** be at least 16 bits and **long** be at least as long as **int** and
8914 not smaller than 32 bits. For 16-bit systems, most implementations assign 8, 16, 16, and 32 bits to
8915 **char**, **short**, **int**, and **long**, respectively. For 32-bit systems, the common practice is to assign 8, 16,
8916 32, and 32 bits to these types. This difference in **int** size can create some problems for users who
8917 migrate from one system to another which assigns different sizes to integer types, because the
8918 ISO C standard integer promotion rule can produce silent changes unexpectedly. The need for
8919 defining an extended integer type increased with the introduction of 64-bit systems.

8920 The purpose of <inttypes.h> is to provide a set of integer types whose definitions are consistent
8921 across machines and independent of operating systems and other implementation
8922 idiosyncrasies. It defines, via **typedef**, integer types of various sizes. Implementations are free to
8923 **typedef** them as ISO C standard integer types or extensions that they support. Consistent use of
8924 this header will greatly increase the portability of a users program across platforms.

8925 **FUTURE DIRECTIONS**

8926 Macro names beginning with PRI or SCN followed by any lowercase letter or 'x' may be added
8927 to the macros defined in the <inttypes.h> header.

8928 **SEE ALSO**8929 The System Interfaces volume of IEEE Std. 1003.1-200x, *imaxdiv()*8930 **CHANGE HISTORY**

8931 First released in Issue 5.

8932 **Issue 6**

8933 The Open Group Base Resolution bwg97-006 is applied.

8934 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

8935 **NAME**

8936 iso646.h — alternative spellings

8937 **SYNOPSIS**

8938 #include <iso646.h>

8939 **DESCRIPTION**

8940 **cx** The functionality described on this reference page extends the ISO C standard. Applications
8941 shall define the appropriate feature test macro (see the System Interfaces volume of
8942 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
8943 symbols in this header.

8944 The **<iso646.h>** header shall define the following eleven macros (on the left) that expand to the
8945 corresponding tokens (on the right):

8946 *and* &&8947 *and_eq* &=8948 *bitand* &8949 *bitor* |8950 *compl* ~8951 *not* !8952 *not_eq* !=8953 *or* | |8954 *or_eq* |=8955 *xor* ^8956 *xor_eq* ^=8957 **APPLICATION USAGE**

8958 None.

8959 **RATIONALE**

8960 None.

8961 **FUTURE DIRECTIONS**

8962 None.

8963 **SEE ALSO**

8964 None.

8965 **CHANGE HISTORY**

8966 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

8967 **NAME**

8968 langinfo.h — language information constants

8969 **SYNOPSIS**

8970 xSI #include <langinfo.h>

8971

8972 **DESCRIPTION**

8973 The <langinfo.h> header contains the constants used to identify items of *langinfo* data (see
8974 *nl_langinfo()*). The type of the constant, **nl_item**, shall be defined as described in <nl_types.h>.

8975 The following constants shall be defined. The entries under **Category** indicate in which
8976 *setlocale()* category each item is defined.

8977

8978

Constant	Category	Meaning
CODESET	LC_CTYPE	Codeset name.
D_T_FMT	LC_TIME	String for formatting date and time.
D_FMT	LC_TIME	Date format string.
T_FMT	LC_TIME	Time format string.
T_FMT_AMPM	LC_TIME	a.m. or p.m. time format string.
AM_STR	LC_TIME	Ante Meridian affix.
PM_STR	LC_TIME	Post Meridian affix.
DAY_1	LC_TIME	Name of the first day of the week (for example, Sunday).
DAY_2	LC_TIME	Name of the second day of the week (for example, Monday).
DAY_3	LC_TIME	Name of the third day of the week (for example, Tuesday).
DAY_4	LC_TIME	Name of the fourth day of the week (for example, Wednesday).
DAY_5	LC_TIME	Name of the fifth day of the week (for example, Thursday).
DAY_6	LC_TIME	Name of the sixth day of the week (for example, Friday).
DAY_7	LC_TIME	Name of the seventh day of the week (for example, Saturday).
ABDAY_1	LC_TIME	Abbreviated name of the first day of the week.
ABDAY_2	LC_TIME	Abbreviated name of the second day of the week.
ABDAY_3	LC_TIME	Abbreviated name of the third day of the week.
ABDAY_4	LC_TIME	Abbreviated name of the fourth day of the week.
ABDAY_5	LC_TIME	Abbreviated name of the fifth day of the week.
ABDAY_6	LC_TIME	Abbreviated name of the sixth day of the week.
ABDAY_7	LC_TIME	Abbreviated name of the seventh day of the week.
MON_1	LC_TIME	Name of the first month of the year.
MON_2	LC_TIME	Name of the second month.
MON_3	LC_TIME	Name of the third month.
MON_4	LC_TIME	Name of the fourth month.
MON_5	LC_TIME	Name of the fifth month.
MON_6	LC_TIME	Name of the sixth month.
MON_7	LC_TIME	Name of the seventh month.
MON_8	LC_TIME	Name of the eighth month.
MON_9	LC_TIME	Name of the ninth month.
MON_10	LC_TIME	Name of the tenth month.
MON_11	LC_TIME	Name of the eleventh month.
MON_12	LC_TIME	Name of the twelfth month.

8913

9014
9015
9016
9017
9018
9019
9020
9021
9022
9023
9024
9025
9026
9027
9028
9029
9030
9031
9032
9033
9034
9035
9036
9037
9038
9039
9040

Constant	Category	Meaning
ABMON_1	LC_TIME	Abbreviated name of the first month.
ABMON_2	LC_TIME	Abbreviated name of the second month.
ABMON_3	LC_TIME	Abbreviated name of the third month.
ABMON_4	LC_TIME	Abbreviated name of the fourth month.
ABMON_5	LC_TIME	Abbreviated name of the fifth month.
ABMON_6	LC_TIME	Abbreviated name of the sixth month.
ABMON_7	LC_TIME	Abbreviated name of the seventh month.
ABMON_8	LC_TIME	Abbreviated name of the eighth month.
ABMON_9	LC_TIME	Abbreviated name of the ninth month.
ABMON_10	LC_TIME	Abbreviated name of the tenth month.
ABMON_11	LC_TIME	Abbreviated name of the eleventh month.
ABMON_12	LC_TIME	Abbreviated name of the twelfth month.
ERA	LC_TIME	Era description segments.
ERA_D_FMT	LC_TIME	Era date format string.
ERA_D_T_FMT	LC_TIME	Era date and time format string.
ERA_T_FMT	LC_TIME	Era time format string.
ALT_DIGITS	LC_TIME	Alternative symbols for digits.
RADIXCHAR	LC_NUMERIC	Radix character.
THOUSEP	LC_NUMERIC	Separator for thousands.
YESEXPR	LC_MESSAGES	Affirmative response expression.
NOEXPR	LC_MESSAGES	Negative response expression.
CRNCYSTR	LC_MONETARY	Currency symbol, preceded by '-' if the symbol should appear before the value, '+' if the symbol should appear after the value, or '.' if the symbol should replace the radix character.

9041 If the locale's value for **p_cs_precedes** and **n_cs_precedes** does not match, the value of
9042 *nl_langinfo*(CRNCYSTR) is unspecified.

9043 The following shall be declared as a function and may also be declared as a macro. Function
9044 prototypes shall be provided for use with an ISO C standard compiler.

9045

```
char *nl_langinfo(nl_item);
```

9046 Inclusion of the **<langinfo.h>** header may also make visible all symbols from **<nl_types.h>**.

9047 APPLICATION USAGE

9048 Wherever possible, users are advised to use functions compatible with those in the ISO C
9049 standard to access items of *langinfo* data. In particular, the *strptime*() function should be used to
9050 access date and time information defined in category *LC_TIME*. The *localeconv*() function
9051 should be used to access information corresponding to *RADIXCHAR*, *THOUSEP*, and
9052 *CRNCYSTR*.

9053 RATIONALE

9054 None.

9055 FUTURE DIRECTIONS

9056 None.

9057 SEE ALSO

9058 The System Interfaces volume of IEEE Std. 1003.1-200x, *nl_langinfo*(), *localeconv*(), *strfmon*(),
9059 *strptime*(), Chapter 7 (on page 143)

9060 **CHANGE HISTORY**

9061 First released in Issue 2.

9062 **Issue 4**

9063 The function declarations in this header are expanded to full ISO C standard prototypes.

9064 The constants CODESET, T_FMT_AMPM, ERA, ERA_D_FMT, ALT_DIGITS, YESEXPR, and
9065 NOEXPR are added.

9066 The constants YESSTR and NOSTR are marked TO BE WITHDRAWN.

9067 Reference to the Gregorian calendar is removed.

9068 The constants YESSTR and NOSTR are now defined as belonging to category *LC_MESSAGES*.
9069 Previously they were defined as constants in category *LC_ALL*.9070 A warning is added indicating that inclusion of <langinfo.h> may also make visible all symbols
9071 from <nl_types.h>.9072 The APPLICATION USAGE section is expanded to recommend use of the *localeconv()* function.9073 **Issue 5**

9074 The constants YESSTR and NOSTR are marked LEGACY.

9075 **Issue 6**

9076 The constants YESSTR and NOSTR are removed.

9077 **NAME**

9078 libgen.h — definitions for pattern matching functions

9079 **SYNOPSIS**

9080 xSI #include <libgen.h>

9081

9082 **DESCRIPTION**9083 The following shall be declared as functions and may also be defined as macros. Function
9084 prototypes shall be provided for use with an ISO C standard compiler.

9085 char *basename(char *);

9086 char *dirname(char *);

9087 **APPLICATION USAGE**

9088 None.

9089 **RATIONALE**

9090 None.

9091 **FUTURE DIRECTIONS**

9092 None.

9093 **SEE ALSO**9094 The System Interfaces volume of IEEE Std. 1003.1-200x, *basename()*, *dirname()*9095 **CHANGE HISTORY**

9096 First released in Issue 4, Version 2.

9097 **Issue 5**9098 The function prototypes for *basename()* and *dirname()* are changed to indicate that the first
9099 argument is of type **char*** rather than **const char***.

9100 **NAME**

9101 limits.h — implementation-defined constants

9102 **SYNOPSIS**

9103 #include <limits.h>

9104 **DESCRIPTION**

9105 cx The functionality described on this reference page extends the ISO C standard. Applications
 9106 shall define the appropriate feature test macro (see the System Interfaces volume of
 9107 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 9108 symbols in this header.

9109 The <limits.h> header shall define various symbolic names. Different categories of names are
 9110 described below.

9111 The names represent various limits on resources that the implementation imposes on
 9112 applications.

9113 Implementations may choose any appropriate value for each limit, provided it is not more
 9114 restrictive than the Minimum Acceptable Values listed below. Symbolic constant names
 9115 beginning with _POSIX may be found in <unistd.h>.

9116 Applications should not assume any particular value for a limit. To achieve maximum
 9117 portability, an application should not require more resource than the Minimum Acceptable
 9118 Value quantity. However, an application wishing to avail itself of the full amount of a resource
 9119 available on an implementation may make use of the value given in <limits.h> on that
 9120 particular implementation, by using the symbolic names listed below. It should be noted,
 9121 however, that many of the listed limits are not invariant, and at runtime, the value of the limit
 9122 may differ from those given in this header, for the following reasons:

- 9123 • The limit is path name-dependent.
- 9124 • The limit differs between the compile and runtime machines.

9125 For these reasons, an application may use the *fpathconf()*, *pathconf()*, and *sysconf()* functions to
 9126 determine the actual value of a limit at runtime.

9127 The items in the list ending in _MIN give the most negative values that the mathematical types
 9128 are guaranteed to be capable of representing. Numbers of a more negative value may be
 9129 supported on some implementations, as indicated by the <limits.h> header on the
 9130 implementation, but applications requiring such numbers are not guaranteed to be portable to
 9131 all implementations. For positive constants ending in _MIN, this indicates the minimum
 9132 acceptable value.

9133 The Minimum Acceptable Value symbol ' * ' indicates that there is no guaranteed value across
 9134 all conforming implementations.

9135 **Runtime Invariant Values (Possibly Indeterminate)**

9136 A definition of one of the symbolic names in the following list shall be omitted from <limits.h>
 9137 on specific implementations where the corresponding value is equal to or greater than the stated
 9138 minimum, but is indeterminate.

9139 This indetermination might depend on the amount of available memory space on a specific
 9140 instance of a specific implementation. The actual value supported by a specific instance shall be
 9141 provided by the *sysconf()* function.

9142 AIO {AIO_LISTIO_MAX}

9143 Maximum number of I/O operations in a single list I/O call supported by the

9144		implementation.
9145		Minimum Acceptable Value: <code>{_POSIX_AIO_LISTIO_MAX}</code>
9146	AIO	<code>{AIO_MAX}</code>
9147		Maximum number of outstanding asynchronous I/O operations supported by the
9148		implementation.
9149		Minimum Acceptable Value: <code>{_POSIX_AIO_MAX}</code>
9150	AIO	<code>{AIO_PRIO_DELTA_MAX}</code>
9151		The maximum amount by which a process can decrease its asynchronous I/O priority level
9152		from its own scheduling priority.
9153		Minimum Acceptable Value: 0
9154		<code>{ARG_MAX}</code>
9155		Maximum length of argument to the <i>exec</i> functions including environment data.
9156		Minimum Acceptable Value: <code>{_POSIX_ARG_MAX}</code>
9157	XSI	<code>{ATEXIT_MAX}</code>
9158		Maximum number of functions that may be registered with <i>atexit()</i> .
9159		Minimum Acceptable Value: 32
9160		<code>{CHILD_MAX}</code>
9161		Maximum number of simultaneous processes per real user ID.
9162		Minimum Acceptable Value: 25
9163	TMR	<code>{DELAYTIMER_MAX}</code>
9164		Maximum number of timer expiration overruns.
9165		Minimum Acceptable Value: <code>{_POSIX_DELAYTIMER_MAX}</code>
9166	XSI	<code>{IOV_MAX}</code>
9167		Maximum number of <i>iovec</i> structures that one process has available for use with <i>readv()</i> or
9168		<i>writenv()</i> .
9169		Minimum Acceptable Value: <code>{_XOPEN_IOV_MAX}</code>
9170		<code>{LOGIN_NAME_MAX}</code>
9171		Maximum length of a login name.
9172		Minimum Acceptable Value: <code>{_POSIX_LOGIN_NAME_MAX}</code>
9173	MSG	<code>{MQ_OPEN_MAX}</code>
9174		The maximum number of open message queue descriptors a process may hold.
9175		Minimum Acceptable Value: <code>{_POSIX_MQ_OPEN_MAX}</code>
9176	MSG	<code>{MQ_PRIO_MAX}</code>
9177		The maximum number of message priorities supported by the implementation.
9178		Minimum Acceptable Value: <code>{_POSIX_MQ_PRIO_MAX}</code>
9179		<code>{OPEN_MAX}</code>
9180		Maximum number of files that one process can have open at any one time.
9181		Minimum Acceptable Value: 20
9182		<code>{PAGESIZE}</code>
9183		Size in bytes of a page.
9184		Minimum Acceptable Value: 1
9185	XSI	<code>{PAGE_SIZE}</code>
9186		Same as <code>{PAGESIZE}</code> . If either <code>{PAGESIZE}</code> or <code>{PAGE_SIZE}</code> is defined, the other is defined
9187		with the same value.

9188	THR	{PTHREAD_DESTRUCTOR_ITERATIONS}
9189		Maximum number of attempts made to destroy a thread's thread-specific data values on thread exit.
9190		
9191		Minimum Acceptable Value: {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
9192	THR	{PTHREAD_KEYS_MAX}
9193		Maximum number of data keys that can be created by a process.
9194		Minimum Acceptable Value: {_POSIX_THREAD_KEYS_MAX}
9195	THR	{PTHREAD_STACK_MIN}
9196		Minimum size in bytes of thread stack storage.
9197		Minimum Acceptable Value: 0
9198	THR	{PTHREAD_THREADS_MAX}
9199		Maximum number of threads that can be created per process.
9200		Minimum Acceptable Value: {_POSIX_THREAD_THREADS_MAX}
9201		{RE_DUP_MAX}
9202		The number of repeated occurrences of a BRE permitted by the <i>regex()</i> and <i>regcomp()</i> functions when using the interval notation $\{m,n\}$; see Section 9.3.6 (on page 201).
9203		
9204		Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}
9205	RTS	{RTSIG_MAX}
9206		Maximum number of realtime signals reserved for application use in this implementation.
9207		Minimum Acceptable Value: {_POSIX_RTSIG_MAX}
9208	SEM	{SEM_NSEMS_MAX}
9209		Maximum number of semaphores that a process may have.
9210		Minimum Acceptable Value: {_POSIX_SEM_NSEMS_MAX}
9211	SEM	{SEM_VALUE_MAX}
9212		The maximum value a semaphore may have.
9213		Minimum Acceptable Value: {_POSIX_SEM_VALUE_MAX}
9214	RTS	{SIGQUEUE_MAX}
9215		Maximum number of queued signals that a process may send and have pending at the receiver(s) at any time.
9216		
9217		Minimum Acceptable Value: {_POSIX_SIGQUEUE_MAX}
9218	SS TSP	{SS_REPL_MAX}
9219		The maximum number of replenishment operations that may be simultaneously pending for a particular sporadic server scheduler.
9220		
9221		Minimum Acceptable Value: {_POSIX_SS_REPL_MAX}
9222		{STREAM_MAX}
9223		The number of streams that one process can have open at one time. If defined, it has the same value as {FOPEN_MAX} (see <stdio.h>).
9224		
9225		Minimum Acceptable Value: {_POSIX_STREAM_MAX}
9226		{SYMLOOP_MAX}
9227		Maximum number of symbolic links that can be reliably traversed in the resolution of a path name in the absence of a loop.
9228		
9229		Minimum Acceptable Value: {_POSIX_SYMLOOP_MAX}
9230	TMR	{TIMER_MAX}
9231		Maximum number of timers per-process supported by the implementation.
9232		Minimum Acceptable Value: {_POSIX_TIMER_MAX}

9233	TRC	{TRACE_EVENT_NAME_MAX}
9234		Maximum length of the trace event name.
9235		Minimum Acceptable Value: {_POSIX_TRACE_EVENT_NAME_MAX}
9236	TRC	{TRACE_NAME_MAX}
9237		Maximum length of the trace generation version string or of the trace stream name.
9238		Minimum Acceptable Value: {_POSIX_TRACE_NAME_MAX}
9239	TRC	{TRACE_SYS_MAX}
9240		Maximum number of trace streams that may simultaneously exist in the system.
9241		Minimum Acceptable Value: {_POSIX_TRACE_SYS_MAX}
9242	TRC	{TRACE_USER_EVENT_MAX}
9243		Maximum number of user trace event type identifiers that may simultaneously exist in a
9244		traced process, including the predefined user trace event
9245		POSIX_TRACE_UNNAMED_USER_EVENT.
9246		Minimum Acceptable Value: {_POSIX_TRACE_USER_EVENT_MAX}
9247		{TTY_NAME_MAX}
9248		Maximum length of terminal device name.
9249		Minimum Acceptable Value: {_POSIX_TTY_NAME_MAX}
9250		{TZNAME_MAX}
9251		Maximum number of bytes supported for the name of a timezone (not of the TZ variable).
9252		Minimum Acceptable Value: {_POSIX_TZNAME_MAX}
9253		Note: The length given by {TZNAME_MAX} does not include the quoting characters
9254		mentioned in Section 8.3 (on page 192).
9255		Path Name Variable Values
9256		The values in the following list may be constants within an implementation or may vary from
9257		one path name to another. For example, file systems or directories may have different
9258		characteristics.
9259		A definition of one of the values shall be omitted from the <limits.h> header on specific
9260		implementations where the corresponding value is equal to or greater than the stated minimum,
9261		but where the value can vary depending on the file to which it is applied. The actual value
9262		supported for a specific path name shall be provided by the <i>pathconf()</i> function.
9263		{FILESIZEBITS}
9264		Minimum number of bits needed to represent, as a signed integer value, the maximum size
9265		of a regular file allowed in the specified directory.
9266		Minimum Acceptable Value: 32
9267		{LINK_MAX}
9268		Maximum number of links to a single file.
9269		Minimum Acceptable Value: {_POSIX_LINK_MAX}
9270		{MAX_CANON}
9271		Maximum number of bytes in a terminal canonical input line.
9272		Minimum Acceptable Value: {_POSIX_MAX_CANON}
9273		{MAX_INPUT}
9274		Minimum number of bytes for which space is available in a terminal input queue; therefore,
9275		the maximum number of bytes a portable application may require to be typed as input
9276		before reading them.
9277		Minimum Acceptable Value: {_POSIX_MAX_INPUT}

9278 {NAME_MAX}
 9279 Maximum number of bytes in a file name (not including terminating null).
 9280 Minimum Acceptable Value: {_POSIX_NAME_MAX}

9281 {PATH_MAX}
 9282 Maximum number of bytes in a path name, including the terminating null character.
 9283 Minimum Acceptable Value: {_POSIX_PATH_MAX}

9284 {PIPE_BUF}
 9285 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
 9286 Minimum Acceptable Value: {_POSIX_PIPE_BUF}

9287 ADV {POSIX_ALLOC_SIZE_MIN}
 9288 Minimum number of bytes of storage actually allocated for any portion of a file.
 9289 Minimum Acceptable Value: Not specified.

9290 ADV {POSIX_REC_INCR_XFER_SIZE}
 9291 Recommended increment for file transfer sizes between the
 9292 {POSIX_REC_MIN_XFER_SIZE} and {POSIX_REC_MAX_XFER_SIZE} values.
 9293 Minimum Acceptable Value: Not specified.

9294 ADV {POSIX_REC_MAX_XFER_SIZE}
 9295 Maximum recommended file transfer size.
 9296 Minimum Acceptable Value: Not specified.

9297 ADV {POSIX_REC_MIN_XFER_SIZE}
 9298 Minimum recommended file transfer size.
 9299 Minimum Acceptable Value: Not specified.

9300 ADV {POSIX_REC_XFER_ALIGN}
 9301 Recommended file transfer buffer alignment.
 9302 Minimum Acceptable Value: Not specified.

9303 {SYMLINK_MAX}
 9304 Maximum number of bytes in a symbolic link.
 9305 Minimum Acceptable Value: {_POSIX_SYMLINK_MAX}

9306 **Runtime Inceasable Values**

9307 The magnitude limitations in the following list shall be fixed by specific implementations. An
 9308 application should assume that the value supplied by <limits.h> in a specific implementation is
 9309 the minimum that pertains whenever the application is run under that implementation. A
 9310 specific instance of a specific implementation may increase the value relative to that supplied by
 9311 <limits.h> for that implementation. The actual value supported by a specific instance shall be
 9312 provided by the *sysconf()* function.

9313 {BC_BASE_MAX}
 9314 Maximum *obase* values allowed by the *bc* utility.
 9315 Minimum Acceptable Value: {_POSIX2_BC_BASE_MAX}

9316 {BC_DIM_MAX}
 9317 Maximum number of elements permitted in an array by the *bc* utility.
 9318 Minimum Acceptable Value: {_POSIX2_BC_DIM_MAX}

9319 {BC_SCALE_MAX}
 9320 Maximum *scale* value allowed by the *bc* utility.
 9321 Minimum Acceptable Value: {_POSIX2_BC_SCALE_MAX}

- 9322 {BC_STRING_MAX}
- 9323 Maximum length of a string constant accepted by the *bc* utility.
- 9324 Minimum Acceptable Value: {_POSIX2_BC_STRING_MAX}

- 9325 {CHARCLASS_NAME_MAX}
- 9326 Maximum number of bytes in a character class name.
- 9327 Minimum Acceptable Value: {_POSIX2_CHARCLASS_NAME_MAX}

- 9328 {COLL_WEIGHTS_MAX}
- 9329 Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order**
- 9330 keyword in the locale definition file; see Chapter 7 (on page 143).
- 9331 Minimum Acceptable Value: {_POSIX2_COLL_WEIGHTS_MAX}

- 9332 {EXPR_NEST_MAX}
- 9333 Maximum number of expressions that can be nested within parentheses by the *expr* utility.
- 9334 Minimum Acceptable Value: {_POSIX2_EXPR_NEST_MAX}

- 9335 {LINE_MAX}
- 9336 Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
- 9337 standard input or another file), when the utility is described as processing text files. The
- 9338 length includes room for the trailing newline.
- 9339 Minimum Acceptable Value: {_POSIX2_LINE_MAX}

- 9340 {NGROUPS_MAX}
- 9341 Maximum number of simultaneous supplementary group IDs per process.
- 9342 Minimum Acceptable Value: 8

- 9343 {RE_DUP_MAX}
- 9344 Maximum number of repeated occurrences of a regular expression permitted when using
- 9345 the interval notation $\{m,n\}$; see Chapter 9 (on page 195).
- 9346 Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}

9347 **Maximum Values**

9348 TMR The symbolic constants in the following list shall be defined in <limits.h> with the values
9349 shown. These are symbolic names for the most restrictive value for certain features on an
9350 implementation supporting the Timers option. A conforming implementation shall provide
9351 values no larger than these values. A portable application must not require a smaller value for
9352 correct operation.

9353 TMR {_POSIX_CLOCKRES_MIN}

9354 The resolution of the CLOCK_REALTIME clock, in nanoseconds.

9355 Value: 20 000 000

9356 MON If the Monotonic Clock option is supported, the resolution of the CLOCK_MONOTONIC
9357 clock, in nanoseconds, is represented by {_POSIX_CLOCKRES_MIN}.

9358 **Minimum Values**

9359 The symbolic constants in the following list shall be defined in <limits.h> with the values
9360 shown. These are symbolic names for the most restrictive value for certain features on an
9361 implementation conforming to this volume of IEEE Std. 1003.1-200x. Related symbolic constants
9362 are defined elsewhere in this volume of IEEE Std. 1003.1-200x which reflect the actual
9363 implementation and which need not be as restrictive. A conforming implementation shall
9364 provide values at least this large. A strictly conforming application must not require a larger
9365 value for correct operation.

9366	AIO	{_POSIX_AIO_LISTIO_MAX}
9367		The number of I/O operations that can be specified in a list I/O call.
9368		Value: 2
9369	AIO	{_POSIX_AIO_MAX}
9370		The number of outstanding asynchronous I/O operations.
9371		Value: 1
9372		{_POSIX_ARG_MAX}
9373		Maximum length of argument to the <i>exec</i> functions including environment data.
9374		Value: 4 096
9375		{_POSIX_CHILD_MAX}
9376		Maximum number of simultaneous processes per real user ID.
9377		Value: 6
9378	TMR	{_POSIX_DELAYTIMER_MAX}
9379		The number of timer expiration overruns.
9380		Value: 32
9381		{_POSIX_LINK_MAX}
9382		Maximum number of links to a single file.
9383		Value: 8
9384		{_POSIX_LOGIN_NAME_MAX}
9385		The size of the storage required for a login name, in bytes, including the terminating null.
9386		Value: 9
9387		{_POSIX_MAX_CANON}
9388		Maximum number of bytes in a terminal canonical input queue.
9389		Value: 255
9390		{_POSIX_MAX_INPUT}
9391		Maximum number of bytes allowed in a terminal input queue.
9392		Value: 255
9393	MSG	{_POSIX_MQ_OPEN_MAX}
9394		The number of message queues that can be open for a single process.
9395		Value: 8
9396	MSG	{_POSIX_MQ_PRIO_MAX}
9397		The maximum number of message priorities supported by the implementation.
9398		Value: 32

9399 **Notes to Reviewers**

9400 *This section with side shading will not appear in the final copy. - Ed.*

9401 D1, XSH, ERN 436 proposes increasing the value of {_POSIX_NAME_MAX} to 256.

9402 Similarly, it proposes {_POSIX_PATH_MAX} be 1 024.

9403 {_POSIX_NAME_MAX}

9404 Maximum number of bytes in a file name (not including terminating null).

9405 Value: 14

9406 **Notes to Reviewers**

9407 *This section with side shading will not appear in the final copy. - Ed.*

9408 D1, XSH, ERN 19 proposes to increase `{_POSIX_NGROUPS_MAX}`, `{_POSIX_OPEN_MAX}`,
9409 and `{_POSIX_CHILD_MAX}` to their FIPS values (8, 20, 25) as with the limits equivalents
9410 without the leading `_POSIX`).

9411 `{_POSIX_NGROUPS_MAX}`
9412 Maximum number of simultaneous supplementary group IDs per process.
9413 Value: 0

9414 `{_POSIX_OPEN_MAX}`
9415 Maximum number of files that one process can have open at any one time.
9416 Value: 16

9417 `{_POSIX_PATH_MAX}`
9418 Maximum number of bytes in a path name.
9419 Value: 256

9420 `{_POSIX_PIPE_BUF}`
9421 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
9422 Value: 512

9423 `{_POSIX_RE_DUP_MAX}`
9424 The number of repeated occurrences of a BRE permitted by the `regex()` and `regcomp()`
9425 functions when using the interval notation `{\ (m,n)\}`; see Section 9.3.6 (on page 201).
9426 Value: 255

9427 RTS `{_POSIX_RTSIG_MAX}`
9428 The number of realtime signal numbers reserved for application use.
9429 Value: 8

9430 SEM `{_POSIX_SEM_NSEMS_MAX}`
9431 The number of semaphores that a process may have.
9432 Value: 256

9433 SEM `{_POSIX_SEM_VALUE_MAX}`
9434 The maximum value a semaphore may have.
9435 Value: 32 767

9436 RTS `{_POSIX_SIGQUEUE_MAX}`
9437 The number of queued signals that a process may send and have pending at the receiver(s)
9438 at any time.
9439 Value: 32

9440 `{_POSIX_SSIZE_MAX}`
9441 The value that can be stored in an object of type `ssize_t`.
9442 Value: 32 767

9443 `{_POSIX_STREAM_MAX}`
9444 The number of streams that one process can have open at one time.
9445 Value: 8

9446 SS|TSP `{_POSIX_SS_REPL_MAX}`
9447 The number of replenishment operations that may be simultaneously pending for a
9448 particular sporadic server scheduler.
9449 Value: 4

9450 { _POSIX_SYMLINK_MAX }
 9451 The number of bytes in a symbolic link.
 9452 Value: 255

9453 { _POSIX_SYMLOOP_MAX }
 9454 The number of symbolic links that can be traversed in the resolution of a path name in the
 9455 absence of a loop.
 9456 Value: 8

9457 THR { _POSIX_THREAD_DESTRUCTOR_ITERATIONS }
 9458 The number of attempts made to destroy a thread's thread-specific data values on thread
 9459 exit.
 9460 Value: 4

9461 THR { _POSIX_THREAD_KEYS_MAX }
 9462 The number of data keys per process.
 9463 Value: 128

9464 THR { _POSIX_THREAD_THREADS_MAX }
 9465 The number of threads per process.
 9466 Value: 64

9467 TMR { _POSIX_TIMER_MAX }
 9468 The per process number of timers.
 9469 Value: 32

9470 TRC { _POSIX_TRACE_EVENT_NAME_MAX }
 9471 The length in bytes of a trace event name.
 9472 Value: 30

9473 TRC { _POSIX_TRACE_NAME_MAX }
 9474 The length in bytes of a trace generation version string or a trace stream name.
 9475 Value: 8

9476 TRC { _POSIX_TRACE_SYS_MAX }
 9477 The number of trace streams that may simultaneously exist in the system.
 9478 Value: 8

9479 TRC { _POSIX_TRACE_USER_EVENT_MAX }
 9480 The number of user trace event type identifiers that may simultaneously exist in a traced
 9481 process, including the predefined user trace event
 9482 _POSIX_TRACE_UNNAMED_USER_EVENT.
 9483 Value: 32

9484 { _POSIX_TTY_NAME_MAX }
 9485 The size of the storage required for a terminal device name, in bytes, including the
 9486 terminating null.
 9487 Value: 9

9488 { _POSIX_TZNAME_MAX }
 9489 Maximum number of bytes supported for the name of a timezone (not of the TZ variable).
 9490 Value: 6

9491 **Note:** The length given by { _POSIX_TZNAME_MAX } does not include the quoting
 9492 characters mentioned in Section 8.3 (on page 192).

9493 { _POSIX2_BC_BASE_MAX }
 9494 Maximum *obase* values allowed by the *bc* utility.
 9495 Value: 99

9496 { _POSIX2_BC_DIM_MAX}
9497 Maximum number of elements permitted in an array by the *bc* utility.
9498 Value: 2 048

9499 { _POSIX2_BC_SCALE_MAX}
9500 Maximum *scale* value allowed by the *bc* utility.
9501 Value: 99

9502 { _POSIX2_BC_STRING_MAX}
9503 Maximum length of a string constant accepted by the *bc* utility.
9504 Value: 1 000

9505 { _POSIX2_CHARCLASS_NAME_MAX}
9506 Maximum number of bytes in a character class name.
9507 Value: 14

9508 { _POSIX2_COLL_WEIGHTS_MAX}
9509 Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order**
9510 keyword in the locale definition file; see Chapter 7 (on page 143).
9511 Value: 2

9512 { _POSIX2_EXPR_NEST_MAX}
9513 Maximum number of expressions that can be nested within parentheses by the *expr* utility.
9514 Value: 32

9515 { _POSIX2_LINE_MAX}
9516 Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
9517 standard input or another file), when the utility is described as processing text files. The
9518 length includes room for the trailing newline.
9519 Value: 2 048

9520 { _POSIX2_RE_DUP_MAX}
9521 Maximum number of repeated occurrences of a regular expression permitted when using
9522 the interval notation $\{m,n\}$; see Chapter 9 (on page 195).
9523 Value: 255

9524 XSI { _XOPEN_IOV_MAX}
9525 Maximum number of **iovec** structures that one process has available for use with *readv()* or
9526 *writenv()*.
9527 Value: 16
9528

9529 Numerical Limits

9530 The values in the following lists shall be defined in **<limits.h>** and are constant expressions
9531 XSI suitable for use in **#if** preprocessing directives. Moreover, except for {CHAR_BIT}, {DBL_DIG},
9532 {DBL_MAX}, {FLT_DIG}, {FLT_MAX}, {LONG_BIT}, {WORD_BIT}, and {MB_LEN_MAX}, the
9533 symbolic names are defined as expressions of the correct type.

9534 If the value of an object of type **char** is treated as a signed integer when used in an expression,
9535 the value of {CHAR_MIN} is the same as that of {SCHAR_MIN} and the value of {CHAR_MAX}
9536 is the same as that of {SCHAR_MAX}. Otherwise, the value of {CHAR_MIN} is 0 and the value
9537 of {CHAR_MAX} is the same as that of {UCHAR_MAX}.

9538 {CHAR_BIT}
9539 Number of bits in a type **char**.
9540 Minimum Acceptable Value: 8

9541 {CHAR_MAX}
 9542 Maximum value of type **char**.
 9543 Minimum Acceptable Value: {UCHAR_MAX} or {SCHAR_MAX}

9544 {INT_MAX}
 9545 Maximum value of an **int**.
 9546 Minimum Acceptable Value: 2 147 483 647

9547 XSI {LONG_BIT}
 9548 Number of bits in a **long**.
 9549 Minimum Acceptable Value: 32

9550 {LONG_MAX}
 9551 Maximum value of a **long**.
 9552 Minimum Acceptable Value: +2 147 483 647

9553 {MB_LEN_MAX}
 9554 Maximum number of bytes in a character, for any supported locale.
 9555 Minimum Acceptable Value: 1

9556 {SCHAR_MAX}
 9557 Maximum value of type **signed char**.
 9558 Minimum Acceptable Value: +127

9559 {SHRT_MAX}
 9560 Maximum value of type **short**.
 9561 Minimum Acceptable Value: +32 767

9562 {SSIZE_MAX}
 9563 Maximum value of an object of type **ssize_t**.
 9564 Minimum Acceptable Value: {_POSIX_SSIZE_MAX}

9565 {UCHAR_MAX}
 9566 Maximum value of type **unsigned char**.
 9567 Minimum Acceptable Value: 255

9568 {UINT_MAX}
 9569 Maximum value of type **unsigned**.
 9570 Minimum Acceptable Value: 4 294 967 295

9571 {ULONG_MAX}
 9572 Maximum value of type **unsigned long**.
 9573 Minimum Acceptable Value: 4 294 967 295

9574 {USHRT_MAX}
 9575 Maximum value for a type **unsigned short**.
 9576 Minimum Acceptable Value: 65 535

9577 XSI {WORD_BIT}
 9578 Number of bits in a word or type **int**.
 9579 Minimum Acceptable Value: 16

9580 {CHAR_MIN}
 9581 Minimum value of type **char**.
 9582 Maximum Acceptable Value: {SCHAR_MIN} or 0

9583 {INT_MIN}
 9584 Minimum value of type **int**.
 9585 Maximum Acceptable Value: -2 147 483 647

9586 {LONG_MIN}
 9587 Minimum value of type **long**.
 9588 Maximum Acceptable Value: -2 147 483 647

9589 {SCHAR_MIN}
 9590 Minimum value of type **signed char**.
 9591 Maximum Acceptable Value: -127

9592 {SHRT_MIN}
 9593 Minimum value of type **short**.
 9594 Maximum Acceptable Value: -32 767

9595 {LLONG_MIN}
 9596 Minimum value of type **long long**.
 9597 Maximum Acceptable Value: -9223372036854775807

9598 {LLONG_MAX}
 9599 Maximum value of type **long long**.
 9600 Minimum Acceptable Value: +9223372036854775807

9601 {ULLONG_MAX}
 9602 Maximum value of type **unsigned long long**.
 9603 Minimum Acceptable Value: 18446744073709551615

9604 **Other Invariant Values**

9605 XSI The following constants shall be defined on all implementations in <limits.h>:

9606 XSI {CHARCLASS_NAME_MAX}
 9607 Maximum number of bytes in a character class name.
 9608 Minimum Acceptable Value: 14

9609 XSI {NL_ARGMAX}
 9610 Maximum value of *digit* in calls to the *printf()* and *scanf()* functions.
 9611 Minimum Acceptable Value: 9

9612 XSI {NL_LANGMAX}
 9613 Maximum number of bytes in a *LANG* name.
 9614 Minimum Acceptable Value: 14

9615 XSI {NL_MSGMAX}
 9616 Maximum message number.
 9617 Minimum Acceptable Value: 32 767

9618 XSI {NL_NMAX}
 9619 Maximum number of bytes in an N-to-1 collation mapping.
 9620 Minimum Acceptable Value: ' * '

9621 XSI {NL_SETMAX}
 9622 Maximum set number.
 9623 Minimum Acceptable Value: 255

9624 XSI {NL_TEXTMAX}
 9625 Maximum number of bytes in a message string.
 9626 Minimum Acceptable Value: {_POSIX2_LINE_MAX}

9627 XSI {NZERO}
 9628 Default process priority.
 9629 Minimum Acceptable Value: 20

9630 XSI {TMP_MAX}
 9631 Minimum number of unique path names generated by *tmpnam()*. Maximum number of
 9632 times an application can call *tmpnam()* reliably. (LEGACY)
 9633 Minimum Acceptable Value: 10 000

9634 **APPLICATION USAGE**
 9635 None.

9636 **RATIONALE**
 9637 A request was made to reduce the value of {_POSIX_LINK_MAX} from the value of 8 specified
 9638 for it in the POSIX.1-1990 standard to 2. The standard developers decided to deny this request
 9639 for several reasons.

- 9640 • They wanted to avoid making any changes to the standard that could break conforming
 9641 applications, and the requested change could have that effect.
- 9642 • The use of multiple hard links to a file cannot always be replaced with use of symbolic links.
 9643 Symbolic links are semantically different from hard links in that they associate a path name
 9644 with another path name rather than a path name with a file. This has implications for access
 9645 control, file permanence, and transparency.
- 9646 • The original standard developers had considered the issue of allowing for implementations
 9647 that did not in general support hard links, and decided that this would reduce consensus on
 9648 the standard.

9649 Systems that support historical versions of the development option of the ISO POSIX-2 standard
 9650 retain the name {_POSIX2_RE_DUP_MAX} as an alias for {_POSIX_RE_DUP_MAX}.

9651 {PATH_MAX}
 9652 IEEE PASC Interpretation 1003.1 #15 addressed the inconsistency in the standard with the
 9653 definition of path name and the description of {PATH_MAX}, allowing application writers
 9654 to allocate either {PATH_MAX} or {PATH_MAX}+1 bytes. The inconsistency has been
 9655 removed by correction to the {PATH_MAX} definition to include the null character. With
 9656 this change, applications that previously allocated {PATH_MAX} bytes will continue to
 9657 succeed.

9658 {SYMLINK_MAX}
 9659 This symbol refers to space for data that is stored in the file system, as opposed to
 9660 {PATH_MAX} which is the length of a name that can be passed to a function. In some
 9661 existing implementations, the file names pointed to by symbolic links are stored in the
 9662 inodes of the links, so it is important that {SYMLINK_MAX} not be constrained to be as
 9663 large as {PATH_MAX}.

9664 **FUTURE DIRECTIONS**
 9665 None.

9666 **SEE ALSO**
 9667 The System Interfaces volume of IEEE Std. 1003.1-200x, *fpathconf()*, *pathconf()*, *sysconf()*

9668 **CHANGE HISTORY**
 9669 First released in Issue 1.

9670 **Issue 4**
 9671 A sentence is added to the DESCRIPTION indicating that names beginning with _POSIX can be
 9672 found in <unistd.h>.

9673 The {PASS_MAX} and {TMP_MAX} symbols are marked LEGACY.

9674 Use of the terms “bytes” and “characters” is rationalized to make it clear when the description is
9675 referring to either single-byte values or possibly multi-byte characters.

9676 {CHARCLASS_NAME_MAX} is added to the list of **Other Invariant Values** and marked as an
9677 extension.

9678 This entry is largely restructured to improve symbol grouping. A great many symbols, too
9679 numerous to mention, have also been added for alignment with the ISO POSIX-2 standard.

9680 The following changes are incorporated for alignment with the ISO C standard:

9681 • The constants {INT_MIN}, {LONG_MIN}, and {SHRT_MIN} are changed from values ending
9682 in 8 to ones ending in 7.

9683 • The {DBL_DIG}, {DBL_MAX}, {FLT_DIG}, and {FLT_MAX} symbols are marked both as
9684 extensions and LEGACY.

9685 • The {LONG_BIT} and {WORD_BIT} symbols are marked as extensions.

9686 • The {DBL_MIN} and {FLT_MIN} symbols are withdrawn.

9687 • Text introducing numerical limits now indicates that they are constant expressions suitable
9688 for use in #if preprocessing directives.

9689 The following change is incorporated for alignment with the FIPS requirements:

9690 • The minimum acceptable value for {NGROUPS_MAX} is changed from
9691 {_POSIX_NGROUPS_MAX} to 8. This is marked as an extension.

9692 **Issue 4, Version 2**

9693 The DESCRIPTION is revised for X/OPEN UNIX conformance as follows:

9694 • Under **Runtime Invariant Values**, {ATEXIT_MAX}, {IOV_MAX}, {PAGESIZE}, and
9695 {PAGE_SIZE} are added.

9696 • Under **Minimum Values**, {_XOPEN_IOV_MAX} is added.

9697 **Issue 5**

9698 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
9699 Threads Extension.

9700 {FILESIZEBITS} added for the Large File Summit extensions.

9701 The minimum acceptable values for {INT_MAX}, {INT_MIN}, and {UINT_MAX} are changed to
9702 make 32-bit values the minimum requirement.

9703 The entry is restructured to improve readability.

9704 **Issue 6**

9705 The Open Group corrigenda item U033/4 has been applied. The wording is made clear for
9706 {CHAR_MIN}, {INT_MIN}, {LONG_MIN}, {SCHAR_MIN}, and {SHRT_MIN} that these are
9707 maximum acceptable values.

9708 The following new requirements on POSIX implementations derive from alignment with the
9709 Single UNIX Specification:

9710 • The minimum value for {CHILD_MAX} is 25. This is a FIPS requirement.

9711 • The minimum value for {OPEN_MAX} is 20. This is a FIPS requirement.

9712 • The minimum value for {NGROUPS_MAX} is 8. This is also a FIPS requirement.

9713 Symbolic constants are added for {_POSIX_SYMLINK_MAX}, {_POSIX_SYMLOOP_MAX},
9714 {_POSIX_RE_DUP_MAX}, {RE_DUP_MAX}, {SYMLOOP_MAX}, and {SYMLINK_MAX}.

9715 The following values are added for alignment with IEEE Std. 1003.1d-1999:

9716 {_POSIX_SS_REPL_MAX}
9717 {SS_REPL_MAX}
9718 {POSIX_ALLOC_SIZE_MIN}
9719 {POSIX_REC_INCR_XFER_SIZE}
9720 {POSIX_REC_MAX_XFER_SIZE}
9721 {POSIX_REC_MIN_XFER_SIZE}
9722 {POSIX_REC_XFER_ALIGN}

9723 Reference to CLOCK_MONOTONIC is added in the description of {_POSIX_CLOCKRES_MIN}
9724 for alignment with IEEE Std. 1003.1j-2000.

9725 The constants {LLONG_MIN}, {LLONG_MAX}, and {ULLONG_MAX} are added for alignment
9726 with the ISO/IEC 9899:1999 standard.

9727 The following values are added for alignment with IEEE Std. 1003.1q-2000:
9728 {_POSIX_TRACE_EVENT_NAME_MAX}, {_POSIX_TRACE_NAME_MAX},
9729 {_POSIX_TRACE_SYS_MAX}, {_POSIX_TRACE_USER_EVENT_MAX},
9730 {TRACE_EVENT_NAME_MAX}, {TRACE_NAME_MAX}, {TRACE_SYS_MAX},
9731 {TRACE_USER_EVENT_MAX}

9732 **NAME**

9733 locale.h — category macros

9734 **SYNOPSIS**

9735 #include <locale.h>

9736 **DESCRIPTION**

9737 cx The functionality described on this reference page extends the ISO C standard. Applications
9738 shall define the appropriate feature test macro (see the System Interfaces volume of
9739 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
9740 symbols in this header.

9741 The <locale.h> header shall provide a definition for structure **lconv**, which shall include at least
9742 the following members. (See the definitions of *LC_MONETARY* in the Section 7.3.3 (on page
9743 163), and Section 7.3.4 (on page 166).)

9744 char *currency_symbol
9745 char *decimal_point
9746 char frac_digits
9747 char *grouping
9748 char *int_curr_symbol
9749 char int_frac_digits
9750 char int_n_cs_precedes
9751 char int_n_sep_by_space
9752 char int_n_sign_posn
9753 char int_p_cs_precedes
9754 char int_p_sep_by_space
9755 char int_p_sign_posn
9756 char *mon_decimal_point
9757 char *mon_grouping
9758 char *mon_thousands_sep
9759 char *negative_sign
9760 char n_cs_precedes
9761 char n_sep_by_space
9762 char n_sign_posn
9763 char *positive_sign
9764 char p_cs_precedes
9765 char p_sep_by_space
9766 char p_sign_posn
9767 char *thousands_sep

9768 The <locale.h> header shall define NULL (as defined in <stddef.h>) and at least the following as
9769 macros:

9770 *LC_ALL*
9771 *LC_COLLATE*
9772 *LC_CTYPE*
9773 *LC_MESSAGES*
9774 *LC_MONETARY*
9775 *LC_NUMERIC*
9776 *LC_TIME*

9777 which shall expand to distinct integral constant expressions, for use as the first argument to the
9778 *setlocale()* function.

9779 Additional macro definitions, beginning with the characters *LC_* and an uppercase letter, may
9780 also be given here.

9781 The following shall be declared as functions and may also be defined as macros. Function
9782 prototypes shall be provided for use with an ISO C standard compiler.

```
9783 struct lconv *localeconv (void);  
9784 char    setlocale(int, const char *);
```

9785 **APPLICATION USAGE**

9786 None.

9787 **RATIONALE**

9788 None.

9789 **FUTURE DIRECTIONS**

9790 None.

9791 **SEE ALSO**

9792 The System Interfaces volume of IEEE Std. 1003.1-200x, *localeconv()*, *setlocale()*, Chapter 8 (on
9793 page 187)

9794 **CHANGE HISTORY**

9795 First released in Issue 3.

9796 Entry included for alignment with the ISO C standard.

9797 **Issue 4**

9798 The following changes are incorporated for alignment with the ISO C standard:

- 9799 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 9800 • The definition of **struct lconv** is added.
- 9801 • A reference to <stddef.h> is added for the definition of NULL.

9802 **Issue 6**

9803 The **lconv** structure is expanded with new members (**int_n_cs_precedes**, **int_n_sep_by_space**,
9804 **int_n_sign_posn**, **int_p_cs_precedes**, **int_p_sep_by_space**, and **int_p_sign_posn**) for alignment
9805 with the ISO/IEC 9899:1999 standard.

9806 **NAME**9807 `math.h` — mathematical declarations9808 **SYNOPSIS**9809 `#include <math.h>`9810 **DESCRIPTION**

9811 `CX` The functionality described on this reference page extends the ISO C standard. Applications
 9812 shall define the appropriate feature test macro (see the System Interfaces volume of
 9813 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 9814 symbols in this header.

9815 The **<math.h>** header shall include definitions for at least the following types:

9816 **float_t** A floating type at least as wide as **float**.

9817 **double_t** A floating type at least as wide as **double**, and at least as wide as **float_t**.

9818 If `FLT_EVAL_METHOD` equals 0, **float_t** and **double_t** shall be **float** and **double**, respectively; if
 9819 `FLT_EVAL_METHOD` equals 1, they shall both be **double**; if `FLT_EVAL_METHOD` equals 2,
 9820 they shall both be **long double**; for other values of `FLT_EVAL_METHOD`, they are otherwise
 9821 implementation-defined.

9822 The **<math.h>** header shall define the following macros, where **real-floating** indicates that the
 9823 argument shall be an expression of **real-floating** type:

```

9824 int fpclassify(real-floating x);
9825 int isfinite(real-floating x);
9826 int isinf(real-floating x);
9827 int isnan(real-floating x);
9828 int isnormal(real-floating x);
9829 int signbit(real-floating x);
9830 int isgreater(real-floating x, real-floating y);
9831 int isgreaterequal(real-floating x, real-floating y);
9832 int isless(real-floating x, real-floating y);
9833 int islessequal(real-floating x, real-floating y);
9834 int islessgreater(real-floating x, real-floating y);
9835 int isunordered(real-floating x, real-floating y);

```

9836 The **<math.h>** header shall provide for the following constants. The values are of type **double**
 9837 and are accurate within the precision of the **double** type.

9838	<code>XSI</code>	<code>M_E</code>	Value of e
9839		<code>M_LOG2E</code>	Value of $\log_2 e$
9840		<code>M_LOG10E</code>	Value of $\log_{10} e$
9841		<code>M_LN2</code>	Value of $\log_e 2$
9842		<code>M_LN10</code>	Value of $\log_e 10$
9843		<code>M_PI</code>	Value of π
9844		<code>M_PI_2</code>	Value of $\pi/2$
9845		<code>M_PI_4</code>	Value of $\pi/4$
9846		<code>M_1_PI</code>	Value of $1/\pi$
9847		<code>M_2_PI</code>	Value of $2/\pi$

9848	M_2_SQRTPI	Value of $2\sqrt{\pi}$
9849	M_SQRT2	Value of $\sqrt{2}$
9850	M_SQRT1_2	Value of $1\sqrt{2}$
9851	The header shall define the following symbolic constants:	
9852	XSI MAXFLOAT	Value of maximum non-infinite single-precision floating-point number.
9853	HUGE_VAL	A positive double expression, not necessarily representable as a float . Used as an error value returned by the mathematics library. HUGE_VAL evaluates to $+\infty$ on systems supporting IEEE Std. 754-1985.
9854		
9855		
9856	HUGE_VALF	A positive float constant expression. Used as an error value returned by the mathematics library. HUGE_VALF evaluates to $+\infty$ on systems supporting IEEE Std. 754-1985.
9857		
9858		
9859	HUGE_VALD	A positive long double constant expression. Used as an error value returned by the mathematics library. HUGE_VALD evaluates to $+\infty$ on systems supporting IEEE Std. 754-1985.
9860		
9861		
9862	INFINITY	A constant expression of type float representing positive or unsigned infinity, if available; else a positive constant of type float that overflows at translation time.
9863		
9864		
9865	NAN	A constant expression of type float representing a quiet NaN. This symbolic constant is only defined if the implementation supports quiet NaNs for the float type.
9866		
9867		
9868	The following macros shall be defined for number classification. They represent the mutually-exclusive kinds of floating-point values. They expand to integer constant expressions with distinct values. Additional implementation-defined floating-point classifications, with macro definitions beginning with FP_ and an uppercase letter, may also be specified by the implementation.	
9869		
9870		
9871		
9872		
9873	FP_INFINITE	
9874	FP_NAN	
9875	FP_NORMAL	
9876	FP_SUBNORMAL	
9877	FP_ZERO	
9878	The following macros are optional. If FP_FAST_FMA is defined, it shall indicate that the <i>fma()</i> function generally executes about as fast as, or faster than, a multiply and an add of double operands.	
9879		
9880		
9881	FP_FAST_FMA	
9882	FP_FAST_FMAF	
9883	FP_FAST_FMAL	
9884	FP_FAST_FMAF and FP_FAST_FMAL are, respectively, float and long double analogs of FP_FAST_FMA.	
9885		
9886	The following macros shall expand to integer constant expressions whose values are returned by <i>ilogb(x)</i> if <i>x</i> is zero or NaN, respectively. The value of FP_ILOGB0 shall be either {INT_MIN} or $-\{INT_MAX\}$. The value of FP_ILOGBNAN shall be either {INT_MAX} or {INT_MIN}.	
9887		
9888		
9889	FP_ILOGB0	
9890	FP_ILOGBNAN	

9891 The following macros shall expand to the integer constants 1 and 2, respectively;

9892 MATH_ERRNO
9893 MATH_ERREXCEPT

9894 The following macro shall expand to an expression that has type **int** and the value
9895 MATH_ERRNO, MATH_ERREXCEPT, or the bitwise-inclusive OR of both. The value of
9896 *math_errhandling* is constant for the duration of the program. It is unspecified whether
9897 *math_errhandling* is a macro or an identifier with external linkage. If a macro definition is
9898 suppressed or a program defines an identifier with the name *math_errhandling*, the behavior is
9899 undefined. If the expression *math_errhandling* & MATH_ERREXCEPT can be non-zero, the
9900 implementation shall define the macros FE_DIVBYZERO, FE_INVALID, and FE_OVERFLOW in
9901 **<fenv.h>**.

9902 *math_errhandling*

9903 The following shall be declared as functions and may also be defined as macros. Function
9904 prototypes shall be provided for use with an ISO C standard compiler.

9905 double acos(double);
9906 float acosf(float);
9907 XSI double acosh(double);
9908 float acoshf(float);
9909 long double acoshl(long double);
9910 long double acosl(long double);
9911 double asin(double);
9912 float asinf(float);
9913 XSI double asinh(double);
9914 float asinhf(float);
9915 long double asinhl(long double);
9916 long double asinl(long double);
9917 double atan(double);
9918 double atan2(double, double);
9919 float atan2f(float, float);
9920 long double atan2l(long double, long double);
9921 float atanf(float);
9922 XSI double atanh(double);
9923 float atanhf(float);
9924 long double atanh1(long double);
9925 long double atanl(long double);
9926 XSI double cbrt(double);
9927 float cbrtf(float);
9928 long double cbrtl(long double);
9929 double ceil(double);
9930 float ceilf(float);
9931 long double ceill(long double);
9932 double copysign(double, double);
9933 float copysignf(float, float);
9934 long double copysignl(long double, long double);
9935 double cos(double);
9936 float cosf(float);
9937 double cosh(double);
9938 float coshf(float);
9939 long double coshl(long double);
9940 long double cosl(long double);


```

9941 XSI    double    erf(double);
9942        double    erfc(double);
9943        float    erfcf(float);
9944        long double erfcl(long double);
9945        float    erff(float);
9946        long double erfl(long double);
9947        double    exp(double);
9948        double    exp2(double);
9949        float    exp2f(float);
9950        long double exp2l(long double);
9951        float    expf(float);
9952        long double expl(long double);
9953 XSI    double    expm1(double);
9954        float    expm1f(float);
9955        long double expm1l(long double);
9956        double    fabs(double);
9957        float    fabsf(float);
9958        long double fabsl(long double);
9959        double    fdim(double, double);
9960        float    fdimf(float, float);
9961        long double fdiml(long double, long double);
9962        double    floor(double);
9963        float    floorf(float);
9964        long double floorl(long double);
9965        double    fma(double, double, double);
9966        float    fmaf(float, float, float);
9967        long double fmal(long double, long double, long double);
9968        double    fmax(double, double);
9969        float    fmaxf(float, float);
9970        long double fmaxl(long double, long double);
9971        double    fmin(double, double);
9972        float    fminf(float, float);
9973        long double fminl(long double, long double);
9974        double    fmod(double, double);
9975        float    fmodf(float, float);
9976        long double fmodl(long double, long double);
9977        double    frexp(double, int *);
9978        float    frexpf(float value, int *);
9979        long double frexpl(long double value, int *);
9980 XSI    double    hypot(double, double);
9981        float    hypotf(float, float);
9982        long double hypotl(long double, long double);
9983 XSI    int    ilogb(double);
9984        int    ilogbf(float);
9985        int    ilogbl(long double);
9986 XSI    int    isnan(double);
9987        double    j0(double);
9988        double    j1(double);
9989        double    jn(int, double);
9990        double    ldexp(double, int);
9991        float    ldexpf(float, int);
9992        long double ldexpl(long double, int);

```

```
9993 XSI double lgamma(double);
9994 float lgammaf(float);
9995 long double lgammal(long double);
9996 double log(double);
9997 double log10(double);
9998 float log10f(float);
9999 long double log10l(long double);
10000 XSI double log1p(double);
10001 float log1pf(float);
10002 long double log1pl(long double);
10003 double log2(double);
10004 float log2f(float);
10005 long double log2l(long double);
10006 XSI double logb(double);
10007 float logbf(float);
10008 long double logbl(long double);
10009 float logf(float);
10010 long double logl(long double);
10011 long long llrint(double);
10012 long long llrintf(float);
10013 long long llrintl(long double);
10014 long long llround(double);
10015 long long llroundf(float);
10016 long long llroundl(long double);
10017 long lrint(double);
10018 long lrintf(float);
10019 long lrintl(long double);
10020 long lround(double);
10021 long lroundf(float);
10022 long lroundl(long double);
10023 double modf(double, double *);
10024 float modff(float, float *);
10025 long double modfl(long double, long double *);
10026 double nan(const char *);
10027 float nanf(const char *);
10028 long double nanl(const char *);
10029 double nearbyint(double);
10030 float nearbyintf(float);
10031 long double nearbyintl(long double);
10032 XSI double nextafter(double, double);
10033 float nextafterf(float, float);
10034 long double nextafterl(long double, long double);
10035 double nexttoward(double, long double);
10036 float nexttowardf(float, long double);
10037 long double nexttowardl(long double, long double);
10038 double pow(double, double);
10039 float powf(float, float);
10040 long double powl(long double, long double);
10041 XSI double remainder(double, double);
10042 float remainderf(float, float);
10043 long double remainderl(long double, long double);
10044 double remquo(double, double, int *);
```

```

10045     float      remquof(float, float, int *);
10046     long double remquol(long double, long double, int *);
10047 XSI   double      rint(double);
10048     float      rintf(float);
10049     long double rintl(long double);
10050     double     round(double);
10051     float      roundf(float);
10052     long double roundl(long double);
10053 XSI   double     scalb(double, double);
10054     double     scalbln(double, long);
10055     float      scalblnf(float, long);
10056     long double scalblnl(long double, long);
10057     double     scalbn(double, int);
10058     float      scalbnf(float, int);
10059     long double scalbnl(long double, int);
10060     double     sin(double);
10061     float      sinf(float);
10062     double     sinh(double);
10063     float      sinhf(float);
10064     long double sinhl(long double);
10065     long double sinl(long double);
10066     double     sqrt(double);
10067     float      sqrtf(float);
10068     long double sqrtl(long double);
10069     double     tan(double);
10070     float      tanf(float);
10071     double     tanh(double);
10072     float      tanhf(float);
10073     long double tanhl(long double);
10074     long double tanl(long double);
10075     double     tgamma(double);
10076     float      tgammaf(float);
10077     long double tgamma_l(long double);
10078     double     trunc(double);
10079     float      truncf(float);
10080     long double trunc_l(long double);
10081 XSI   double     y0(double);
10082     double     y1(double);
10083     double     yn(int, double);
10084

```

10085 **The following external variable shall be defined:**

```

10086 XSI   extern int signgam;
10087

```

10088 APPLICATION USAGE

10089 The FP_CONTRACT pragma can be used to allow (if the state is on) or disallow (if the state is
 10090 off) the implementation to contract expressions. Each pragma can occur either outside external
 10091 declarations or preceding all explicit declarations and statements inside a compound statement.
 10092 When outside external declarations, the pragma takes effect from its occurrence until another
 10093 FP_CONTRACT pragma is encountered, or until the end of the translation unit. When inside a
 10094 compound statement, the pragma takes effect from its occurrence until another FP_CONTRACT
 10095 pragma is encountered (including within a nested compound statement), or until the end of the
 10096 compound statement; at the end of a compound statement the state for the pragma is restored to
 10097 its condition just before the compound statement. If this pragma is used in any other context, the
 10098 behavior is undefined. The default state (on or off) for the pragma is implementation-defined.

10099 RATIONALE

10100 Before the ISO/IEC 9899:1999 standard, the math library was defined only for the floating type
 10101 **double**. All the names formed by appending 'f' or 'l' to a name in <math.h> were reserved
 10102 to allow for the definition of **float** and **long double** libraries; and the ISO/IEC 9899:1999
 10103 standard provides for all three versions of math functions.

10104 The functions *ecvt()*, *fcvt()*, and *gcvt()* have been dropped from the ISO C standard since their
 10105 capability is available through *sprintf()*. These are provided on XSI-conformant systems
 10106 supporting the Legacy Option Group.

10107 FUTURE DIRECTIONS

10108 None.

10109 SEE ALSO

10110 The System Interfaces volume of IEEE Std. 1003.1-200x, *acos()*, *acosh()*, *asin()*, *atan()*, *atan2()*,
 10111 *cbrt()*, *ceil()*, *cos()*, *cosh()*, *erf()*, *exp()*, *expm1()*, *fabs()*, *floor()*, *fmod()*, *frexp()*, *hypot()*, *ilogb()*,
 10112 *isnan()*, *j0()*, *ldexp()*, *lgamma()*, *log()*, *log10()*, *log1p()*, *logb()*, *modf()*, *nextafter()*, *pow()*,
 10113 *remainder()*, *rint()*, *scalb()*, *sin()*, *sinh()*, *sqrt()*, *tan()*, *tanh()*, *y0()*

10114 CHANGE HISTORY

10115 First released in Issue 1.

10116 Issue 4

10117 The constants M_E and MAXFLOAT are marked as extensions.

10118 The functions declared in this header are subdivided into those defined in the ISO C standard,
 10119 and those defined only by The Open Group. Functions in the latter group are marked as
 10120 extensions, as is the external variable *signgam*.

10121 The following changes are incorporated for alignment with the ISO C standard:

- 10122 • The description of HUGE_VAL is changed to indicate that this value is not necessarily
- 10123 representable as a **float**.
- 10124 • The function declarations in this header are expanded to full ISO C standard prototypes.

10125 Issue 4, Version 2

10126 The following change is incorporated for X/OPEN UNIX conformance:

- 10127 • The *acosh()*, *asinh()*, *atanh()*, *cbrt()*, *expm1()*, *ilogb()*, *log1p()*, *logb()*, *nextafter()*, *remainder()*,
- 10128 *rint()*, and *scalb()* functions are added to the list of functions declared in this header.

10129 Issue 6

10130 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

10131 **NAME**

10132 monetary.h — monetary types

10133 **SYNOPSIS**

10134 XSI #include <monetary.h>

10135

10136 **DESCRIPTION**10137 The <monetary.h> header shall define the following data types through **typedef**:10138 **size_t** As described in <stddef.h>.10139 **ssize_t** As described in <sys/types.h>.10140 The following shall be declared as a function and may also be defined as a macro. Function
10141 prototypes shall be provided for use with an ISO C standard compiler.10142 **ssize_t** strfmon(char *restrict, size_t, const char *restrict, ...);10143 **APPLICATION USAGE**

10144 None.

10145 **RATIONALE**

10146 None.

10147 **FUTURE DIRECTIONS**

10148 None.

10149 **SEE ALSO**10150 The System Interfaces volume of IEEE Std. 1003.1-200x, *strfmon()*10151 **CHANGE HISTORY**

10152 First released in Issue 4.

10153 **Issue 6**10154 The **restrict** keyword is added to the prototype for *strfmon()*.

10155 **NAME**10156 mqqueue.h — message queues (**REALTIME**)10157 **SYNOPSIS**

10158 MSG #include <mqqueue.h>

10159

10160 **DESCRIPTION**10161 The <mqqueue.h> header shall define the **mqd_t** type, which is used for message queue
10162 descriptors. This is not an array type.10163 The <mqqueue.h> header shall define the **sigevent** structure (as described in <signal.h>) and the
10164 **mq_attr** structure, which is used in getting and setting the attributes of a message queue.
10165 Attributes are initially set when the message queue is created. An **mq_attr** structure shall have at
10166 least the following fields:

10167	long	mq_flags	Message queue flags.
10168	long	mq_maxmsg	Maximum number of messages.
10169	long	mq_msgsize	Maximum message size.
10170	long	mq_curmsgs	Number of messages currently queued.

10171 The following shall be declared as functions and may also be declared as macros. Function
10172 prototypes shall be provided for use with an ISO C standard compiler.

```

10173 int      mq_close(mqd_t);
10174 int      mq_getattr(mqd_t, struct mq_attr *);
10175 int      mq_notify(mqd_t, const struct sigevent *);
10176 mqd_t    mq_open(const char *, int, ...);
10177 ssize_t  mq_receive(mqd_t, char *, size_t, unsigned *);
10178 int      mq_send(mqd_t, const char *, size_t, unsigned);
10179 int      mq_setattr(mqd_t, const struct mq_attr *restrict,
10180                  struct mq_attr *restrict);
10181 TMO int      mq_timedreceive(mqd_t, char *restrict, size_t,
10182                          unsigned *restrict, const struct timespec *restrict);
10183 int      mq_timedsend(mqd_t, const char *, size_t, unsigned,
10184                      const struct timespec *);
10185 int      mq_unlink(const char *);

```

10186 **Notes to Reviewers**10187 *This section with side shading will not appear in the final copy. - Ed.*10188 D3, XBD, ERN 163: The return type from mq_timedreceive() should be ssize_t and not int. An
10189 interpretation should be filed against .1d to bring this change into scope.10190 Inclusion of the <mqqueue.h> header may make visible symbols defined in the headers <fcntl.h>,
10191 <signal.h>, <sys/types.h>, and <time.h>.

10192 **APPLICATION USAGE**

10193 None.

10194 **RATIONALE**

10195 None.

10196 **FUTURE DIRECTIONS**

10197 None.

10198 **SEE ALSO**

10199 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>, the System Interfaces volume of
10200 IEEE Std. 1003.1-200x, *mq_close()*, *mq_getattr()*, *mq_notify()*, *mq_open()*, *mq_receive()*, *mq_send()*,
10201 *mq_setattr()*, *mq_timedreceive()*, *mq_timedsend()*, *mq_unlink()*

10202 **CHANGE HISTORY**

10203 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

10204 **Issue 6**

10205 The <mqqueue.h> header is marked as part of the Message Passing option. |

10206 The *mq_timedreceive()* and *mq_timedsend()* functions are added for alignment with |
10207 IEEE Std. 1003.1d-1999. |

10208 The **restrict** keyword is added to the prototypes for *mq_setattr()* and *mq_timedreceive()*. |

10209 **NAME**

10210 ndbm.h — definitions for ndbm database operations

10211 **SYNOPSIS**10212 XSI `#include <ndbm.h>`

10213

10214 **DESCRIPTION**10215 The **<ndbm.h>** header shall define the **datum** type as a structure that includes at least the
10216 following members:10217 `void *dptr` A pointer to the application's data.10218 `size_t dsize` The size of the object pointed to by *dptr*.10219 The `size_t` type shall be defined through **typedef** as described in **<stddef.h>**.10220 The **<ndbm.h>** header shall define the **DBM** type through **typedef**.10221 The following constants shall be defined as possible values for the *store_mode* argument to
10222 *dbm_store()*:10223 **DBM_INSERT** Insertion of new entries only.10224 **DBM_REPLACE** Allow replacing existing entries.10225 The following shall be declared as functions and may also be defined as macros. Function
10226 prototypes shall be provided for use with an ISO C standard compiler.10227 `int dbm_clearerr(DBM *);`10228 `void dbm_close(DBM *);`10229 `int dbm_delete(DBM *, datum);`10230 `int dbm_error(DBM *);`10231 `datum dbm_fetch(DBM *, datum);`10232 `datum dbm_firstkey(DBM *);`10233 `datum dbm_nextkey(DBM *);`10234 `DBM *dbm_open(const char *, int, mode_t);`10235 `int dbm_store(DBM *, datum, datum, int);`10236 The `mode_t` type shall be defined through **typedef** as described in **<sys/types.h>**.10237 **APPLICATION USAGE**

10238 None.

10239 **RATIONALE**

10240 None.

10241 **FUTURE DIRECTIONS**

10242 None.

10243 **SEE ALSO**10244 The System Interfaces volume of IEEE Std. 1003.1-200x, *dbm_clearerr()*10245 **CHANGE HISTORY**

10246 First released in Issue 4, Version 2.

10247 **Issue 5**10248 References to the definitions of `size_t` and `mode_t` are added to the DESCRIPTION.

10249 **NAME**

10250 net/if.h — sockets local interfaces

10251 **SYNOPSIS**

10252 #include <net/if.h>

10253 **DESCRIPTION**10254 The <net/if.h> header shall define the **if_nameindex** structure that includes at least the
10255 following members:10256 unsigned if_index Numeric index of the interface.
10257 char *if_name Null-terminated name of the interface.10258 The <net/if.h> header shall define the following macro for the length of a buffer containing an
10259 interface name (including the terminating NULL character):10260 **IF_NAMESIZE** Interface name length.10261 The following shall be declared as functions, and may also be defined as macros. Function
10262 prototypes shall be provided for use with an ISO C standard compiler.10263 unsigned if_nametoindex(const char*);
10264 char *if_indextoname(unsigned, char*);
10265 struct if_nameindex *if_nameindex(void);
10266 void if_freenameindex(struct if_nameindex*);10267 **APPLICATION USAGE**

10268 None.

10269 **RATIONALE**

10270 None.

10271 **FUTURE DIRECTIONS**

10272 None.

10273 **SEE ALSO**10274 The System Interfaces volume of IEEE Std. 1003.1-200x, *if_freenameindex()*, *if_indextoname()*,
10275 *if_nameindex()*, *if_nametoindex()*10276 **CHANGE HISTORY**

10277 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10278 **NAME**

10279 netdb.h — definitions for network database operations

10280 **SYNOPSIS**

10281 #include <netdb.h>

10282 **DESCRIPTION**10283 The <netdb.h> header may make available the **in_port_t** type and the **in_addr_t** type as defined
10284 in <netinet/in.h>.10285 The <netdb.h> header shall define the **hostent** structure that includes at least the following
10286 members:

10287	char	*h_name	Official name of the host.
10288	char	**h_aliases	A pointer to an array of pointers to
10289			alternative host names, terminated by a
10290			null pointer.
10291	int	h_addrtype	Address type.
10292	int	h_length	The length, in bytes, of the address.
10293	char	**h_addr_list	A pointer to an array of pointers to network
10294			addresses (in network byte order) for the host,
10295			terminated by a null pointer.

10296 The <netdb.h> header shall define the **netent** structure that includes at least the following
10297 members:

10298	char	*n_name	Official, fully-qualified (including the
10299			domain) name of the host.
10300	char	**n_aliases	A pointer to an array of pointers to
10301			alternative network names, terminated by a
10302			null pointer.
10303	int	n_addrtype	The address type of the network.
10304	uint32_t	n_net	The network number, in host byte order.

10305 The **uint32_t** type shall be defined as described in <inttypes.h>.10306 The <netdb.h> header shall define the **protoent** structure that includes at least the following
10307 members:

10308	char	*p_name	Official name of the protocol.
10309	char	**p_aliases	A pointer to an array of pointers to
10310			alternative protocol names, terminated by
10311			a null pointer.
10312	int	p_proto	The protocol number.

10313 The <netdb.h> header shall define the **servent** structure that includes at least the following
10314 members:

10315	char	*s_name	Official name of the service.
10316	char	**s_aliases	A pointer to an array of pointers to
10317			alternative service names, terminated by
10318			a null pointer.
10319	int	s_port	The port number at which the service
10320			resides, in network byte order.
10321	char	*s_proto	The name of the protocol to use when
10322			contacting the service.

10323 The <netdb.h> header shall define the IPPORT_RESERVED macro with the value of the highest
 10324 reserved Internet port number.

10325 When the <netdb.h> header is included, *h_errno* shall be available as a modifiable *l*-value of type
 10326 **int**. It is unspecified whether *h_errno* is a macro or an identifier declared with external linkage.

10327 The <netdb.h> header shall define the following macros for use as error values for
 10328 *gethostbyaddr()*, *gethostbyname()*, *getipnodebyaddr()*, and *getipnodebyname()*:

```
10329     HOST_NOT_FOUND
10330     NO_DATA
10331     NO_RECOVERY
10332     TRY_AGAIN
```

10333 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer
 10334 constants, for use in the *flags* argument of *getipnodebyname()*:

10335 IP6 **AI_V4MAPPED** IPv4-mapped IPv6 addresses are acceptable.

10336 **AI_ALL** Return all addresses: IPv6 and IPv4-mapped IPv6.

10337 **AI_ADDRCONFIG**
 10338 Return addresses depending on what source addresses are configured.

10339 The <netdb.h> header shall define the **AI_DEFAULT** macro, which evaluates to the logical OR of
 10340 **AI_V4MAPPED** and **AI_ADDRCONFIG**.

10341 **Address Information Structure**

10342 The <netdb.h> header shall define the **addrinfo** structure that includes at least the following
 10343 members:

10344	int	ai_flags	Input flags.
10345	int	ai_family	Address family of socket.
10346	int	ai_socktype	Socket type.
10347	int	ai_protocol	Protocol of socket.
10348	socklen_t	ai_addrlen	Length of socket address.
10349	struct sockaddr	*ai_addr	Socket address of socket.
10350	char	*ai_canonname	Canonical name of service location.
10351	struct addrinfo	*ai_next	Pointer to next in list.

10352 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer
 10353 constants for use in the *flags* field of the **addrinfo** structure:

10354 **AI_PASSIVE** Socket address is intended for *bind()*.

10355 **AI_CANONNAME**
 10356 Request for canonical name.

10357 **AI_NUMERICHOST**
 10358 Return numeric host address as name.

10359 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer
 10360 constants for use in the *flags* argument to *getnameinfo()*:

10361 **NI_NOFQDN** Only the nodename portion of the FQDN is returned for local hosts.

10362 **NI_NUMERICHOST**
 10363 The numeric form of the node's address is returned instead of its name.

10364 NI_NAMEREQD Return an error if the node's name cannot be located in the database.

10365 NI_NUMERICSERV

10366 The numeric form of the service address is returned instead of its name.

10367 NI_DGRAM Indicates that the service is a datagram service (SOCK_DGRAM).

10368 **Address Information Errors**

10369 The <netdb.h> header shall define the following macros for use as error values for *getaddrinfo()*

10370 and *getnameinfo()*:

10371 EAI_AGAIN The name could not be resolved at this time. Future attempts may succeed.

10372 EAI_BADFLAGS The flags had an invalid value.

10373 EAI_FAIL A non-recoverable error occurred.

10374 EAI_FAMILY The address family was not recognized or the address length was invalid for

10375 the specified family.

10376 EAI_MEMORY There was a memory allocation failure.

10377 EAI_NONAME The name does not resolve for the supplied parameters.

10378 NI_NAMEREQD is set and the host's name cannot be located, or both

10379 *nodename* and *servname* were null.

10380 EAI_SERVICE The service passed was not recognized for the specified socket type.

10381 EAI_SOCKTYPE The intended socket type was not recognized.

10382 EAI_SYSTEM A system error occurred. The error code can be found in *errno*.

10383 The following shall be declared as functions, and may also be defined as macros. Function

10384 prototypes shall be provided for use with an ISO C standard compiler.

10385 void endhostent(void);

10386 void endnetent(void);

10387 void endprotoent(void);

10388 void endservent(void);

10389 void freeaddrinfo(struct addrinfo *);

10390 void freehostent(struct hostent *);

10391 char *gai_strerror(int);

10392 int getaddrinfo(const char *, const char *,

10393 const struct addrinfo *, struct addrinfo **);

10394 struct hostent *gethostbyaddr(const void *, socklen_t, int);

10395 struct hostent *gethostbyname(const char *);

10396 struct hostent *gethostent(void);

10397 struct hostent *getipnodebyaddr(const void *restrict, socklen_t, int,

10398 int *restrict);

10399 struct hostent *getipnodebyname(const char *, int, int, int *);

10400 int getnameinfo(const struct sockaddr *, socklen_t,

10401 char *, socklen_t, char *, socklen_t, unsigned);

10402 struct netent *getnetbyaddr(uint32_t, int);

10403 struct netent *getnetbyname(const char *);

10404 struct netent *getnetent(void);

10405 struct protoent *getprotobyname(const char *);

10406 struct protoent *getprotobynumber(int);

10407 struct protoent *getprotoent(void);

```
10408     struct servent     *getservbyname(const char *, const char *);
10409     struct servent     *getservbyport(int, const char *);
10410     struct servent     *getservent(void);
10411     void                sethostent(int);
10412     void                setnetent(int);
10413     void                setprotoent(int);
10414     void                setservent(int);

10415     The type socklen_t shall be defined through typedef as described in <sys/socket.h>.

10416     Inclusion of the <netdb.h> header may also make visible all symbols from <netinet/in.h> and
10417     <inttypes.h>.

10418 APPLICATION USAGE
10419     None.

10420 RATIONALE
10421     None.

10422 FUTURE DIRECTIONS
10423     None.

10424 SEE ALSO
10425     <netinet/in.h>, <inttypes.h>, <sys/socket.h>, the System Interfaces volume of
10426     IEEE Std. 1003.1-200x, bind(), endhostent(), endnetent(), endprotoent(), endservent(), getaddrinfo(),
10427     getnameinfo()

10428 CHANGE HISTORY
10429     First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10430     The restrict keyword is added to the prototype for getipnodebyaddr().
```

10431 NAME

10432 netinet/in.h — Internet protocol family

10433 SYNOPSIS

10434 #include <netinet/in.h>

10435 DESCRIPTION

10436 The <netinet/in.h> header shall define the following types through **typedef**:

10437 **in_port_t** An unsigned integer type of exactly 16 bits.

10438 **in_addr_t** An unsigned integer type of exactly 32 bits.

10439 The **sa_family_t** type shall be defined as described in <sys/socket.h>.

10440 The **uint32_t** type shall be defined as described in <inttypes.h>. Inclusion of the <netinet/in.h> header may also make visible all symbols from <inttypes.h>.

10442 The <netinet/in.h> header shall define the **in_addr** structure that includes at least the following member:

10444 in_addr_t s_addr

10445 The <netinet/in.h> header shall define the **sockaddr_in** structure that includes at least the following members:

10447	sa_family_t	sin_family
10448	in_port_t	sin_port
10449	struct in_addr	sin_addr
10450	unsigned char	sin_zero[8]

10451 The **sockaddr_in** structure is used to store addresses for the Internet protocol family. Values of this type shall be cast by applications to **struct sockaddr** for use with socket functions.

10453 IP6 The <netinet/in.h> header shall define the **in6_addr** structure that contains at least the following member:

10455 uint8_t s6_addr[16]

10456 This array is used to contain a 128-bit IPv6 address, stored in network byte order.

10457 The <netinet/in.h> header shall define the **sockaddr_in6** structure that includes at least the following members:

10459	sa_family_t	sin6_family	AF_INET6.
10460	in_port_t	sin6_port	Port number.
10461	uint32_t	sin6_flowinfo	IPv6 traffic class and flow information.
10462	struct in6_addr	sin6_addr	IPv6 address.
10463	uint32_t	sin6_scope_id	Set of interfaces for a scope.

10464 The **sockaddr_in6** structure shall be set to zero by an application prior to using it, since implementations are free to have additional, implementation-defined fields in **sockaddr_in6**.

10466 The *sin6_scope_id* field is a 32-bit integer that identifies a set of interfaces as appropriate for the scope of the address carried in the *sin6_addr* field. For a link scope *sin6_addr*, *sin6_scope_id* would be an interface index. For a site scope *sin6_addr*, *sin6_scope_id* would be a site identifier. The mapping of *sin6_scope_id* to an interface or set of interfaces is implementation-defined.

10470 The <netinet/in.h> header shall declare the following external variable:

10471 struct in6_addr in6addr_any

10472 This variable is initialized by the system to contain the wildcard IPv6 address. The
 10473 <netinet/in.h> header also defines the IN6ADDR_ANY_INIT macro. This macro must be
 10474 constant at compile time and can be used to initialize a variable of type **struct in6_addr** to the
 10475 IPv6 wildcard address.

10476 The <netinet/in.h> header shall declare the following external variable:

```
10477 struct in6_addr in6addr_loopback
```

10478 This variable is initialized by the system to contain the loopback IPv6 address. The
 10479 <netinet/in.h> header also defines the IN6ADDR_LOOPBACK_INIT macro. This macro must be
 10480 constant at compile time and can be used to initialize a variable of type **struct in6_addr** to the
 10481 IPv6 loopback address.

10482 The <netinet/in.h> header shall define the **ipv6_mreq** structure that includes at least the
 10483 following members:

```
10484 struct in6_addr  ipv6mr_multiaddr  IPv6 multicast address.  

10485 unsigned        ipv6mr_interface  Interface index.
```

10486

10487 The <netinet/in.h> header shall define the following macros for use as values of the *level*
 10488 argument of *getsockopt()* and *setsockopt()*:

10489 IPPROTO_IP Internet protocol.

10490 ^{IP6} IPPROTO_IPV6 Internet Protocol Version 6.

10491 IPPROTO_ICMP Control message protocol.

10492 IPPROTO_TCP Transmission control protocol.

10493 IPPROTO_UDP User datagram protocol.

10494 The <netinet/in.h> header shall define the following macros for use as destination addresses for
 10495 *connect()*, *sendmsg()*, and *sendto()*:

10496 INADDR_ANY IPv4 local host address.

10497 INADDR_BROADCAST IPv4 broadcast address.

10498 The <netinet/in.h> header shall define the following macro to help applications declare buffers
 10499 of the proper size to store IPv4 addresses in string form:

10500 INET_ADDRSTRLEN 16.

10501 The *htonl()*, *htons()*, *ntohl()*, and *ntohs()* functions shall be available as defined in <arpa/inet.h>.
 10502 Inclusion of the <netinet/in.h> header may also make visible all symbols from <arpa/inet.h>.

10503 ^{IP6} The <netinet/in.h> header shall define the following macro to help applications declare buffers
 10504 of the proper size to store IPv6 addresses in string form:

10505 INET6_ADDRSTRLEN 46.

10506 The <netinet/in.h> header shall define the following macros, with distinct integral values, for
 10507 use in the *option_name* argument in the *getsockopt()* or *setsockopt()* functions at protocol level
 10508 IPPROTO_IPV6:

10509 IPV6_JOIN_GROUP Join a multicast group.

10510 IPV6_LEAVE_GROUP Quit a multicast group.

10511 IPV6_MULTICAST_HOPS
10512 Multicast hop limit.

10513 IPV6_MULTICAST_IF Interface to use for outgoing multicast packets.

10514 IPV6_MULTICAST_LOOP
10515 Multicast packets are delivered back to the local application.

10516 IPV6_UNICAST_HOPS Unicast hop limit.

10517 The <netinet/in.h> header shall define the following macros that test for special IPv6 addresses.
10518 Each macro is of type **int** and takes a single argument of type **const struct in6_addr***:

10519 IN6_IS_ADDR_UNSPECIFIED
10520 Unspecified address.

10521 IN6_IS_ADDR_LOOPBACK
10522 Loopback address.

10523 IN6_IS_ADDR_MULTICAST
10524 Multicast address.

10525 IN6_IS_ADDR_LINKLOCAL
10526 Unicast link-local address.

10527 IN6_IS_ADDR_SITELOCAL
10528 Unicast site-local address.

10529 IN6_IS_ADDR_V4MAPPED
10530 IPv4 mapped address.

10531 IN6_IS_ADDR_V4COMPAT
10532 IPv4-compatible address.

10533 IN6_IS_ADDR_MC_NODELOCAL
10534 Multicast node-local address.

10535 IN6_IS_ADDR_MC_LINKLOCAL
10536 Multicast link-local address.

10537 IN6_IS_ADDR_MC_SITELOCAL
10538 Multicast site-local address.

10539 IN6_IS_ADDR_MC_ORGLOCAL
10540 Multicast organization-local address.

10541 IN6_IS_ADDR_MC_GLOBAL
10542 Multicast global address.

10543 IN6_IS_ADDR_LINKLOCAL and IN6_IS_ADDR_SITELOCAL return true only for the two
10544 local-use IPv6 unicast addresses. They do not return true for multicast addresses of either link-
10545 local or site-local scope.

10546 **APPLICATION USAGE**

10547 None.

10548 **RATIONALE**

10549 None.

10550 **FUTURE DIRECTIONS**

10551 None.

10552 **SEE ALSO**

10553 <arpa/inet.h>, <inttypes.h>, <sys/socket.h>, the System Interfaces volume of |
10554 IEEE Std. 1003.1-200x, *connect()*, *getsockopt()*, *htonl()*, *htons()*, *ntohl()*, *ntohs()*, *sendmsg()*,
10555 *sendto()*, *setsockopt()*

10556 **CHANGE HISTORY**

10557 First released in Issue 6. Derived from the XNS, Issue 5.2 specification. |

10558 **NAME**

10559 netinet/tcp.h — definitions for the Internet Transmission Control Protocol (TCP)

10560 **SYNOPSIS**

10561 #include <netinet/tcp.h>

10562 **DESCRIPTION**

10563 The **<netinet/tcp.h>** header shall define the following macro for use as a socket option at the
10564 IPPROTO_TCP level:

10565 TCP_NODELAY Avoid coalescing of small segments.

10566 The macro shall be defined in the header. The implementation need not allow the value of the
10567 option to be set via *setsockopt()* or retrieved via *getsockopt()*.

10568 **APPLICATION USAGE**

10569 None.

10570 **RATIONALE**

10571 None.

10572 **FUTURE DIRECTIONS**

10573 None.

10574 **SEE ALSO**

10575 **<sys/socket.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, *getsockopt()*, *setsockopt()*

10576 **CHANGE HISTORY**

10577 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10578 **NAME**

10579 nl_types.h — data types

10580 **SYNOPSIS**

10581 XSI #include <nl_types.h>

10582

10583 **DESCRIPTION**

10584 The <nl_types.h> header shall contain definitions of at least the following types:

10585 **nl_catd** Used by the message catalog functions *catopen()*, *catgets()*, and *catclose()*
 10586 to identify a catalog descriptor.

10587 **nl_item** Used by *nl_langinfo()* to identify items of *langinfo* data. Values of objects
 10588 of type **nl_item** are defined in <langinfo.h>.

10589 The <nl_types.h> header shall contain definitions of at least the following constants:

10590 **NL_SETD** Used by *genocat* when no *\$set* directive is specified in a message text source
 10591 file; see the Internationalization Guide. This constant can be passed as the
 10592 value of *set_id* on subsequent calls to *catgets()* (that is, to retrieve
 10593 messages from the default message set). The value of **NL_SETD** is
 10594 implementation-defined.

10595 **NL_CAT_LOCALE** Value that must be passed as the *offlag* argument to *catopen()* to ensure
 10596 that message catalog selection depends on the *LC_MESSAGES* locale
 10597 category, rather than directly on the *LANG* environment variable.

10598 The following shall be declared as functions and may also be defined as macros. Function
 10599 prototypes shall be provided for use with an ISO C standard compiler.

```
10600 int      catclose(nl_catd);
10601 char    *catgets(nl_catd, int, int, const char *);
10602 nl_catd catopen(const char *, int);
```

10603 **APPLICATION USAGE**

10604 None.

10605 **RATIONALE**

10606 None.

10607 **FUTURE DIRECTIONS**

10608 None.

10609 **SEE ALSO**

10610 <langinfo.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *catclose()*, *catgets()*,
 10611 *catopen()*, *nl_langinfo()*, the Shell and Utilities volume of IEEE Std. 1003.1-200x, *genocat*

10612 **CHANGE HISTORY**

10613 First released in Issue 2.

10614 **Issue 4**

10615 The following change is incorporated for alignment with the ISO C standard:

- 10616 • The function declarations in this header are expanded to full ISO C standard prototypes.

10617 **NAME**

10618 poll.h — definitions for the poll() function

10619 **SYNOPSIS**

10620 XSI #include <poll.h>

10621

10622 **DESCRIPTION**10623 The <poll.h> header shall define the **pollfd** structure that includes at least the following
10624 members:

10625 int fd The following descriptor being polled.

10626 short events The input event flags (see below).

10627 short revents The output event flags (see below).

10628 The <poll.h> header shall define the following type through **typedef**:10629 **nfds_t** An unsigned integer type used for the number of file descriptors.10630 The following symbolic constants shall be defined, zero or more of which may be OR'ed together
10631 to form the *events* or *revents* members in the **pollfd** structure:

10632 POLLIN Same effect as POLLRDNORM | POLLRDBAND.

10633 POLLRDNORM Data on priority band 0 may be read.

10634 POLLRDBAND Data on priority bands greater than 0 may be read.

10635 POLLPRI High priority data may be read.

10636 POLLOUT Same value as POLLWRNORM.

10637 POLLWRNORM Data on priority band 0 may be written.

10638 POLLWRBAND Data on priority bands greater than 0 may be written. This event only
10639 examines bands that have been written to at least once.10640 POLLERR An error has occurred (*revents* only).10641 POLLHUP Device has been disconnected (*revents* only).10642 POLLNVAL Invalid *fd* member (*revents* only).10643 The <poll.h> header shall declare the following function which may also be defined as a macro.
10644 Function prototypes shall be provided for use with an ISO C standard compiler.

10645 int poll(struct pollfd[], nfds_t, int);

10646 **APPLICATION USAGE**

10647 None.

10648 **RATIONALE**

10649 None.

10650 **FUTURE DIRECTIONS**

10651 None.

10652 **SEE ALSO**10653 The System Interfaces volume of IEEE Std. 1003.1-200x, *poll()*

10654 **CHANGE HISTORY**

10655 First released in Issue 4, Version 2.

10656 NAME

10657 pthread.h — threads

10658 SYNOPSIS

10659 THR #include <pthread.h>

10660

10661 DESCRIPTION

10662 The <pthread.h> header shall define the following symbols:

- 10663 BAR PTHREAD_BARRIER_SERIAL_THREAD
- 10664 PTHREAD_CANCEL_ASYNCHRONOUS
- 10665 PTHREAD_CANCEL_ENABLE
- 10666 PTHREAD_CANCEL_DEFERRED
- 10667 PTHREAD_CANCEL_DISABLE
- 10668 PTHREAD_CANCELED
- 10669 PTHREAD_COND_INITIALIZER
- 10670 PTHREAD_CREATE_DETACHED
- 10671 PTHREAD_CREATE_JOINABLE
- 10672 PTHREAD_EXPLICIT_SCHED
- 10673 PTHREAD_INHERIT_SCHED
- 10674 XSI PTHREAD_MUTEX_DEFAULT
- 10675 PTHREAD_MUTEX_ERRORCHECK
- 10676 PTHREAD_MUTEX_INITIALIZER
- 10677 XSI PTHREAD_MUTEX_NORMAL
- 10678 PTHREAD_MUTEX_RECURSIVE
- 10679 PTHREAD_ONCE_INIT
- 10680 TPP|TPI PTHREAD_PRIO_INHERIT
- 10681 PTHREAD_PRIO_NONE
- 10682 PTHREAD_PRIO_PROTECT
- 10683 PTHREAD_PROCESS_SHARED
- 10684 PTHREAD_PROCESS_PRIVATE
- 10685 TPS PTHREAD_SCOPE_PROCESS
- 10686 PTHREAD_SCOPE_SYSTEM

10687

10688 The following types shall be defined as described in <sys/types.h>:

- 10689 pthread_attr_t
- 10690 BAR pthread_barrier_t
- 10691 pthread_barrierattr_t
- 10692 pthread_cond_t
- 10693 pthread_condattr_t
- 10694 pthread_key_t
- 10695 pthread_mutex_t
- 10696 pthread_mutexattr_t
- 10697 pthread_once_t
- 10698 pthread_rwlock_t
- 10699 pthread_rwlockattr_t
- 10700 SPI pthread_spinlock_t
- 10701 pthread_t

10702 The following shall be declared as functions and may also be declared as macros. Function
10703 prototypes shall be provided for use with an ISO C standard compiler.

```

10704     int   pthread_atfork(void (*)(void), void (*)(void),
10705                          void (*)(void));
10706     int   pthread_attr_destroy(pthread_attr_t *);
10707     int   pthread_attr_getdetachstate(const pthread_attr_t *, int *);
10708 XSI    int   pthread_attr_getguardsize(const pthread_attr_t *restrict,
10709                                       size_t *restrict);
10710 TPS    int   pthread_attr_getinheritsched(const pthread_attr_t *restrict,
10711                                           int *restrict);
10712     int   pthread_attr_getschedparam(const pthread_attr_t *restrict,
10713                                     struct sched_param *restrict);
10714 TPS    int   pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
10715                                         int *restrict);
10716 TPS    int   pthread_attr_getscope(const pthread_attr_t *restrict,
10717                                    int *restrict);
10718 TSA    int   pthread_attr_getstackaddr(const pthread_attr_t *restrict,
10719                                       void **restrict);
10720     int   pthread_attr_getstacksize(const pthread_attr_t *restrict,
10721                                    size_t *restrict);
10722     int   pthread_attr_init(pthread_attr_t *);
10723     int   pthread_attr_setdetachstate(pthread_attr_t *, int);
10724 XSI    int   pthread_attr_setguardsize(pthread_attr_t *, size_t);
10725 TPS    int   pthread_attr_setinheritsched(pthread_attr_t *, int);
10726     int   pthread_attr_setschedparam(pthread_attr_t *restrict,
10727                                     const struct sched_param *restrict);
10728 TPS    int   pthread_attr_setschedpolicy(pthread_attr_t *, int);
10729     int   pthread_attr_setscope(pthread_attr_t *, int);
10730 TSA    int   pthread_attr_setstackaddr(pthread_attr_t *, void *);
10731     int   pthread_attr_setstacksize(pthread_attr_t *, size_t);
10732 BAR    int   pthread_barrier_destroy(pthread_barrier_t *);
10733     int   pthread_barrier_init(pthread_barrier_t *restrict,
10734                               const pthread_barrierattr_t *restrict, unsigned);
10735     int   pthread_barrier_wait(pthread_barrier_t *);
10736     int   pthread_barrierattr_destroy(pthread_barrierattr_t *);
10737     int   pthread_barrierattr_getpshared(const pthread_barrierattr_t *restrict,
10738                                         int *restrict);
10739     int   pthread_barrierattr_init(pthread_barrierattr_t *);
10740     int   pthread_barrierattr_setpshared(pthread_barrierattr_t *, int);
10741     int   pthread_cancel(pthread_t);
10742     void  pthread_cleanup_push(void (*)(void *), void *);
10743     void  pthread_cleanup_pop(int);
10744     int   pthread_cond_broadcast(pthread_cond_t *);
10745     int   pthread_cond_destroy(pthread_cond_t *);
10746     int   pthread_cond_init(pthread_cond_t *restrict,
10747                             const pthread_condattr_t *restrict);
10748     int   pthread_cond_signal(pthread_cond_t *);
10749     int   pthread_cond_timedwait(pthread_cond_t *restrict,
10750                                 pthread_mutex_t *restrict,
10751                                 const struct timespec *restrict);
10752     int   pthread_cond_wait(pthread_cond_t *restrict,
10753                             pthread_mutex_t *restrict);
10754 CS     int   pthread_condattr_getclock(const pthread_condattr_t *restrict,
10755                                       clockid_t *restrict);

```

```
10756     int    pthread_condattr_getpshared(const pthread_condattr_t *restrict,
10757                                     int *restrict);
10758     int    pthread_condattr_init(pthread_condattr_t *);
10759 CS     int    pthread_condattr_setclock(pthread_condattr_t *, clockid_t);
10760     int    pthread_condattr_setpshared(pthread_condattr_t *, int);
10761     int    pthread_create(pthread_t *restrict, const pthread_attr_t *restrict,
10762                           void *(*)(void *), void *);
10763     int    pthread_detach(pthread_t);
10764     int    pthread_equal(pthread_t, pthread_t);
10765     void   pthread_exit(void *);
10766 XSI   int    pthread_getconcurrency(void);
10767 TCT   int    pthread_getcpuclockid(pthread_t, clockid_t *);
10768 TPS   int    pthread_getschedparam(pthread_t, int *restrict,
10769                                   struct sched_param *restrict);
10770     void *pthread_getspecific(pthread_key_t);
10771     int    pthread_join(pthread_t, void **);
10772     int    pthread_key_create(pthread_key_t *, void (*)(void *));
10773     int    pthread_key_delete(pthread_key_t);
10774     int    pthread_kill(pthread_t, int);
10775     int    pthread_mutex_destroy(pthread_mutex_t *);
10776 TPP   int    pthread_mutex_getprioceiling(const pthread_mutex_t *restrict,
10777                                         int *restrict);
10778     int    pthread_mutex_init(pthread_mutex_t *restrict,
10779                               const pthread_mutexattr_t *restrict);
10780     int    pthread_mutex_lock(pthread_mutex_t *);
10781 TPP   int    pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int,
10782                                         int *restrict);
10783 TMO   int    pthread_mutex_timedlock(pthread_mutex_t *,
10784                                       const struct timespec *);
10785     int    pthread_mutex_trylock(pthread_mutex_t *);
10786     int    pthread_mutex_unlock(pthread_mutex_t *);
10787     int    pthread_mutexattr_destroy(pthread_mutexattr_t *);
10788 TPP|TPI int    pthread_mutexattr_getprioceiling(const pthread_mutexattr_t *restrict,
10789                                               int *restrict);
10790     int    pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict,
10791                                         int *restrict);
10792     int    pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict,
10793                                         int *restrict);
10794 XSI   int    pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
10795                                       int *restrict);
10796     int    pthread_mutexattr_init(pthread_mutexattr_t *);
10797 TPP|TPI int    pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
10798     int    pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
10799     int    pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
10800 XSI   int    pthread_mutexattr_settype(pthread_mutexattr_t *, int);
10801     int    pthread_once(pthread_once_t *, void (*)(void));
10802     int    pthread_rwlock_destroy(pthread_rwlock_t *);
10803     int    pthread_rwlock_init(pthread_rwlock_t *restrict,
10804                               const pthread_rwlockattr_t *restrict);
10805     int    pthread_rwlock_rdlock(pthread_rwlock_t *);
10806     int    pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict,
10807                                       const struct timespec *restrict);
```



```

10808     int    pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict,
10809         const struct timespec *restrict);
10810     int    pthread_rwlock_tryrdlock(pthread_rwlock_t *);
10811     int    pthread_rwlock_trywrlock(pthread_rwlock_t *);
10812     int    pthread_rwlock_unlock(pthread_rwlock_t *);
10813     int    pthread_rwlock_wrlock(pthread_rwlock_t *);
10814     int    pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
10815     int    pthread_rwlockattr_getpshared(const pthread_rwlockattr_t *restrict,
10816         int *restrict);
10817     int    pthread_rwlockattr_init(pthread_rwlockattr_t *);
10818     int    pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
10819     pthread_t
10820         pthread_self(void);
10821     int    pthread_setcancelstate(int, int *);
10822     int    pthread_setcanceltype(int, int *);
10823 XSI    int    pthread_setconcurrency(int);
10824 TPS    int    pthread_setschedparam(pthread_t, int,
10825         const struct sched_param *);
10826     int    pthread_setspecific(pthread_key_t, const void *);
10827     int    pthread_sigmask(int, const sigset_t *restrict, sigset_t *restrict);
10828 SPI    int    pthread_spin_destroy(pthread_spinlock_t *);
10829     int    pthread_spin_init(pthread_spinlock_t *, int);
10830     int    pthread_spin_lock(pthread_spinlock_t *);
10831     int    pthread_spin_trylock(pthread_spinlock_t *);
10832     int    pthread_spin_unlock(pthread_spinlock_t *);
10833     void   pthread_testcancel(void);

```

10834 XSI **Inclusion of the <pthread.h> header shall make symbols defined in the headers <sched.h> and**
10835 **<time.h> visible.**

10836 **APPLICATION USAGE**

10837 An interpretation request has been filed with IEEE PASC concerning requirements for visibility
10838 of symbols in this header.

10839 **RATIONALE**

10840 None.

10841 **FUTURE DIRECTIONS**

10842 None.

10843 **SEE ALSO**

10844 <sched.h>, <time.h>, the System Interfaces volume of IEEE Std. 1003.1-200x,
10845 *pthread_attr_getguardsize()*, *pthread_attr_init()*, *pthread_attr_setscope()*, *pthread_barrier_destroy()*,
10846 *pthread_barrier_init()*, *pthread_barrier_wait()*, *pthread_barrierattr_destroy()*,
10847 *pthread_barrierattr_getpshared()*, *pthread_barrierattr_init()*, *pthread_barrierattr_setpshared()*,
10848 *pthread_cancel()*, *pthread_cleanup_pop()*, *pthread_cond_init()*, *pthread_cond_signal()*,
10849 *pthread_cond_wait()*, *pthread_condattr_getclock()*, *pthread_condattr_init()*,
10850 *pthread_condattr_setclock()*, *pthread_create()*, *pthread_detach()*, *pthread_equal()*, *pthread_exit()*,
10851 *pthread_getconcurrency()*, *pthread_getcpuclockid()*, *pthread_getschedparam()*, *pthread_join()*,
10852 *pthread_key_create()*, *pthread_key_delete()*, *pthread_mutex_init()*, *pthread_mutex_lock()*,
10853 *pthread_mutex_setprioceiling()*, *pthread_mutex_timedlock()*, *pthread_mutexattr_init()*,
10854 *pthread_mutexattr_gettype()*, *pthread_mutexattr_setprotocol()*, *pthread_once()*,
10855 *pthread_rwlock_destroy()*, *pthread_rwlock_init()*, *pthread_rwlock_rdlock()*,
10856 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlock_tryrdlock()*,
10857 *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*, *pthread_rwlock_wrlock()*,

10858 *pthread_rwlockattr_destroy()*, *pthread_rwlockattr_getpshared()*, *pthread_rwlockattr_init()*,
10859 *pthread_rwlockattr_setpshared()*, *pthread_self()*, *pthread_setcancelstate()*, *pthread_setspecific()*,
10860 *pthread_spin_destroy()*, *pthread_spin_init()*, *pthread_spin_lock()*, *pthread_spin_trylock()*,
10861 *pthread_spin_unlock()*

10862 **CHANGE HISTORY**

10863 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

10864 **Issue 6**

10865 The RTT margin markers are now broken out into their POSIX options.

10866 The Open Group corrigenda item U021/9 has been applied, correcting the prototype for the
10867 *pthread_cond_wait()* function.

10868 The Open Group corrigenda item U026/2 has been applied correcting the prototype for the
10869 *pthread_setschedparam()* function so that its second argument is of type **int**.

10870 The *pthread_getcpuclockid()* and *pthread_mutex_timedlock()* functions are added for alignment
10871 with IEEE Std. 1003.1d-1999.

10872 The following functions are added for alignment with IEEE Std. 1003.1j-2000:

10873 *pthread_barrier_destroy()*, *pthread_barrier_init()*, *pthread_barrier_wait()*,
10874 *pthread_barrierattr_destroy()*, *pthread_barrierattr_getpshared()*, *pthread_barrierattr_init()*,
10875 *pthread_barrierattr_setpshared()*, *pthread_condattr_getclock()*, *pthread_condattr_setclock()*,
10876 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_spin_destroy()*,
10877 *pthread_spin_init()*, *pthread_spin_lock()*, *pthread_spin_trylock()*, and *pthread_spin_unlock()*.

10878 PTHREAD_RWLOCK_INITIALIZER is deleted for alignment with IEEE Std. 1003.1j-2000.

10879 Functions previously marked as part of the Read-Write Locks option are now moved to the
10880 Threads option.

10881 The **restrict** keyword is added to the prototypes for *pthread_attr_getguardsize()*,
10882 *pthread_attr_getinheritsched()*, *pthread_attr_getschedparam()*, *pthread_attr_getschedpolicy()*,
10883 *pthread_attr_getscope()*, *pthread_attr_getstackaddr()*, *pthread_attr_getstacksize()*,
10884 *pthread_attr_setschedparam()*, *pthread_barrier_init()*, *pthread_barrierattr_getpshared()*,
10885 *pthread_cond_init()*, *pthread_cond_signal()*, *pthread_cond_timedwait()*, *pthread_cond_wait()*,
10886 *pthread_condattr_getclock()*, *pthread_condattr_getpshared()*, *pthread_create()*,
10887 *pthread_getschedparam()*, *pthread_mutex_getprioceiling()*, *pthread_mutex_init()*,
10888 *pthread_mutex_setprioceiling()*, *pthread_mutexattr_getprioceiling()*, *pthread_mutexattr_getprotocol()*,
10889 *pthread_mutexattr_getpshared()*, *pthread_mutexattr_gettype()*, *pthread_rwlock_init()*,
10890 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlockattr_getpshared()*, and
10891 *pthread_sigmask()*.

10892 **NAME**

10893 pwd.h — password structure

10894 **SYNOPSIS**

10895 #include <pwd.h>

10896 **DESCRIPTION**10897 The <pwd.h> header shall provide a definition for **struct passwd**, which shall include at least the
10898 following members:

10899	char	*pw_name	User's login name.
10900	uid_t	pw_uid	Numerical user ID.
10901	gid_t	pw_gid	Numerical group ID.
10902	char	*pw_dir	Initial working directory.
10903	char	*pw_shell	Program to use as shell.

10904 The **gid_t** and **uid_t** types shall be defined as described in <sys/types.h>.10905 The following shall be declared as functions and may also be defined as macros. Function
10906 prototypes shall be provided for use with an ISO C standard compiler.

```

10907 struct passwd *getpwnam(const char *);
10908 struct passwd *getpwuid(uid_t);
10909 TSF int getpwnam_r(const char *, struct passwd *, char *,
10910                size_t, struct passwd **);
10911 int getpwuid_r(uid_t, struct passwd *, char *,
10912               size_t, struct passwd **);
10913 XSI void endpwent(void);
10914 struct passwd *getpwent(void);
10915 void setpwent(void);
10916

```

10917 **APPLICATION USAGE**

10918 None.

10919 **RATIONALE**

10920 None.

10921 **FUTURE DIRECTIONS**

10922 None.

10923 **SEE ALSO**10924 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *endpwent()*, *getpwnam()*,
10925 *getpwuid()*10926 **CHANGE HISTORY**

10927 First released in Issue 1.

10928 **Issue 4**10929 Reference to the <sys/types.h> header is added for the definitions of **gid_t** and **uid_t**. This is
10930 marked as an extension.

10931 The following change is incorporated for alignment with the ISO POSIX-1 standard:

10932 • The function declarations in this header are expanded to full ISO C standard prototypes.

10933 **Issue 4, Version 2**

10934 For X/OPEN UNIX conformance, the *getpwent()*, *endpwent()*, and *setpwent()* functions are added
10935 to the list of functions declared in this header.

10936 **Issue 5**

10937 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

10938 **Issue 6**

10939 The following new requirements on POSIX implementations derive from alignment with the
10940 Single UNIX Specification:

- 10941 • The **gid_t** and **uid_t** types are mandated.
- 10942 • The *getpwnam_r()* and *getpwuid_r()* functions are marked as part of the
10943 `_POSIX_THREAD_SAFE_FUNCTIONS` option.

10944 **NAME**

10945 regex.h — regular expression matching types

10946 **SYNOPSIS**

10947 #include <regex.h>

10948 **DESCRIPTION**

10949 The <regex.h> header shall define the structures and symbolic constants used by the *regcomp()*,
10950 *regexexec()*, *regerror()*, and *regfree()* functions.

10951 The structure type **regex_t** shall contain at least the following member:

10952 size_t re_nsub Number of parenthesized subexpressions.

10953 The type **regoff_t** shall be defined as a signed arithmetic type that can hold the largest value that
10954 can be stored in either a type **off_t** or type **ssize_t**. The structure type **regmatch_t** shall contain
10955 at least the following members:

10956 regoff_t rm_so Byte offset from start of string
10957 to start of substring.

10958 regoff_t rm_eo Byte offset from start of string of the
10959 first character after the end of substring.

10960 Values for the *cflags* parameter to the *regcomp()* function:

10961 REG_EXTENDED Use Extended Regular Expressions.

10962 REG_ICASE Ignore case in match.

10963 REG_NOSUB Report only success or fail in *regexexec()*.

10964 REG_NEWLINE Change the handling of newline.

10965 Values for the *eflags* parameter to the *regexexec()* function:

10966 REG_NOTBOL The circumflex character ('^'), when taken as a special character, does
10967 not match the beginning of *string*.

10968 REG_NOTEOL The dollar sign ('\$'), when taken as a special character, does not match
10969 the end of *string*.

10970 The following constants shall be defined as error return values:

10971 REG_NOMATCH *regexexec()* failed to match.

10972 REG_BADPAT Invalid regular expression.

10973 REG_ECOLLATE Invalid collating element referenced.

10974 REG_ECTYPE Invalid character class type referenced.

10975 REG_EESCAPE Trailing '\\ ' in pattern.

10976 REG_ESUBREG Number in *\digit* invalid or in error.

10977 REG_EBRACK "[]" imbalance.

10978 REG_EPAREN "\\(\\)" or "\\()" imbalance.

10979 REG_EBRACE "\\{\\}" imbalance.

10980 REG_BADBR Content of "\\{\\}" invalid: not a number, number too large, more than
10981 two numbers, first larger than second.

10982 **REG_ERANGE** Invalid endpoint in range expression.

10983 **REG_ESPACE** Out of memory.

10984 **REG_BADRPT** '?', '*', or '+' not preceded by valid regular expression.

10985 **REG_ENOSYS** The implementation does not support the function. **(LEGACY)**

10986 The following shall be declared as functions and may also be declared as macros. Function
10987 prototypes shall be provided for use with an ISO C standard compiler.

```
10988        int     regcomp(regex_t *restrict, const char *restrict, int);
10989        size_t regerror(int, const regex_t *restrict, char *restrict, size_t);
10990        int     regexec(const regex_t *restrict, const char *restrict, size_t,
10991                        regmatch_t[restrict], int);
10992        void    regfree(regex_t *);
```

10993 The implementation may define additional macros or constants using names beginning with
10994 **REG_**.

10995 **APPLICATION USAGE**

10996 None.

10997 **RATIONALE**

10998 None.

10999 **FUTURE DIRECTIONS**

11000 None.

11001 **SEE ALSO**

11002 The System Interfaces volume of IEEE Std. 1003.1-200x, *regcomp()*, the Shell and Utilities volume
11003 of IEEE Std. 1003.1-200x

11004 **CHANGE HISTORY**

11005 First released in Issue 4.

11006 Originally derived from the ISO POSIX-2 standard.

11007 **Issue 6**

11008 The **REG_ENOSYS** constant is marked **LEGACY**.

11009 The **restrict** keyword is added to the prototypes for *regcomp()*, *regerror()*, and *regexec()*.

11010 **NAME**

11011 sched.h — execution scheduling (**REALTIME**)

11012 **SYNOPSIS**

11013 PS #include <sched.h>

11014

11015 **DESCRIPTION**

11016 The <sched.h> header shall define the **sched_param** structure, which contains the scheduling
 11017 parameters required for implementation of each supported scheduling policy. This structure
 11018 shall contain at least the following member:

11019 int sched_priority Process execution scheduling priority.

11020 SS|TSP In addition, if **_POSIX_SPORADIC_SERVER** or **_POSIX_THREAD_SPORADIC_SERVER** is
 11021 defined, the **sched_param** structure defined in <sched.h> shall contain the following members
 11022 in addition to those specified above:

11023	int	sched_ss_low_priority	Low scheduling priority for
11024			sporadic server.
11025	struct timespec	sched_ss_repl_period	Replenishment period for
11026			sporadic server.
11027	struct timespec	sched_ss_init_budget	Initial budget for sporadic server.
11028	int	sched_ss_max_repl	Maximum pending replenishments for
11029			sporadic server.

11030

11031 Each process is controlled by an associated scheduling policy and priority. Associated with each
 11032 policy is a priority range. Each policy definition specifies the minimum priority range for that
 11033 policy. The priority ranges for each policy may overlap the priority ranges of other policies.

11034 Four scheduling policies are defined; others may be defined by the implementation. The four
 11035 standard policies are indicated by the values of the following symbolic constants:

11036 **SCHED_FIFO** First in-first out (FIFO) scheduling policy.

11037 **SCHED_RR** Round robin scheduling policy.

11038 SS|TSP **SCHED_SPORADIC** Sporadic server scheduling policy.

11039 **SCHED_OTHER** Another scheduling policy.

11040 The values of these constants are distinct.

11041 The following shall be declared as functions and may also be declared as macros. Function
 11042 prototypes shall be provided for use with an ISO C standard compiler.

```

11043 int sched_get_priority_max(int);
11044 int sched_get_priority_min(int);
11045 int sched_getparam(pid_t, struct sched_param *);
11046 int sched_getscheduler(pid_t);
11047 int sched_rr_get_interval(pid_t, struct timespec *);
11048 int sched_setparam(pid_t, const struct sched_param *);
11049 int sched_setscheduler(pid_t, int, const struct sched_param *);
11050 int sched_yield(void);
    
```

11051 Inclusion of the <sched.h> header makes symbols defined in the header <time.h> visible.

11052 **APPLICATION USAGE**

11053 None.

11054 **RATIONALE**

11055 None.

11056 **FUTURE DIRECTIONS**

11057 None.

11058 **SEE ALSO**

11059 <time.h>

11060 **CHANGE HISTORY**

11061 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

11062 **Issue 6**

11063 The <sched.h> header is marked as part of the Process Scheduling option.

11064 Sporadic server members are added to the **sched_param** structure, and the SCHED_SPORADIC
11065 scheduling policy is added for alignment with IEEE Std. 1003.1d-1999.

11066 IEEE PASC Interpretation 1003.1 #108 is applied, correcting the **sched_param** structure whose
11067 members *sched_ss_repl_period* and *sched_ss_init_budget* members should be type **struct timespec**
11068 and not **timespec**.

11069 **NAME**

11070 search.h — search tables

11071 **SYNOPSIS**

11072 XSI #include <search.h>

11073

11074 **DESCRIPTION**

11075 The <search.h> header shall provide a type definition, **ENTRY**, for structure **entry** which shall
 11076 include the following members:

11077 char *key
 11078 void *data

11079 and shall define **ACTION** and **VISIT** as enumeration data types through type definitions as
 11080 follows:

11081 enum { FIND, ENTER } ACTION;
 11082 enum { preorder, postorder, endorder, leaf } VISIT;

11083 The **size_t** type shall be defined as described in <sys/types.h>.

11084 Each of the following shall be declared as a function, or defined as a macro, or both. Function
 11085 prototypes shall be provided for use with an ISO C standard compiler.

11086 int hcreate(size_t);
 11087 void hdestroy(void);
 11088 ENTRY *hsearch(ENTRY, ACTION);
 11089 void insque(void *, void *);
 11090 void *lfind(const void *, const void *, size_t *,
 11091 size_t, int (*)(const void *, const void *));
 11092 void *lsearch(const void *, void *, size_t *,
 11093 size_t, int (*)(const void *, const void *));
 11094 void remque(void *);
 11095 void *tdelete(const void *restrict, void **restrict,
 11096 int (*)(const void *, const void *));
 11097 void *tfind(const void *, void *const *,
 11098 int (*)(const void *, const void *));
 11099 void *tsearch(const void *, void **,
 11100 int (*)(const void *, const void *));
 11101 void twalk(const void *,
 11102 void (*)(const void *, VISIT, int));

11103 **APPLICATION USAGE**

11104 None.

11105 **RATIONALE**

11106 None.

11107 **FUTURE DIRECTIONS**

11108 None.

11109 **SEE ALSO**

11110 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *hcreate()*, *insque()*,
 11111 *lsearch()*, *remque()*, *tsearch()*

11112 **CHANGE HISTORY**

11113 First released in Issue 1. Derived from Issue 1 of the SVID.

11114 **Issue 4**

11115 The function declarations in this header are expanded to full ISO C standard prototypes.

11116 Reference to the <sys/types.h> header is added for the definition of **size_t**.

11117 **Issue 4, Version 2**

11118 For X/OPEN UNIX conformance, the *insque()* and *remque()* functions are added to the list of
11119 functions declared in this header.

11120 **Issue 6**

11121 The Open Group corrigenda item U021/6 has been applied updating the prototypes for *tdelete()*
11122 and *tsearch()*.

11123 The **restrict** keyword is added to the prototype for *tdelete()*.

11124 **NAME**

11125 semaphore.h — semaphores (**REALTIME**)

11126 **SYNOPSIS**

11127 SEM `#include <semaphore.h>`

11128

11129 **DESCRIPTION**

11130 The <semaphore.h> header shall define the **sem_t** type, used in performing semaphore
 11131 operations. The semaphore may be implemented using a file descriptor, in which case
 11132 applications are able to open up at least a total of {OPEN_MAX} files and semaphores. The
 11133 symbol SEM_FAILED shall be defined (see *sem_open()*).

11134 The following shall be declared as functions and may also be declared as macros. Function
 11135 prototypes shall be provided for use with an ISO C standard compiler.

```

11136 int    sem_close(sem_t *);
11137 int    sem_destroy(sem_t *);
11138 int    sem_getvalue(sem_t *restrict, int *restrict);
11139 int    sem_init(sem_t *, int, unsigned);
11140 sem_t *sem_open(const char *, int, ...);
11141 int    sem_post(sem_t *);
11142 TMO   int    sem_timedwait(sem_t *restrict, const struct timespec *restrict);
11143 int    sem_trywait(sem_t *);
11144 int    sem_unlink(const char *);
11145 int    sem_wait(sem_t *);
    
```

11146 Inclusion of the <semaphore.h> header may make visible symbols defined in the headers
 11147 <fcntl.h> and <sys/types.h>.

11148 **APPLICATION USAGE**

11149 None.

11150 **RATIONALE**

11151 None.

11152 **FUTURE DIRECTIONS**

11153 None.

11154 **SEE ALSO**

11155 <fcntl.h>, <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *sem_destroy()*,
 11156 *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_timedwait()*, *sem_trywait()*, *sem_unlink()*,
 11157 *sem_wait()*

11158 **CHANGE HISTORY**

11159 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

11160 **Issue 6**

11161 The <semaphore.h> header is marked as part of the Semaphores option.

11162 The Open Group corrigenda item U021/3 has been applied, adding a description of
 11163 SEM_FAILED.

11164 The *sem_timedwait()* function is added for alignment with IEEE Std. 1003.1d-1999.

11165 The **restrict** keyword is added to the prototypes for *sem_getvalue()* and *sem_timedwait()*.

11166 **NAME**

11167 setjmp.h — stack environment declarations

11168 **SYNOPSIS**

11169 #include <setjmp.h>

11170 **DESCRIPTION**

11171 cx The functionality described on this reference page extends the ISO C standard. Applications
11172 shall define the appropriate feature test macro (see the System Interfaces volume of
11173 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
11174 symbols in this header.

11175 The <setjmp.h> header shall contain the type definitions for array types **jmp_buf** and
11176 **sigjmp_buf**.

11177 The following shall be declared as functions and may also be defined as macros. Function
11178 prototypes shall be provided for use with an ISO C standard compiler.

```
11179       void    longjmp(jmp_buf, int);  
11180       void    siglongjmp(sigjmp_buf, int);  
11181 xSI       void  _longjmp(jmp_buf, int);
```

11182

11183 Each of the following may be declared as a function, or defined as a macro, or both. Function
11184 prototypes shall be provided for use with an ISO C standard compiler.

```
11185       int     setjmp(jmp_buf);  
11186       int     sigsetjmp(sigjmp_buf, int);  
11187 xSI       int  _setjmp(jmp_buf);
```

11188

11189 **APPLICATION USAGE**

11190 None.

11191 **RATIONALE**

11192 None.

11193 **FUTURE DIRECTIONS**

11194 None.

11195 **SEE ALSO**

11196 The System Interfaces volume of IEEE Std. 1003.1-200x, *longjmp()*, *_longjmp()*, *setjmp()*,
11197 *siglongjmp()*, *sigsetjmp()*

11198 **CHANGE HISTORY**

11199 First released in Issue 1.

11200 **Issue 4**

11201 The following changes are incorporated for alignment with the ISO C standard:

- 11202 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 11203 • The DESCRIPTION is changed to indicate that all functions in this header can also be
11204 declared as macros.
- 11205 • The arguments **jmp_buf** and **sigjmp_buf** are specified as array types.

11206 **Issue 4, Version 2**

11207 For X/OPEN UNIX conformance, the *_longjmp()* and *_setjmp()* functions are added to the list of
11208 functions declared in this header.

11209 NAME

11210 signal.h — signals

11211 SYNOPSIS

11212 #include <signal.h>

11213 DESCRIPTION

11214 cx The functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of symbols in this header.

11218 The <signal.h> header shall define the following symbolic constants, each of which expands to a distinct constant expression of the type:

11220 void (*)(int)

11221 whose value matches no declarable function.

11222 SIG_DFL Request for default signal handling.

11223 SIG_ERR Return value from *signal()* in case of error.

11224 SIG_HOLD Request that signal be held.

11225 SIG_IGN Request that signal be ignored.

11226 The following data types shall be defined through **typedef**:

11227 **sig_atomic_t** Possibly volatile-qualified integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.

11229 **sigset_t** Integer or structure type of an object used to represent sets of signals.

11230 **pid_t** As described in <sys/types.h>.

11231 RTS The <signal.h> header shall define the **sigevent** structure, which has at least the following members:

11233	int	sigev_notify	Notification type.
11234	int	sigev_signo	Signal number.
11235	union sigval	sigev_value	Signal value.
11236	void (*)(union sigval)	sigev_notify_function	Notification function.
11237	(pthread_attr_t *)	sigev_notify_attributes	Notification attributes.

11238 The following values of *sigev_notify* shall be defined:

11239 SIGEV_NONE No asynchronous notification is delivered when the event of interest occurs.

11241 SIGEV_SIGNAL A queued signal, with an application-defined value, is generated when the event of interest occurs.

11243 SIGEV_THREAD A notification function is called to perform notification.

11244 The **sigval** union shall be defined as:

11245	int	sival_int	Integer signal value.
11246	void *	sival_ptr	Pointer signal value.

11247 This header shall also declare the macros SIGRTMIN and SIGRTMAX, which evaluate to integral expressions and, if the Realtime Signals Extension option is supported, specify a range of signal numbers that are reserved for application use and for which the realtime signal

11250 behavior specified in this volume of IEEE Std. 1003.1-200x is supported. The signal numbers in
 11251 this range do not overlap any of the signals specified in the following table.

11252 The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG_MAX} signal
 11253 numbers.

11254 It is implementation-defined whether realtime signal behavior is supported for other signals.

11255 This header also declares the constants that are used to refer to the signals that occur in the
 11256 system. Signals defined here begin with the letters SIG. Each of the signals have distinct positive
 11257 integral values. The value 0 is reserved for use as the null signal (see *kill()*). Additional
 11258 implementation-defined signals may occur in the system.

11259 The following signals shall be supported on all implementations (default actions are explained
 11260 below the table):

11261

11262

	Signal	Default Action	Description
11263	SIGABRT	A	Process abort signal.
11264	SIGALRM	T	Alarm clock.
11265	SIGBUS	A	Access to an undefined portion of a memory object.
11266	SIGCHLD	I	Child process terminated or stopped.
11267	SIGCONT	C	Continue executing, if stopped.
11268	SIGFPE	A	Erroneous arithmetic operation.
11269	SIGHUP	T	Hangup.
11270	SIGILL	A	Illegal instruction.
11271	SIGINT	T	Terminal interrupt signal.
11272	SIGKILL	T	Kill (cannot be caught or ignored).
11273	SIGPIPE	T	Write on a pipe with no one to read it.
11274	SIGQUIT	A	Terminal quit signal.
11275	SIGSEGV	A	Invalid memory reference.
11276	SIGSTOP	S	Stop executing (cannot be caught or ignored).
11277	SIGTERM	T	Termination signal.
11278	SIGTSTP	S	Terminal stop signal.
11279	SIGTTIN	S	Background process attempting read.
11280	SIGTTOU	S	Background process attempting write.
11281	SIGUSR1	T	User-defined signal 1.
11282	SIGUSR2	T	User-defined signal 2.
11283 XSI	SIGPOLL	T	Pollable event.
11284	SIGPROF	T	Profiling timer expired.
11285	SIGSYS	A	Bad system call.
11286	SIGTRAP	A	Trace/breakpoint trap.
11287	SIGURG	I	High bandwidth data is available at a socket.
11288 XSI	SIGVTALRM	T	Virtual timer expired.
11289	SIGXCPU	A	CPU time limit exceeded.
11290	SIGXFSZ	A	File size limit exceeded.

11291 The default actions are as follows:

11292 T Abnormal termination of the process. The process is terminated with all the consequences
 11293 of *_exit()* except that the status made available to *wait()* and *waitpid()* indicates abnormal
 11294 termination by the specified signal.

11295 A Abnormal termination of the process.

11296 XSI Additionally, implementation-defined abnormal termination actions, such as creation of a
 11297 core file, may occur.

11298 I Ignore the signal.
11299 S Stop the process.
11300 C Continue the process, if it is stopped; otherwise, ignore the signal.

11301 The header shall provide a declaration of **struct sigaction**, including at least the following
11302 members:

11303 void (*sa_handler)(int) What to do on receipt of signal.
11304 sigset_t sa_mask Set of signals to be blocked during execution
11305 of the signal handling function.
11306 int sa_flags Special flags.
11307 void (*)(int, siginfo_t *, void *) sa_sigaction
11308 Pointer to signal handler function or one
11309 of the macros SIG_IGN or SIG_DFL.

11310 XSI The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a portable program must
11311 not use both simultaneously.

11312 The following shall be declared as constants:

11313 SA_NOCLDSTOP Do not generate SIGCHLD when children stop.
11314 SIG_BLOCK The resulting set is the union of the current set and the signal set pointed
11315 to by the argument *set*.
11316 SIG_UNBLOCK The resulting set is the intersection of the current set and the complement
11317 of the signal set pointed to by the argument *set*.
11318 SIG_SETMASK The resulting set is the signal set pointed to by the argument *set*.

11319 XSI SA_ONSTACK Causes signal delivery to occur on an alternate stack.
11320 XSI SA_RESETHAND Causes signal dispositions to be set to SIG_DFL on entry to signal
11321 handlers.
11322 XSI SA_RESTART Causes certain functions to become restartable.
11323 XSI SA_SIGINFO Causes extra information to be passed to signal handlers at the time of
11324 receipt of a signal.
11325 XSI SA_NOCLDWAIT Causes implementations not to create zombie processes on child death.
11326 XSI SA_NODEFER Causes signal not to be automatically blocked on entry to signal handler.
11327 XSI SS_ONSTACK Process is executing on an alternate signal stack.
11328 XSI SS_DISABLE Alternate signal stack is disabled.
11329 XSI MINSIGSTKSZ Minimum stack size for a signal handler.
11330 XSI SIGSTKSZ Default size in bytes for the alternate signal stack.

11331 XSI The **ucontext_t** structure shall be defined through **typedef** as described in <ucontext.h>.
11332 The **mcontext_t** type shall be defined through **typedef** as described in <ucontext.h>.
11333 The <signal.h> header shall define the **stack_t** type as a structure that includes at least the
11334 following members:

11335 void *ss_sp Stack base or pointer.
11336 size_t ss_size Stack size.
11337 int ss_flags Flags.

11338 The <signal.h> header shall define the **sigstack** structure that includes at least the following
 11339 members:

11340	int	ss_onstack	Non-zero when signal stack is in use.
11341	void	*ss_sp	Signal stack pointer.

11342

11343 The <signal.h> header shall define the **siginfo_t** type as a structure that includes at least the
 11344 following members:

11345	int	si_signo	Signal number.
11346 XSI	int	si_errno	If non-zero, an <i>errno</i> value associated with 11347 this signal, as defined in <errno.h>.
11348	int	si_code	Signal code.
11349 XSI	pid_t	si_pid	Sending process ID.
11350	uid_t	si_uid	Real user ID of sending process.
11351	void	*si_addr	Address of faulting instruction.
11352	int	si_status	Exit value or signal.
11353	long	si_band	Band event for SIGPOLL.
11354 RTS	union sigval	si_value	Signal value.

11355

11356 The macros specified in the **Code** column of the following table are defined for use as values of
 11357 XSI *si_code* that are signal-specific or non-signal-specific reasons why the signal was generated.

11358

11359

11360 XSI

11361

11362

11363

11364

11365

11366

11367

11368

11369

11370

11371

11372

11373

11374

11375

11376

11377

11378

11379

11380

11381

11382

11383

11384

11385

11386

11387

11388

11389

11390

11391

11392

11393

11394

11395

11396

11397

11398

11399

11400

11401

Signal	Code	Reason
SIGILL	ILL_ILLOPC	Illegal opcode.
	ILL_ILLOPN	Illegal operand.
	ILL_ILLADR	Illegal addressing mode.
	ILL_ILLTRP	Illegal trap.
	ILL_PRVOPC	Privileged opcode.
	ILL_PRVREG	Privileged register.
	ILL_COPROC	Coprocessor error.
	ILL_BADSTK	Internal stack error.
SIGFPE	FPE_INTDIV	Integer divide by zero.
	FPE_INTOVF	Integer overflow.
	FPE_FLTDIV	Floating point divide by zero.
	FPE_FLTOVF	Floating point overflow.
	FPE_FLTUND	Floating point underflow.
	FPE_FLTRES	Floating point inexact result.
	FPE_FLTINV	Invalid floating point operation.
	FPE_FLTSUB	Subscript out of range.
SIGSEGV	SEGV_MAPERR	Address not mapped to object.
	SEGV_ACCERR	Invalid permissions for mapped object.
SIGBUS	BUS_ADRALN	Invalid address alignment.
	BUS_ADRERR	Non-existent physical address.
	BUS_OBJERR	Object specific hardware error.
SIGTRAP	TRAP_BRKPT	Process breakpoint.
	TRAP_TRACE	Process trace trap.
SIGCHLD	CLD_EXITED	Child has exited.
	CLD_KILLED	Child has terminated abnormally and did not create a core file.
	CLD_DUMPED	Child has terminated abnormally and created a core file.
	CLD_TRAPPED	Traced child has trapped.
	CLD_STOPPED	Child has stopped.
	CLD_CONTINUED	Stopped child has continued.
SIGPOLL	POLL_IN	Data input available.
	POLL_OUT	Output buffers available.
	POLL_MSG	Input message available.
	POLL_ERR	I/O error.
	POLL_PRI	High priority input available.
	POLL_HUP	Device disconnected.
Any	SI_USER	Signal sent by <i>kill()</i> .
	SI_QUEUE	Signal sent by the <i>sigqueue()</i> .
	SI_TIMER	Signal generated by expiration of a timer set by <i>timer_settime()</i> .
	SI_ASYNCIO	Signal generated by completion of an asynchronous I/O request.
	SI_MESGQ	Signal generated by arrival of a message on an empty message queue.

11402 XSI

11403

11404

11405

11406

Implementations may support additional *si_code* values not included in this list, may generate values included in this list under circumstances other than those described in this list, and may contain extensions or limitations that prevent some values from being generated. Implementations do not generate a different value from the ones described in this list for circumstances described in this list.

11407 In addition, the following signal-specific information shall be available:

11408

11409

11410

11411

11412

11413

11414

11415

11416

11417

Signal	Member	Value
SIGILL SIGFPE	void * si_addr	Address of faulting instruction.
SIGSEGV SIGBUS	void * si_addr	Address of faulting memory reference.
SIGCHLD	pid_t si_pid int si_status uid_t si_uid	Child process ID. Exit value or signal. Real user ID of the process that sent the signal.
SIGPOLL	long si_band	Band event for POLL_IN, POLL_OUT, or POLL_MSG.

11418

For some implementations, the value of *si_addr* may be inaccurate.

11419

The following shall be declared as functions and may also be defined as macros:

11420 XSI

11421

11422 XSI

11423

11424

11425

11426

11427

11428

11429 XSI

11430

11431

11432

11433 XSI

11434

11435

11436

11437

11438 XSI

11439

11440

11441 RTS

11442 XSI

11443

11444

11445

11446 RTS

11447

11448

11449 RTS

11450

```
void (*bsd_signal(int, void (*)(int)))(int);
int kill(pid_t, int);
int killpg(pid_t, int);
int pthread_kill(pthread_t, int);
int pthread_sigmask(int, const sigset_t *, sigset_t *);
int raise(int);
int sigaction(int, const struct sigaction *restrict,
              struct sigaction *restrict);
int sigaddset(sigset_t *, int);
int sigaltstack(const stack_t *restrict, stack_t *restrict);
int sigdelset(sigset_t *, int);
int sigemptyset(sigset_t *);
int sigfillset(sigset_t *);
int sighold(int);
int sigignore(int);
int siginterrupt(int, int);
int sigismember(const sigset_t *, int);
void (*signal(int, void (*)(int)))(int);
int sigpause(int);
int sigpending(sigset_t *);
int sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
int sigqueue(pid_t, int, const union sigval);
int sigrelse(int);
void (*sigset(int, void (*)(int)))(int);
int sigstack(struct sigstack *, struct sigstack *); (LEGACY)
int sigsuspend(const sigset_t *);
int sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
                const struct timespec *restrict);
int sigwait(const sigset_t *restrict, int *restrict);
int sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);
```

11451 **APPLICATION USAGE**

11452 None.

11453 **RATIONALE**

11454 None.

11455 **FUTURE DIRECTIONS**

11456 None.

11457 **SEE ALSO**

11458 **<errno.h>**, **<stropts.h>**, **<sys/types.h>**, **<ucontext.h>**, the System Interfaces volume of
11459 IEEE Std. 1003.1-200x, *alarm()*, *bsd_signal()*, *ioctl()*, *kill()*, *killpg()*, *raise()*, *sigaction()*, *sigaddset()*,
11460 *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *siginterrupt()*, *sigismember()*, *signal()*,
11461 *sigpending()*, *sigprocmask()*, *sigqueue()*, *sigsuspend()*, *sigwaitinfo()*, *wait()*, *waitid()*

11462 **CHANGE HISTORY**

11463 First released in Issue 1.

11464 **Issue 4**11465 A reference to **<sys/types.h>** is added for the definition of **pid_t**. This is marked as an extension.

11466 In the list of signals starting with SIGCHLD, the statement “but a system not supporting the job
11467 control option is not obliged to support the functionality of these signals” is removed. This is
11468 because job control is defined as mandatory on Issue 4 conforming implementations.

11469 Reference to implementation-defined abnormal termination routines, such as creation of a core
11470 file, in item ii in the defaults action list is marked as an extension.

11471 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 11472 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 11473 • The DESCRIPTION is changed as follows:
 - 11474 — To define the type **sig_atomic_t**.
 - 11475 — To define the syntax of signal names and functions.
 - 11476 — To combine the two tables of constants.
 - 11477 — SIGFPE is no longer limited to floating-point exceptions, but covers all erroneous
11478 arithmetic operations.

11479 The following change is incorporated for alignment with the ISO C standard:

- 11480 • The *raise()* function is added to the list of functions declared in this header.

11481 **Issue 4, Version 2**

11482 The following changes are incorporated for X/OPEN UNIX conformance:

- 11483 • The SIGTRAP, SIGBUS, SIGSYS, SIGPOLL, SIGPROF, SIGXCPU, SIGXFSZ, SIGURG, and
11484 SIGVTALRM signals are added to the list of signals that are supported on all conforming
11485 implementations.
- 11486 • The *sa_sigaction* member is added to the **sigaction** structure, and a note is added that the
11487 storage used by *sa_handler* and *sa_sigaction* may overlap.
- 11488 • The SA_ONSTACK, SA_RESETHAND, SA_RESTART, SA_SIGINFO, SA_NOCLDWAIT,
11489 SS_ONSTACK, SS_DISABLE, MINSIGSTKSZ, and SIGSTKSZ constants are defined. The
11490 **stack_t**, **sigstack**, and **siginfo** structures are defined.
- 11491 • Definitions are given for the **ucontext_t**, **stack_t**, **sigstack**, and **siginfo_t** types.

- 11492 • A table is provided listing macros that are defined as signal-specific reasons why a signal
11493 was generated. Signal-specific additional information is specified.
- 11494 • The *bsd_signal()*, *killpg()*, *_longjmp()*, *_setjmp()*, *sigaltstack()*, *sighold()*, *sigignore()*,
11495 *siginterrupt()*, *sigpause()*, *sigrelse()*, *sigset()*, and *sigstack()* functions are added to the list of
11496 functions declared in this header.
- 11497 **Issue 5**
- 11498 The DESCRIPTION is updated for alignment with POSIX Realtime Extension and the POSIX
11499 Threads Extension.
- 11500 The default action for SIGURG is changed for i to iii. The function prototype for *sigmask()* is
11501 removed.
- 11502 **Issue 6**
- 11503 The Open Group corrigenda item U035/2 has been applied. In the DESCRIPTION, the wording
11504 for abnormal termination is clarified.
- 11505 The Open Group corrigenda item U028/8 has been applied, correcting the prototype for the
11506 *sigset()* function.
- 11507 The Open Group corrigenda item U026/3 has been applied, correcting the type of the
11508 *sigev_notify_function* function member of the **sigevent** structure.
- 11509 The following new requirements on POSIX implementations derive from alignment with the
11510 Single UNIX Specification:
- 11511 • The SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals are now
11512 mandated. This is also a FIPS requirement.
- 11513 • The **pid_t** definition is mandated.
- 11514 The RT markings are now changed to RTS to denote that the semantics are part of the Realtime
11515 Signals Extension option.
- 11516 The **restrict** keyword is added to the prototypes for *sigaction()*, *sigaltstack()*, *sigprocmask()*,
11517 *sigtimedwait()*, *sigwait()*, and *sigwaitinfo()*.

11518 **NAME**11519 spawn.h — spawn (**REALTIME**)11520 **SYNOPSIS**

11521 SPN #include <spawn.h>

11522

11523 **DESCRIPTION**

11524 The <spawn.h> header shall define the **posix_spawnattr_t** and **posix_spawn_file_actions_t**
 11525 types used in performing spawn operations.

11526 The <spawn.h> header shall define the flags that may be set in a **posix_spawnattr_t** object using
 11527 the *posix_spawnattr_setflags()* function:

11528 POSIX_SPAWN_RESETEIDS

11529 POSIX_SPAWN_SETPGROUP

11530 PS POSIX_SPAWN_SETSCHEDPARAM

11531 POSIX_SPAWN_SETSCHEDULER

11532 POSIX_SPAWN_SETSIGDEF

11533 POSIX_SPAWN_SETSIGMASK

11534 The following shall be declared as functions and may also be declared as macros. Function
 11535 prototypes shall be provided for use with an ISO C standard compiler.

```

11536 int    posix_spawn(pid_t *restrict, const char *restrict,
11537                  const posix_spawn_file_actions_t *,
11538                  const posix_spawnattr_t *restrict, char *const [restrict],
11539                  char *const [restrict]);
11540 int    posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *,
11541                  int);
11542 int    posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *,
11543                  int, int);
11544 int    posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict,
11545                  int, const char *restrict, int, mode_t);
11546 int    posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *);
11547 int    posix_spawn_file_actions_init(posix_spawn_file_actions_t *);
11548 int    posix_spawnattr_destroy(posix_spawnattr_t *);
11549 int    posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,
11550                  sigset_t *restrict);
11551 int    posix_spawnattr_getflags(const posix_spawnattr_t *restrict,
11552                  short *restrict);
11553 int    posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict,
11554                  pid_t *restrict);
11555 PS int    posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict,
11556                  struct sched_param *restrict);
11557 int    posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict,
11558                  int *restrict);
11559 int    posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,
11560                  sigset_t *restrict);
11561 int    posix_spawnattr_init(posix_spawnattr_t *);
11562 int    posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict,
11563                  const sigset_t *restrict);
11564 int    posix_spawnattr_setflags(posix_spawnattr_t *, short);
11565 int    posix_spawnattr_setpgroup(posix_spawnattr_t *, pid_t);

```

11566 PS

```

11567 int    posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,
11568                                     const struct sched_param *restrict);
11569 int    posix_spawnattr_setschedpolicy(posix_spawnattr_t *, int);
11570 int    posix_spawnattr_setsigmask(posix_spawnattr_t *restrict,
11571                                   const sigset_t *restrict);
11572 const posix_spawnattr_t *, char *const [], char *const []);
11573 int    posix_spawn(pid_t *restrict, const char *restrict,
11574                  const posix_spawn_file_actions_t *,
11575                  const posix_spawnattr_t *restrict,
11576                  char *const [restrict], char *const [restrict]);

```

11577 Inclusion of the <spawn.h> header may make visible symbols defined in the <sched.h>, |
11578 <signal.h>, and <sys/types.h> headers. |

11579 **APPLICATION USAGE**

11580 None.

11581 **RATIONALE**

11582 None.

11583 **FUTURE DIRECTIONS**

11584 None.

11585 **SEE ALSO**

11586 <sched.h>, <semaphore.h>, <signal.h>, <sys/types.h>, the System Interfaces volume of |
11587 IEEE Std. 1003.1-200x, *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*, |
11588 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, |
11589 *posix_spawnattr_getschedpolicy()*, *posix_spawnattr_getsigmask()*, *posix_spawnattr_init()*, |
11590 *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setflags()*, *posix_spawnattr_setpgroup()*, |
11591 *posix_spawnattr_setschedparam()*, *posix_spawnattr_setschedpolicy()*, *posix_spawnattr_setsigmask()*, |
11592 *posix_spawn()*, *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*, |
11593 *posix_spawn_file_actions_addopen()*, *posix_spawn_file_actions_destroy()*, |
11594 *posix_spawn_file_actions_init()*, *posix_spawnnp()* |

11595 **CHANGE HISTORY**

11596 First released in Issue 6. Included for alignment with IEEE Std. 1003.1d-1999. |

11597 The **restrict** keyword is added to the prototypes for *posix_spawn()*, |
11598 *posix_spawn_file_actions_addopen()*, *posix_spawnattr_getsigdefault()*, *posix_spawnattr_getflags()*, |
11599 *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*, |
11600 *posix_spawnattr_getsigmask()*, *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setschedparam()*, |
11601 *posix_spawnattr_setsigmask()*, and *posix_spawnnp()*. |

11602 **NAME**11603 **stdarg.h** — handle variable argument list11604 **SYNOPSIS**

11605 #include <stdarg.h>

11606 void va_start(va_list *ap*, *argN*);11607 void va_copy(va_list *dest*, va_list *src*);11608 type va_arg(va_list *ap*, *type*);11609 void va_end(va_list *ap*);11610 **DESCRIPTION**

11611 **CX** The functionality described on this reference page extends the ISO C standard. Applications
11612 shall define the appropriate feature test macro (see the System Interfaces volume of
11613 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
11614 symbols in this header.

11615 The **<stdarg.h>** header contains a set of macros which allows portable functions that accept
11616 variable argument lists to be written. Functions that have variable argument lists (such as
11617 *printf()*) but do not use these macros, are inherently non-portable, as different systems use
11618 different argument-passing conventions.

11619 The type **va_list** is defined for variables used to traverse the list.

11620 The *va_start()* macro is invoked to initialize *ap* to the beginning of the list before any calls to
11621 *va_arg()*.

11622 The *va_copy()* macro initializes *dest* as a copy of *src*, as if the *va_start()* macro had been applied to
11623 *dest* followed by the same sequence of uses of the *va_arg()* macro as had previously been used to
11624 reach the present state of *src*. Neither the *va_copy()* nor *va_start()* macro shall be invoked to
11625 reinitialize *dest* without an intervening invocation of the *va_end()* macro for the same *dest*.

11626 The object *ap* may be passed as an argument to another function; if that function invokes the
11627 *va_arg()* macro with parameter *ap*, the value of *ap* in the calling function is indeterminate and
11628 must be passed to the *va_end()* macro prior to any further reference to *ap*. The parameter *argN* is
11629 the identifier of the rightmost parameter in the variable parameter list in the function definition
11630 (the one just before the ...). If the parameter *argN* is declared with the **register** storage class, with
11631 a function type or array type, or with a type that is not compatible with the type that results after
11632 application of the default argument promotions, the behavior is undefined.

11633 The *va_arg()* macro returns the next argument in the list pointed to by *ap*. Each invocation of
11634 *va_arg()* modifies *ap* so that the values of successive arguments are returned in turn. The *type*
11635 parameter is the type the argument is expected to be. This is the type name specified such that
11636 the type of a pointer to an object that has the specified type can be obtained simply by suffixing
11637 a '*' to type. Different types can be mixed, but it is up to the routine to know what type of
11638 argument is expected.

11639 The *va_end()* macro is used to clean up; it invalidates *ap* for use (unless *va_start()* or *va_copy()* is
11640 invoked again).

11641 Each invocation of the *va_start()* and *va_copy()* macros shall be matched by a corresponding
11642 invocation of the *va_end()* macro in the same function.

11643 Multiple traversals, each bracketed by *va_start()* ... *va_end()*, are possible.

11644 **EXAMPLES**

11645 This example is a possible implementation of *execl()*:

```
11646 #include <stdarg.h>
11647 #define MAXARGS 31
11648 /*
11649  * execl is called by
11650  * execl(file, arg1, arg2, ..., (char *) (0));
11651  */
11652 int execl(const char *file, const char *args, ...)
11653 {
11654     va_list ap;
11655     char *array[MAXARGS];
11656     int argno = 0;
11657     va_start(ap, args);
11658     while (args != 0) {
11659         array[argno++] = args;
11660         args = va_arg(ap, const char *);
11661     }
11662     va_end(ap);
11663     return execv(file, array);
11664 }
```

11665 **APPLICATION USAGE**

11666 It is up to the calling routine to communicate to the called routine how many arguments there
11667 are, since it is not always possible for the called routine to determine this in any other way. For
11668 example, *execl()* is passed a null pointer to signal the end of the list. The *printf()* function can tell
11669 how many arguments are there by the *format* argument.

11670 **RATIONALE**

11671 None.

11672 **FUTURE DIRECTIONS**

11673 None.

11674 **SEE ALSO**

11675 The System Interfaces volume of IEEE Std. 1003.1-200x, *exec()*, *printf()*

11676 **CHANGE HISTORY**

11677 First released in Issue 4. Derived from the ANSI C standard.

11678 **NAME**

11679 stdbool.h — boolean type and values

11680 **SYNOPSIS**

11681 #include <stdbool.h>

11682 **DESCRIPTION**

11683 cx The functionality described on this reference page extends the ISO C standard. Applications
11684 shall define the appropriate feature test macro (see the System Interfaces volume of
11685 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
11686 symbols in this header.

11687 The <stdbool.h> header shall define the following macros:

11688 *bool* Expands to *_Bool*.11689 *true* Expands to the integer constant 1.11690 *false* Expands to the integer constant 0.11691 *__bool_true_false_are_defined*

11692 Expands to the integer constant 1.

11693 An application may undefine and then possibly redefine the macros *bool*, *true*, and *false*.11694 **APPLICATION USAGE**

11695 None.

11696 **RATIONALE**

11697 None.

11698 **FUTURE DIRECTIONS**

11699 The ability to undefine and redefine the macros *bool*, *true*, and *false* is an obsolescent feature and
11700 may be withdrawn in the future.

11701 **SEE ALSO**

11702 None.

11703 **CHANGE HISTORY**

11704 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

11705 **NAME**

11706 **stddef.h** — standard type definitions

11707 **SYNOPSIS**

11708 #include <stddef.h>

11709 **DESCRIPTION**

11710 **cx** The functionality described on this reference page extends the ISO C standard. Applications
 11711 shall define the appropriate feature test macro (see the System Interfaces volume of
 11712 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 11713 symbols in this header.

11714 The <stddef.h> header shall define the following:

11715 **NULL** Null pointer constant.

11716 **offsetof(*type*, *member-designator*)**

11717 Integral constant expression of type **size_t**, the value of which is the offset in bytes
 11718 to the structure member (*member-designator*), from the beginning of its structure
 11719 (*type*).

11720 The <stddef.h> header shall define through **typedef**:

11721 **ptrdiff_t** Signed integer type of the result of subtracting two pointers.

11722 **wchar_t** Integer type whose range of values can represent distinct wide-character codes for
 11723 all members of the largest character set specified among the locales supported by
 11724 the compilation environment: the null character has the code value 0 and each
 11725 member of the Portable Character Set has a code value equal to its value when
 11726 used as the lone character in an integer character constant.

11727 **size_t** Unsigned integer type of the result of the *sizeof* operator.

11728 **APPLICATION USAGE**

11729 None.

11730 **RATIONALE**

11731 None.

11732 **FUTURE DIRECTIONS**

11733 None.

11734 **SEE ALSO**

11735 <wchar.h>, <sys/types.h>

11736 **CHANGE HISTORY**

11737 First released in Issue 4. Derived from the ANSI C standard.

11738 **NAME**

11739 stdint.h — integer types

11740 **SYNOPSIS**

11741 #include <stdint.h>

11742 **DESCRIPTION**

11743 **CX** The functionality described on this reference page extends the ISO C standard. Applications
11744 shall define the appropriate feature test macro (see the System Interfaces volume of
11745 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
11746 symbols in this header.

11747 The <stdint.h> header declares sets of integer types having specified widths, and defines
11748 corresponding sets of macros. It also defines macros that specify limits of integer types
11749 corresponding to types defined in other standard headers.

11750 Types are defined in the following categories:

- 11751 • Integer types having certain exact widths
- 11752 • Integer types having at least certain specified widths
- 11753 • Fastest integer types having at least certain specified widths
- 11754 • Integer types wide enough to hold pointers to objects
- 11755 • Integer types having greatest width

11756 (Some of these types may denote the same type.)

11757 Corresponding macros specify limits of the declared types and construct suitable constants.

11758 For each type described herein that the implementation provides, the <stdint.h> header shall
11759 declare that **typedef** name and define the associated macros. Conversely, for each type described
11760 herein that the implementation does not provide, the <stdint.h> header shall not declare that
11761 **typedef** name, nor shall it define the associated macros. An implementation shall provide those
11762 types described as required, but need not provide any of the others (described as optional).

11763 **Integer Types**

11764 When **typedef** names differing only in the absence or presence of the initial *u* are defined, they
11765 shall denote corresponding signed and unsigned types as described in the ISO/IEC 9899:1999
11766 standard, Section 6.2.5; an implementation providing one of these corresponding types shall also
11767 provide the other.

11768 In the following descriptions, the symbol *N* represents an unsigned decimal integer with no
11769 leading zeros (for example, 8 or 24, but not 04 or 048).

- 11770 • Exact-width integer types

11771 The **typedef** name **intN_t** designates a signed integer type with width *N*, no padding bits,
11772 and a two's-complement representation. Thus, **int8_t** denotes a signed integer type with a
11773 width of exactly 8 bits.

11774 The **typedef** name **uintN_t** designates an unsigned integer type with width *N*. Thus,
11775 **uint24_t** denotes an unsigned integer type with a width of exactly 24 bits.

11776 These types are optional. However, if an implementation provides integer types with widths
11777 of 8, 16, 32, or 64 bits, it shall define the corresponding **typedef** names.

- 11778 • Minimum-width integer types

11779 The **typedef** name **int_leastN_t** designates a signed integer type with a width of at least *N*,
 11780 such that no signed integer type with lesser size has at least the specified width. Thus,
 11781 **int_least32_t** denotes a signed integer type with a width of at least 32 bits.

11782 The **typedef** name **uint_leastN_t** designates an unsigned integer type with a width of at least
 11783 *N*, such that no unsigned integer type with lesser size has at least the specified width. Thus,
 11784 **uint_least16_t** denotes an unsigned integer type with a width of at least 16 bits.

11785 The following types are required:

```
11786     int_least8_t
11787     int_least16_t
11788     int_least32_t
11789     int_least64_t
11790     uint_least8_t
11791     uint_least16_t
11792     uint_least32_t
11793     uint_least64_t
```

11794 All other types of this form are optional.

- 11795 • Fastest minimum-width integer types

11796 Each of the following types designates an integer type that is usually fastest to operate with
 11797 among all integer types that have at least the specified width.

11798 The designated type is not guaranteed to be fastest for all purposes; if the implementation
 11799 has no clear grounds for choosing one type over another, it will simply pick some integer
 11800 type satisfying the signedness and width requirements.

11801 The **typedef** name **int_fastN_t** designates the fastest signed integer type with a width of at
 11802 least *N*. The **typedef** name **uint_fastN_t** designates the fastest unsigned integer type with a
 11803 width of at least *N*.

11804 The following types are required:

```
11805     int_fast8_t
11806     int_fast16_t
11807     int_fast32_t
11808     int_fast64_t
11809     uint_fast8_t
11810     uint_fast16_t
11811     uint_fast32_t
11812     uint_fast64_t
```

11813 All other types of this form are optional.

- 11814 • Integer types capable of holding object pointers

11815 The following type designates a signed integer type with the property that any valid pointer
 11816 to **void** can be converted to this type, then converted back to a pointer to **void**, and the result
 11817 will compare equal to the original pointer:

```
11818     intptr_t
```

11819 The following type designates an unsigned integer type with the property that any valid
 11820 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and
 11821 the result will compare equal to the original pointer:

11822 **uintptr_t**

11823 These types are optional.

11824 • Greatest-width integer types

11825 The following type designates a signed integer type capable of representing any value of any

11826 signed integer type:

11827 **intmax_t**

11828 The following type designates an unsigned integer type capable of representing any value of

11829 any unsigned integer type:

11830 **uintmax_t**

11831 These types are required.

11832 **Limits of Specified-Width Integer Types**

11833 The following object-like macros specify the minimum and maximum limits of the types

11834 declared in the <stdint.h> header. Each macro name corresponds to a similar type name in

11835 **Integer Types** (on page 352).

11836 Each instance of any defined macro shall be replaced by a constant expression suitable for use in

11837 #if preprocessing directives, and this expression shall have the same type as would an

11838 expression that is an object of the corresponding type converted according to the integer

11839 promotions. Its implementation-defined value shall be equal to or greater in magnitude

11840 (absolute value) than the corresponding value given below, with the same sign, except where

11841 stated to be exactly the given value.

11842 • Limits of exact-width integer types

11843 — Minimum values of exact-width signed integer types:

11844 {INTN_MIN} Exactly $-(2^{N-1})$

11845 — Maximum values of exact-width signed integer types:

11846 {INTN_MAX} Exactly $2^{N-1} - 1$

11847 — Maximum values of exact-width unsigned integer types:

11848 {UINTN_MAX} Exactly $2^N - 1$

11849 • Limits of minimum-width integer types

11850 — Minimum values of minimum-width signed integer types:

11851 {INT_LEASTN_MIN} $-(2^{N-1} - 1)$

11852 — Maximum values of minimum-width signed integer types:

11853 {INT_LEASTN_MAX} $2^N - 1$

11854 — Maximum values of minimum-width unsigned integer types:

11855 {UINT_LEASTN_MAX} $2^N - 1$

11856 • Limits of fastest minimum-width integer types

11857 — Minimum values of fastest minimum-width signed integer types:

11858 {INT_FASTN_MIN} $-(2^{N-1} - 1)$

11859	—	Maximum values of fastest minimum-width signed integer types:	
11860		{INT_FASTN_MAX}	$2^{N-1} - 1$
11861	—	Maximum values of fastest minimum-width unsigned integer types:	
11862		{UINT_FASTN_MAX}	$2^N - 1$
11863	•	Limits of integer types capable of holding object pointers	
11864	—	Minimum value of pointer-holding signed integer type:	
11865		{INTPTR_MIN}	$-(2^{15} - 1)$
11866	—	Maximum value of pointer-holding signed integer type:	
11867		{INTPTR_MAX}	$2^{15} - 1$
11868	—	Maximum value of pointer-holding unsigned integer type:	
11869		{UINTPTR_MAX}	$2^{16} - 1$
11870	•	Limits of greatest-width integer types	
11871	—	Minimum value of greatest-width signed integer type:	
11872		{INTMAX_MIN}	$-(2^{63} - 1)$
11873	—	Maximum value of greatest-width signed integer type:	
11874		{INTMAX_MAX}	$2^{63} - 1$
11875	—	Maximum value of greatest-width unsigned integer type:	
11876		{UINTMAX_MAX}	$2^{64} - 1$
11877		Limits of Other Integer Types	
11878		The following object-like macros specify the minimum and maximum limits of integer types	
11879		corresponding to types defined in other standard headers.	
11880		Each instance of these macros shall be replaced by a constant expression suitable for use in #if	
11881		preprocessing directives, and this expression shall have the same type as would an expression	
11882		that is an object of the corresponding type converted according to the integer promotions. Its	
11883		implementation-defined value shall be equal to or greater in magnitude (absolute value) than	
11884		the corresponding value given below, with the same sign.	
11885	•	Limits of ptrdiff_t :	
11886		{PTRDIFF_MIN}	-65535
11887		{PTRDIFF_MAX}	+65535
11888	•	Limits of sig_atomic_t :	
11889		{SIG_ATOMIC_MIN}	See below.
11890		{SIG_ATOMIC_MAX}	See below.
11891	•	Limit of size_t :	
11892		{SIZE_MAX}	65535
11893	•	Limits of wchar_t :	
11894		{WCHAR_MIN}	See below.

11895 {WCHAR_MAX} See below.

11896 • Limits of **wint_t**:

11897 {WINT_MIN} See below.

11898 [WINT_MAX] See below.

11899 If **sig_atomic_t** (see the <signal.h> header) is defined as a signed integer type, the value of
11900 {SIG_ATOMIC_MIN} shall be no greater than -127 and the value of {SIG_ATOMIC_MAX} shall
11901 be no less than 127; otherwise, **sig_atomic_t** is defined as an unsigned integer type, and the
11902 value of {SIG_ATOMIC_MIN} shall be 0 and the value of {SIG_ATOMIC_MAX} shall be no less
11903 than 255.

11904 If **wchar_t** (see the <stddef.h> header) is defined as a signed integer type, the value of
11905 {WCHAR_MIN} shall be no greater than -127 and the value of {WCHAR_MAX} shall be no less
11906 than 127; otherwise, **wchar_t** is defined as an unsigned integer type, and the value of
11907 {WCHAR_MIN} shall be 0 and the value of {WCHAR_MAX} shall be no less than 255.

11908 If **wint_t** (see the <wchar.h> header) is defined as a signed integer type, the value of
11909 {WINT_MIN} shall be no greater than -32767 and the value of {WINT_MAX} shall be no less
11910 than 32767; otherwise, **wint_t** is defined as an unsigned integer type, and the value of
11911 {WINT_MIN} shall be 0 and the value of {WINT_MAX} shall be no less than 65535.

11912 **Macros for Integer Constants**

11913 The following function-like macros expand to integer constants suitable for initializing objects
11914 that have integer types corresponding to types defined in the <stdint.h> header. Each macro
11915 name corresponds to a similar type name listed under *Minimum-width integer types* and *Greatest-*
11916 *width integer types*.

11917 The argument in any instance of these macros shall be a decimal, octal, or hexadecimal constant
11918 with a value that does not exceed the limits for the corresponding type.

11919 • Macros for minimum-width integer constants

11920 Each of the following macros expands to an integer constant having the value specified by its
11921 argument and a type with at least the specified width.

11922 The macro *INTN_C(value)* shall expand to a signed integer constant with the specified value
11923 and type **int_leastN_t**. The macro *UINTN_C(value)* shall expand to an unsigned integer
11924 constant with the specified value and type **uint_leastN_t**. For example, if **uint_least64_t** is a
11925 name for the type **unsigned long long**, then *UINT64_C(0x123)* might expand to the integer
11926 constant 0x123ULL.

11927 • Macros for greatest-width integer constants

11928 The following macro expands to an integer constant having the value specified by its
11929 argument and the type **intmax_t**:

11930 INTMAX_C(value)

11931 The following macro expands to an integer constant having the value specified by its
11932 argument and the type **uintmax_t**:

11933 UINMAX_C(value)

11934 **APPLICATION USAGE**

11935 None.

11936 **RATIONALE**

11937 The <stdint.h> header is a subset of the <inttypes.h> header more suitable for use in
11938 freestanding environments, which might not support the formatted I/O functions. In some
11939 environments, if the formatted conversion support is not wanted, using this header instead of
11940 the <inttypes.h> header avoids defining such a large number of macros.

11941 **FUTURE DIRECTIONS**

11942 **typedef** names beginning with **int** or **uint** and ending with **_t** may be added to the types defined
11943 in the <stdint.h> header. Macro names beginning with **INT** or **UINT** and ending with **_MAX**,
11944 **_MIN**, or **_C** may be added to the macros defined in the <stdint.h> header.

11945 **SEE ALSO**

11946 <signal.h>, <stddef.h>, <wchar.h>, <inttypes.h>

11947 **CHANGE HISTORY**

11948 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

11949 **NAME**11950 `stdio.h` — standard buffered input/output11951 **SYNOPSIS**11952 `#include <stdio.h>`11953 **DESCRIPTION**

11954 *CX* The functionality described on this reference page extends the ISO C standard. Applications
 11955 shall define the appropriate feature test macro (see the System Interfaces volume of
 11956 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 11957 symbols in this header.

11958 The `<stdio.h>` header shall define the following macro names as positive integral constant
 11959 expressions:

11960	<code>{BUFSIZ}</code>	Size of <code><stdio.h></code> buffers.
11961	<code>{FILENAME_MAX}</code>	Maximum size in bytes of the longest file name string that the 11962 implementation guarantees can be opened.
11963	<code>{FOPEN_MAX}</code>	Number of streams which the implementation guarantees can be open 11964 simultaneously. The value is at least eight.
11965	<code>{_IOFBF}</code>	Input/output fully buffered.
11966	<code>{_IOLBF}</code>	Input/output line buffered.
11967	<code>{_IONBF}</code>	Input/output unbuffered.
11968	<code>{L_ctermid}</code>	Maximum size of character array to hold <code>ctermid()</code> output.
11969	<code>{L_tmpnam}</code>	Maximum size of character array to hold <code>tmpnam()</code> output.
11970	<code>{SEEK_CUR}</code>	Seek relative to current position.
11971	<code>{SEEK_END}</code>	Seek relative to end-of-file.
11972	<code>{SEEK_SET}</code>	Seek relative to start-of-file.
11973	<code>{TMP_MAX}</code>	Minimum number of unique file names generated by <code>tmpnam()</code> . 11974 <i>XSI</i> Maximum number of times an application can call <code>tmpnam()</code> reliably. 11975 The value of <code>{TMP_MAX}</code> is at least 10,000.

11976 The following macro name shall be defined as a negative integral constant expression:

11977 `EOF` End-of-file return value.

11978 The following macro name shall be defined as a null pointer constant:

11979 `NULL` Null pointer.

11980 The following macro name shall be defined as a string constant:

11981 *XSI* `P_tmpdir` Default directory prefix for `tmpnam()`.

11982 The following macro names shall be defined as expressions of type pointer to `FILE`:

11983 `stderr` Standard error output stream.

11984 `stdin` Standard input stream.

11985 `stdout` Standard output stream.

11986 The following data types shall be defined through **typedef**:

11987	FILE	A structure containing information about a file.
11988	fpos_t	Type containing all information needed to specify uniquely every
11989		position within a file.
11990 XSI	va_list	As described in <stdarg.h>.
11991	size_t	As described in <stddef.h>.
11992	The following shall be declared as functions and may also be defined as macros. Function	
11993	prototypes shall be provided for use with an ISO C standard compiler.	
11994	void	clearerr(FILE *);
11995	char	*ctermid(char *);
11996	int	fclose(FILE *);
11997	FILE	*fdopen(int, const char *);
11998	int	feof(FILE *);
11999	int	ferror(FILE *);
12000	int	fflush(FILE *);
12001	int	fgetc(FILE *);
12002	int	fgetpos(FILE *restrict, fpos_t *restrict);
12003	char	*fgets(char *restrict, int, FILE *restrict);
12004	int	fileno(FILE *);
12005 TSF	void	flockfile(FILE *);
12006	FILE	*fopen(const char *restrict, const char *restrict);
12007	int	fprintf(FILE *restrict, const char *restrict, ...);
12008	int	fputc(int, FILE *);
12009	int	fputs(const char *restrict, FILE *restrict);
12010	size_t	fread(void *restrict, size_t, size_t, FILE *restrict);
12011	FILE	*freopen(const char *restrict, const char *restrict,
12012		FILE *restrict);
12013	int	fscanf(FILE *restrict, const char *restrict, ...);
12014	int	fseek(FILE *, long, int);
12015 XSI	int	fseeko(FILE *, off_t, int);
12016	int	fsetpos(FILE *, const fpos_t *);
12017	long	ftell(FILE *);
12018 XSI	off_t	ftello(FILE *);
12019 TSF	int	ftrylockfile(FILE *);
12020	void	funlockfile(FILE *);
12021	size_t	fwrite(const void *restrict, size_t, size_t, FILE *restrict);
12022	int	getc(FILE *);
12023	int	getchar(void);
12024 TSF	int	getc_unlocked(FILE *);
12025	int	getchar_unlocked(void);
12026	char	*gets(char *);
12027	int	pclose(FILE *);
12028	void	perror(const char *);
12029	FILE	*popen(const char *, const char *);
12030	int	printf(const char *restrict, ...);
12031	int	putc(int, FILE *);
12032	int	putchar(int);
12033 TSF	int	putc_unlocked(int, FILE *);
12034	int	putchar_unlocked(int);
12035	int	puts(const char *);
12036	int	remove(const char *);

```

12037     int        rename(const char *, const char *);
12038     void        rewind(FILE *);
12039     int         scanf(const char *restrict, ...);
12040     void        setbuf(FILE *restrict, char *restrict);
12041     int         setvbuf(FILE *restrict, char *restrict, int, size_t);
12042 XSI     int         snprintf(char *restrict, size_t, const char *restrict, ...);
12043     int         sprintf(char *restrict, const char *restrict, ...);
12044     int         sscanf(const char *restrict, const char *restrict, int ...);
12045 XSI     char        *tempnam(const char *, const char *);
12046     FILE        *tmpfile(void);
12047     char        *tmpnam(char *);
12048     int         ungetc(int, FILE *);
12049     int         vfprintf(FILE *restrict, const char *restrict, va_list);
12050     int         vfscanf(FILE *restrict, const char *restrict, va_list);
12051     int         vprintf(const char *restrict, va_list);
12052     int         vscanf(const char *restrict, va_list);
12053 XSI     int         vsnprintf(char *restrict, size_t, const char *restrict, va_list);
12054     int         vsprintf(char *restrict, const char *restrict, va_list);
12055     int         vsscanf(const char *restrict, const char *restrict, va_list arg);

```

12056 XSI **Inclusion of the <stdio.h> header may also make visible all symbols from <stddef.h>.**

12057 **APPLICATION USAGE**

12058 None.

12059 **RATIONALE**

12060 None.

12061 **FUTURE DIRECTIONS**

12062 None.

12063 **SEE ALSO**

12064 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *clearerr()*, *ctermid()*,
12065 *fclose()*, *fdopen()*, *fgetc()*, *fgetpos()*, *ferror()*, *feof()*, *fflush()*, *fgets()*, *fileno()*, *flockfile()*, *fopen()*,
12066 *fputc()*, *fputs()*, *fread()*, *freopen()*, *fseek()*, *fsetpos()*, *ftell()*, *fwrite()*, *getc()*, *getc_unlocked()*,
12067 *getwchar()*, *getchar()*, *getopt()*, *gets()*, *pclose()*, *perror()*, *popen()*, *printf()*, *putc()*, *putchar()*, *puts()*,
12068 *putwchar()*, *remove()*, *rename()*, *rewind()*, *scanf()*, *setbuf()*, *setvbuf()*, *sscanf()*, *stdin()*, *system()*,
12069 *tempnam()*, *tmpfile()*, *tmpnam()*, *ungetc()*, *vfscanf()*, *vscanf()*, *vprintf()*, *vsscanf()*

12070 **CHANGE HISTORY**

12071 First released in Issue 1. Derived from Issue 1 of the SVID.

12072 **Issue 4**

12073 The constant {L_cuserid} and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked
12074 as extensions and TO BE WITHDRAWN.

12075 The minimum allowable value of {TMP_MAX}, 10,000 on XSI-conformant systems, has been
12076 marked as an extension.

12077 The P_tmpdir constant is moved from the APPLICATION USAGE section to the DESCRIPTION
12078 and marked as an extension. The remainder of the APPLICATION USAGE section is removed.

12079 References to the *va_list* and *size_t* types are added to the DESCRIPTION.

12080 Function declarations of the *cuserid()*, *getopt()*, and *tempnam()* functions and the *va_list* type are
12081 marked as extensions.

- 12082 The *cuserid()* and *getopt()* functions are marked TO BE WITHDRAWN.
- 12083 A warning is added indicating that inclusion of <stdio.h> may also make visible all symbols
12084 from <stddef.h>.
- 12085 The following changes are incorporated for alignment with the ISO C standard:
- 12086 • The function declarations in this header are expanded to full ISO C standard prototypes.
 - 12087 • The DESCRIPTION is restructured to group lists of macro names according to how they are
12088 defined by an implementation (for example, whether they are integral constant expressions,
12089 pointer constants, or string constants).
 - 12090 • The constant {FILENAME_MAX} is added to the list of integral constant expressions. The
12091 text of {FOPEN_MAX} has also been changed for consistency with the ISO C standard.
 - 12092 • The data type **fpos_t** is moved from the APPLICATION USAGE section to the
12093 DESCRIPTION.
 - 12094 • The *fgetpos()* and *fsetpos()* functions are added to the list of functions declared in this header.
- 12095 **Issue 5**
- 12096 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.
- 12097 Large File System extensions are added.
- 12098 The constant {L_cuserid} and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked |
12099 as extensions and LEGACY.
- 12100 The *cuserid()* and *getopt()* functions are marked LEGACY.
- 12101 **Issue 6**
- 12102 The constant {L_cuserid} and the external variables *optarg*, *opterr*, *optind*, and *optopt* are removed |
12103 as they were previously marked LEGACY.
- 12104 The *cuserid()* and *getopt()* functions are removed as they were previously marked LEGACY.
- 12105 Several functions are marked as part of the _POSIX_THREAD_SAFE_FUNCTIONS option. |
- 12106 This reference page is updated to align with the ISO/IEC 9899:1999 standard. |

12107 NAME

12108 stdlib.h — standard library definitions

12109 SYNOPSIS

12110 #include <stdlib.h>

12111 DESCRIPTION

12112 cx The functionality described on this reference page extends the ISO C standard. Applications
12113 shall define the appropriate feature test macro (see the System Interfaces volume of
12114 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
12115 symbols in this header.

12116 The <stdlib.h> header shall define the following macro names:

12117 EXIT_FAILURE Unsuccessful termination for *exit()*; evaluates to a non-zero value.

12118 EXIT_SUCCESS Successful termination for *exit()*; evaluates to 0.

12119 NULL Null pointer.

12120 {RAND_MAX} Maximum value returned by *rand()*; at least 32,767.

12121 {MB_CUR_MAX} Integer expression whose value is the maximum number of bytes in a
12122 character specified by the current locale.

12123 The following data types shall be defined through **typedef**:

12124 **div_t** Structure type returned by the *div()* function.

12125 **ldiv_t** Structure type returned by the *ldiv()* function.

12126 **lldiv_t** Structure type returned by the *lldiv()* function.

12127 **size_t** As described in <stddef.h>.

12128 **wchar_t** As described in <stddef.h>.

12129 In addition, the following symbolic names and macros shall be defined as in <sys/wait.h>, for
12130 use in decoding the return value from *system()*:

12131 xsi WNOHANG

12132 WUNTRACED

12133 WEXITSTATUS

12134 WIFEXITED

12135 WIFSIGNALED

12136 WIFSTOPPED

12137 WSTOPSIG

12138 WTERMSIG

12139

12140 The following shall be declared as functions and may also be defined as macros. Function
12141 prototypes shall be provided for use with an ISO C standard compiler.

12142 void _Exit(int);

12143 xsi long a64l(const char *);

12144 void abort(void);

12145 int abs(int);

12146 int atexit(void (*)(void));

12147 double atof(const char *);

12148 int atoi(const char *);

12149 long atol(const char *);

```

12150     long long    atoll(const char *);
12151     void        *bsearch(const void *, const void *, size_t, size_t,
12152                       int (*)(const void *, const void *));
12153     void        *calloc(size_t, size_t);
12154     div_t       div(int, int);
12155 XSI     double    drand48(void);
12156     char        *ecvt(double, int, int *restrict, int *restrict); (LEGACY)
12157     double     erand48(unsigned short[3]);
12158     void        exit(int);
12159 XSI     char        *fcvt(double, int, int *restrict, int *restrict); (LEGACY)
12160     void        free(void *);
12161 XSI     char        *gcvt(double, int, char *); (LEGACY)
12162     char        *getenv(const char *);
12163 XSI     int        getsubopt(char **, char *const *, char **);
12164     int        grantpt(int);
12165     char        *initstate(unsigned, char *, size_t);
12166     long        jrand48(unsigned short[3]);
12167     char        *l64a(long);
12168     long        labs(long);
12169 XSI     void        lcong48(unsigned short[7]);
12170     ldiv_t     ldiv(long, long);
12171     long long  llabs(long long);
12172 XSI     long        lrand48(void);
12173     void        *malloc(size_t);
12174     int        mblen(const char *, size_t);
12175     size_t     mbstowcs(wchar_t *restrict, const char *restrict, size_t);
12176     int        mbtowc(wchar_t *restrict, const char *restrict, size_t);
12177 XSI     char        *mktemp(char *); (LEGACY)
12178     int        mkstemp(char *);
12179     long        mrand48(void);
12180     long        nrand48(unsigned short[3]);
12181 ADV     int        posix_memalign(void **, size_t, size_t);
12182 XSI     char        *ptsname(int);
12183     int        putenv(char *);
12184     void        qsort(void *, size_t, size_t, int (*)(const void *,
12185                       const void *));
12186     int        rand(void);
12187 TSF     int        rand_r(unsigned *);
12188 XSI     long        random(void);
12189     void        *realloc(void *, size_t);
12190 XSI     char        *realpath(const char *restrict, char *restrict);
12191     unsigned short seed48(unsigned short[3]);
12192     int        setenv(const char *, const char *, int);
12193     void        setkey(const char *);
12194     char        *setstate(const char *);
12195     void        srand(unsigned);
12196 XSI     void        srand48(long);
12197     void        srandom(unsigned);
12198     double    strtod(const char *restrict, char **restrict);
12199     float     strtod(const char *restrict, char **restrict);
12200     long        strtol(const char *restrict, char **restrict, int);
12201     long double strtold(const char *restrict, char **restrict);

```

```

12202     long long      strtoll(const char *restrict, char **restrict, int);
12203     unsigned long  strtoul(const char *restrict, char **restrict, int);
12204     long long      strtoull(const char *restrict, char **restrict, int);
12205     int            system(const char *);
12206 XSI    int            unlockpt(int);
12207     int            unsetenv(const char *);
12208     size_t         wcstombs(char *restrict, const wchar_t *restrict, size_t);
12209     int            wctomb(char *, wchar_t);

```

12210 XSI Inclusion of the **<stdlib.h>** header may also make visible all symbols from **<stddef.h>**,
12211 **<limits.h>**, **<math.h>**, and **<sys/wait.h>**.

12212 APPLICATION USAGE

12213 None.

12214 RATIONALE

12215 None.

12216 FUTURE DIRECTIONS

12217 None.

12218 SEE ALSO

12219 **<sys/types.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, *_Exit()*, *a64l()*, *abort()*,
12220 *abs()*, *atexit()*, *atof()*, *atoi()*, *atol()*, *atoll()*, *bsearch()*, *calloc()*, *div()*, *drand48()*, *erand48()*, *exit()*,
12221 *free()*, *getenv()*, *getsubopt()*, *grantpt()*, *initstate()*, *jrand48()*, *l64a()*, *labs()*, *lcong48()*, *ldiv()*, *llabs()*,
12222 *lldiv()*, *lrand48()*, *malloc()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *mkstemp()*, *mrand48()*, *nrand48()*,
12223 *posix_memalign()*, *ptsname()*, *putenv()*, *qsort()*, *rand()*, *realloc()*, *realpath()*, *setstate()*, *srand()*,
12224 *srand48()*, *srandom()*, *strtod()*, *strtof()*, *strtol()*, *strtold()*, *strtoll()*, *strtoul()*, *strtoull()*, *unlockpt()*,
12225 *wctombs()*, *wctomb()*

12226 CHANGE HISTORY

12227 First released in Issue 3.

12228 Issue 4

12229 A reference is added to **<stddef.h>** and **<wchar.h>** for the definition of **size_t**.

12230 A reference is added to **<sys/wait.h>** for definitions of the symbolic names and macros defined
12231 for decoding the return value from the *system()* function. This reference and the symbolic names
12232 and macros are marked as an extension.

12233 The names *drand48()*, *erand48()*, *jrand48()*, *lcong48()*, *lrand48()*, *mrand48()*, *nrand48()*, *putenv()*,
12234 *seed48()*, *setkey()*, and *srand48()* are added to the list of functions declared in this header and
12235 marked as extensions.

12236 A warning is added indicating that inclusion of **<stdlib.h>** may also make visible all symbols
12237 from **<stddef.h>**, **<limits.h>**, **<math.h>**, and **<sys/wait.h>**.

12238 The APPLICATION USAGE section is removed.

12239 The following changes are incorporated for alignment with the ISO C standard:

- 12240 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 12241 • The maximum value of {RAND_MAX} is defined.
- 12242 • The name {MB_CUR_MAX} is added to the list of macro names defined in this header, while
12243 **div_t** and **ldiv_t** are added to the list of defined types.
- 12244 • The names *atexit()*, *div()*, *labs()*, *ldiv()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *strtoul()*, *wctombs()*,
12245 and *wctomb()* are added to the list of functions declared in this header.

12246 **Issue 4, Version 2**

12247 For X/OPEN UNIX conformance, the *a64l()*, *ecvt()*, *fcvt()*, *gcvt()*, *getsubopt()*, *grantpt()*,
12248 *initstate()*, *l64a()*, *mktemp()*, *mkstemp()*, *ptsname()*, *random()*, *realpath()*, *setstate()*, *srandom()*,
12249 *ttyslot()*, *unlockpt()*, and *valloc()* functions are added to the list of functions declared in this
12250 header.

12251 **Issue 5**

12252 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

12253 The *ttyslot()* and *valloc()* functions are marked LEGACY.

12254 The type of the third argument to *initstate()* is changed from **int** to **size_t**. The type of the return
12255 value from *setstate()* is changed from **char** to **char***, and the type of the first argument is changed
12256 from **char*** to **const char***.

12257 **Issue 6**

12258 The Open Group corrigenda item U021/1 has been applied, correcting the prototype for
12259 *realpath()* to be consistent with the reference page.

12260 The Open Group corrigenda item U028/13 has been applied, correcting the prototype for
12261 *putenv()* to be consistent with the reference page.

12262 The *rand_r()* function is marked as part of the `_POSIX_THREAD_SAFE_FUNCTIONS` option.

12263 Function prototypes for *setenv()* and *unsetenv()* are added.

12264 The *posix_memalign()* function is added for alignment with IEEE Std. 1003.1d-1999. |

12265 This reference page is updated to align with the ISO/IEC 9899:1999 standard. |

12266 The *ecvt()*, *fcvt()*, *gcvt()*, and *mktemp()* functions are marked LEGACY. |

12267 **NAME**

12268 string.h — string operations

12269 **SYNOPSIS**

12270 #include <string.h>

12271 **DESCRIPTION**

12272 CX The functionality described on this reference page extends the ISO C standard. Applications
 12273 shall define the appropriate feature test macro (see the System Interfaces volume of
 12274 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 12275 symbols in this header.

12276 The **<string.h>** header shall define the following:

12277 NULL Null pointer constant.

12278 **size_t** As described in **<stddef.h>**.

12279 The following shall be declared as functions and may also be defined as macros. Function
 12280 prototypes shall be provided for use with an ISO C standard compiler.

12281 XSI void *memcpy(void *restrict, const void *restrict, int, size_t);

12282 void *memchr(const void *, int, size_t);

12283 int memcmp(const void *, const void *, size_t);

12284 void *memcpy(void *restrict, const void *restrict, size_t);

12285 void *memmove(void *, const void *, size_t);

12286 void *memset(void *, int, size_t);

12287 char *strcat(char *restrict, const char *restrict);

12288 char *strchr(const char *, int);

12289 int strcmp(const char *, const char *);

12290 int strcoll(const char *, const char *);

12291 char *strcpy(char *restrict, const char *restrict);

12292 size_t strcspn(const char *, const char *);

12293 XSI char *strdup(const char *);

12294 char *strerror(int);

12295 size_t strlen(const char *);

12296 char *strncat(char *restrict, const char *restrict, size_t);

12297 int strncmp(const char *, const char *, size_t);

12298 char *strncpy(char *restrict, const char *restrict, size_t);

12299 char *strpbrk(const char *, const char *);

12300 char *strrchr(const char *, int);

12301 size_t strspn(const char *, const char *);

12302 char *strstr(const char *, const char *);

12303 char *strtok(char *restrict, const char *restrict);

12304 TSF char *strtok_r(char *, const char *, char **);

12305 size_t strxfrm(char *restrict, const char *restrict, size_t);

12306 XSI **Inclusion of the **<string.h>** header may also make visible all symbols from **<stddef.h>**.**

12307 **APPLICATION USAGE**

12308 None.

12309 **RATIONALE**

12310 None.

12311 **FUTURE DIRECTIONS**

12312 None.

12313 **SEE ALSO**

12314 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *memccpy()*, *memchr()*,
 12315 *memcmp()*, *memcpy()*, *memmove()*, *memset()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*, *strcpy()*,
 12316 *strcspn()*, *strdup()*, *strerror()*, *strlen()*, *strncat()*, *strncmp()*, *strncpy()*, *strpbrk()*, *strrchr()*, *strspn()*,
 12317 *strstr()*, *strtok()*, *strxfrm()*

12318 **CHANGE HISTORY**

12319 First released in Issue 1. Derived from Issue 1 of the SVID.

12320 **Issue 4**12321 A reference is added to <stddef.h> for the definition of **size_t**.12322 The *memccpy()* function is marked as an extension.

12323 A warning is added indicating that inclusion of <string.h> may also make visible all symbols
 12324 from <stddef.h>.

12325 The APPLICATION USAGE section is removed.

12326 The following changes are incorporated for alignment with the ISO C standard:

- 12327 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 12328 • The name *memmove()* is added to the list of functions declared in this header.

12329 **Issue 4, Version 2**

12330 For X/OPEN UNIX conformance, the *strdup()* function is added to the list of functions declared
 12331 in this header.

12332 **Issue 5**

12333 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

12334 **Issue 6**12335 The *strtok_r()* function is marked as part of the `_POSIX_THREAD_SAFE_FUNCTIONS` option.

12336 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

12337 **NAME**

12338 strings.h — string operations

12339 **SYNOPSIS**

12340 XSI #include <strings.h>

12341

12342 **DESCRIPTION**

12343 The following shall be declared as functions and may also be defined as macros. Function
12344 prototypes shall be provided for use with an ISO C standard compiler.

12345 int bcmp(const void *, const void *, size_t); (**LEGACY**)

12346 void bcopy(const void *, void *, size_t); (**LEGACY**)

12347 void bzero(void *, size_t); (**LEGACY**)

12348 int ffs(int);

12349 char *index(const char *, int); (**LEGACY**)

12350 char *rindex(const char *, int); (**LEGACY**)

12351 int strcasecmp(const char *, const char *);

12352 int strncasecmp(const char *, const char *, size_t);

12353 The `size_t` type shall be defined through `typedef` as described in `<stddef.h>`.

12354 **APPLICATION USAGE**

12355 None.

12356 **RATIONALE**

12357 None.

12358 **FUTURE DIRECTIONS**

12359 None.

12360 **SEE ALSO**

12361 `<stddef.h>`, the System Interfaces volume of IEEE Std. 1003.1-200x, `ffs()`, `strcasecmp()`,

12362 `strncasecmp()`

12363 **CHANGE HISTORY**

12364 First released in Issue 4, Version 2.

12365 **Issue 6**

12366 The Open Group corrigenda item U021/2 has been applied, correcting the prototype for `index()`
12367 to be consistent with the reference page.

12368 The `bcmp()`, `bcopy()`, `bzero()`, `index()`, and `rindex()` functions are marked LEGACY.

12369 **NAME**

12370 `stropts.h` — STREAMS interface (**STREAMS**)

12371 **SYNOPSIS**

12372 XSR `#include <stropts.h>`

12373

12374 **DESCRIPTION**

12375 The `<stropts.h>` header shall define the **bandinfo** structure that includes at least the following
 12376 members:

12377 `unsigned char bi_pri`
 12378 `int bi_flag`

12379 The `<stropts.h>` header shall define the **strpeek** structure that includes at least the following
 12380 members:

12381 `struct strbuf ctlbuf`
 12382 `struct strbuf databuf`
 12383 `t_uscalar_t flags`

12384 The `<stropts.h>` header shall define the **strbuf** structure that includes at least the following
 12385 members:

12386 `int maxlen` Maximum buffer length.
 12387 `int len` Length of data.
 12388 `char *buf` Pointer to buffer.

12389 The `<stropts.h>` header shall define the **strfdinsert** structure that includes at least the following
 12390 members:

12391 `struct strbuf ctlbuf`
 12392 `struct strbuf databuf`
 12393 `t_uscalar_t flags`
 12394 `int fildes`
 12395 `int offset`

12396 The `<stropts.h>` header shall define the **striocctl** structure that includes at least the following
 12397 members:

12398 `int ic_cmd`
 12399 `int ic_timeout`
 12400 `int ic_len`
 12401 `char *ic_dp`

12402 The `<stropts.h>` header shall define the **strrecvfd** structure that includes at least the following
 12403 members:

12404 `int fd`
 12405 `uid_t uid`
 12406 `gid_t gid`

12407 The **uid_t** and **gid_t** types shall be defined through **typedef** as described in `<sys/types.h>`.

12408 The **t_uscalar_t** type shall be defined as described in the referenced XNS, Issue 5 specification,
 12409 `<xti.h>`.

12410 The `<stropts.h>` header shall define the **str_list** structure that includes at least the following
 12411 members:

```

12412         int                sl_nmods
12413         struct str_mlist *sl_modlist

```

12414 The <stropts.h> header shall define the **str_mlist** structure that includes at least the following
 12415 member:

```

12416         char                l_name[ FMNAMESZ+1 ]

```

12417 At least the following macros shall be defined for use as the *request* argument to *ioctl()*:

12418 **I_PUSH** Push STREAMS module onto the top of the current STREAM, just below the
 12419 STREAM head.

12420 **I_POP** Remove STREAMS module from just below the STREAM head.

12421 **I_LOOK** Retrieve the name of the module just below the STREAM head and place it in
 12422 a character string. At least the following macros shall be defined for use as the
 12423 *arg* argument:

12424 **FMNAMESZ** The minimum size in bytes of the buffer referred to by the
 12425 *arg* argument.

12426 **I_FLUSH** This request flushes all input and/or output queues, depending on the value
 12427 of the *arg* argument. At least the following macros shall be defined for use as
 12428 the *arg* argument:

12429 **FLUSHR** Flush read queues.

12430 **FLUSHW** Flush write queues.

12431 **FLUSHRW** Flush read and write queues.

12432 **I_FLUSHBAND** Flush only band specified.

12433 **I_SETSIG** Informs the STREAM head that the process wants the SIGPOLL signal issued
 12434 (see *signal()*) when a particular event has occurred on the STREAM.

12435 The <stropts.h> header shall define the following possible values for *arg* when
 12436 **I_SETSIG** is specified:

12437 **S_RDNORM** A normal (priority band set to 0) message has arrived at the
 12438 head of a STREAM head read queue.

12439 **S_RDBAND** A message with a non-zero priority band has arrived at the
 12440 head of a STREAM head read queue.

12441 **S_INPUT** A message, other than a high-priority message, has arrived
 12442 at the head of a STREAM head read queue.

12443 **S_HIPRI** A high-priority message is present on a STREAM head read
 12444 queue.

12445 **S_OUTPUT** The write queue for normal data (priority band 0) just
 12446 below the STREAM head is no longer full. This notifies the
 12447 process that there is room on the queue for sending (or
 12448 writing) normal data downstream.

12449 **S_WRNORM** Same as **S_OUTPUT**.

12450 **S_WRBAND** The write queue for a non-zero priority band just below the
 12451 STREAM head is no longer full.

12452		S_MSG	A STREAMS signal message that contains the SIGPOLL signal reaches the front of the STREAM head read queue.
12453			
12454		S_ERROR	Notification of an error condition reaches the STREAM head.
12455			
12456		S_HANGUP	Notification of a hangup reaches the STREAM head.
12457		S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.
12458			
12459			
12460	I_GETSIG		Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal.
12461			
12462	I_FIND		Compares the names of all modules currently present in the STREAM to the name pointed to by <i>arg</i> .
12463			
12464	I_PEEK		Allows a process to retrieve the information in the first message on the STREAM head read queue without taking the message off the queue. At least the following macros are defined for use as the <i>arg</i> argument:
12465			
12466			
12467		RS_HIPRI	Only look for high-priority messages.
12468	I_SRDOPT		Sets the read mode. At least the following macros shall be defined for use as the <i>arg</i> argument:
12469			
12470		RNORM	Byte-STREAM mode, the default.
12471		RMSGD	Message-discard mode.
12472		RMSGN	Message-nondiscard mode.
12473		RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.
12474			
12475		RPROTDAT	Deliver the control part of a message as data when a process issues a <i>read()</i> .
12476			
12477		RPROTDIS	Discard the control part of a message, delivering any data part, when a process issues a <i>read()</i> .
12478			
12479	I_GRDOPT		Returns the current read mode setting.
12480	I_NREAD		Counts the number of data bytes in data blocks in the first message on the STREAM head read queue.
12481			
12482	I_FDINSERT		Creates a message from the specified buffer(s), adds information about another STREAM, and sends the message downstream.
12483			
12484	I_STR		Constructs an internal STREAMS <i>ioctl()</i> message and sends that message downstream.
12485			
12486	I_SWROPT		Sets the write mode. At least the following macros are defined for use as the <i>arg</i> argument:
12487			
12488		SNDZERO	Send a zero-length message downstream when a <i>write()</i> of 0 bytes occurs.
12489			
12490	I_GWROPT		Returns the current write mode setting.
12491	I_SENDFD		Requests the STREAM associated with <i>fildev</i> to send a message, containing a file pointer, to the STREAM head at the other end of a STREAMS pipe.
12492			

12493	I_RECVFD	Retrieves the file descriptor associated with the message sent by an
12494		I_SENDFD <i>ioctl()</i> over a STREAMS pipe.
12495	I_LIST	This request allows the process to list all the module names on the STREAM,
12496		up to and including the topmost driver name.
12497	I_ATMARK	This request allows the process to see if the current message on the STREAM
12498		head read queue is “marked” by some module downstream. At least the
12499		following macros are defined for use as the <i>arg</i> argument:
12500	ANYMARK	Check if the message is marked.
12501	LASTMARK	Check if the message is the last one marked on the queue.
12502	I_CKBAND	Check if the message of a given priority band exists on the STREAM head
12503		read queue.
12504	I_GETBAND	Return the priority band of the first message on the STREAM head read
12505		queue.
12506	I_CANPUT	Check if a certain band is writable.
12507	I_SETCLTIME	Allow the process to set the time the STREAM head delays when a STREAM
12508		is closing and there is data on the write queues.
12509	I_GETCLTIME	Returns the close time delay.
12510	I_LINK	Connects two STREAMs.
12511	I_UNLINK	Disconnects the two STREAMs. The header shall define at least the following
12512		value for <i>arg</i> :
12513	MUXID_ALL	Unlink all STREAMs linked to the STREAM associated with
12514		<i>files</i> .
12515	I_PLINK	Connects two STREAMs with a persistent link.
12516	I_PUNLINK	Disconnects the two STREAMs that were connected with a persistent link.
12517		The following macros shall be defined for <i>getmsg()</i> , <i>getpmsg()</i> , <i>putmsg()</i> , and <i>putpmsg()</i> :
12518	MSG_ANY	Receive any message.
12519	MSG_BAND	Receive message from specified band.
12520	MSG_HIPRI	Send/receive high-priority message.
12521	MORECTL	More control information is left in message.
12522	MOREDATA	More data is left in message.
12523		The <stropts.h> header may make visible all of the symbols from <unistd.h> .
12524		The following shall be declared as functions in the <stropts.h> header and may also be defined
12525		as macros. Function prototypes shall be provided for use with an ISO C standard compiler.
12526	int	isastream(int);
12527	int	getmsg(int, struct strbuf *restrict, struct strbuf *restrict,
12528		int *restrict);
12529	int	getpmsg(int, struct strbuf *restrict, struct strbuf *restrict,
12530		int *restrict, int *restrict);
12531	int	ioctl(int, int, ...);
12532	int	putmsg(int, const struct strbuf *, const struct strbuf *, int);
12533	int	putpmsg(int, const struct strbuf *, const struct strbuf *, int,


```
12534         int);
12535     int    fattach(int, const char *);
12536     int    fdetach(const char *);
```

12537 APPLICATION USAGE

12538 None.

12539 RATIONALE

12540 None.

12541 FUTURE DIRECTIONS

12542 None.

12543 SEE ALSO

12544 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *close()*, *fcntl()*, *getmsg()*,
12545 *ioctl()*, *open()*, *pipe()*, *read()*, *poll()*, *putmsg()*, *signal()*, *write()* the XNS, Issue 5 specification,
12546 <xti.h>

12547 CHANGE HISTORY

12548 First released in Issue 4, Version 2.

12549 Issue 5

12550 The *flags* member of the **strpeek** and **strfdinsert** structures are changed from **type long** to
12551 **t_uscalar_t**.

12552 Issue 6

12553 This header is marked as part of the XSI STREAMS Option Group.

12554 The **restrict** keyword is added to the prototypes for *getmsg()* and *getpmsg()*.

12555 **NAME**

12556 sys/ipc.h — XSI interprocess communication access structure

12557 **SYNOPSIS**

12558 XSI #include <sys/ipc.h>

12559

12560 **DESCRIPTION**

12561 The **<sys/ipc.h>** header is used by three mechanisms for XSI interprocess communication (IPC):
12562 messages, semaphores, and shared memory. All use a common structure type, **ipc_perm** to pass
12563 information used in determining permission to perform an IPC operation.

12564 The **ipc_perm** structure shall contain the following members:

12565	uid_t	uid	Owner's user ID.
12566	gid_t	gid	Owner's group ID.
12567	uid_t	cuid	Creator's user ID.
12568	gid_t	cgid	Creator's group ID.
12569	mode_t	mode	Read/write permission.

12570 The **uid_t**, **gid_t**, **mode_t**, and **key_t** types shall be defined as described in **<sys/types.h>**.

12571 Definitions shall be provided for the following constants:

12572 Mode bits:

12573	IPC_CREAT	Create entry if key does not exist.
12574	IPC_EXCL	Fail if key exists.
12575	IPC_NOWAIT	Error if request must wait.

12576 Keys:

12577 IPC_PRIVATE Private key.

12578 Control commands:

12579	IPC_RMID	Remove identifier.
12580	IPC_SET	Set options.
12581	IPC_STAT	Get options.

12582 The following shall be declared as a function and may also be defined as a macro. Function
12583 prototypes shall be provided for use with an ISO C standard compiler.

12584 key_t ftok(const char *, int);

12585 **APPLICATION USAGE**

12586 None.

12587 **RATIONALE**

12588 None.

12589 **FUTURE DIRECTIONS**

12590 None.

12591 **SEE ALSO**12592 **<sys/types.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, *ftok()*

12593 **CHANGE HISTORY**

12594 First released in Issue 2. Derived from System V Release 2.0.

12595 **Issue 4**

12596 The DESCRIPTION is corrected to say that the header “is used by three mechanisms ...”.

12597 Reference to the <sys/types.h> header is added for the definitions of **uid_t**, **gid_t**, and **mode_t**.

12598 **Issue 4, Version 2**

12599 For X/OPEN UNIX conformance, the *ftok()* function is added to the list of functions declared in
12600 this header.

12601 **NAME**

12602 sys/mman.h — memory management declarations

12603 **SYNOPSIS**

12604 #include <sys/mman.h>

12605 **DESCRIPTION**12606 The **<sys/mman.h>** header shall be supported if the implementation supports at least one of the
12607 following options:

- 12608 MF • The Memory Mapped Files option
- 12609 SHM • The Shared Memory Objects option
- 12610 ML • The Process Memory Locking option
- 12611 MPR • The Memory Protection option
- 12612 TYM • The Typed Memory Objects option
- 12613 SIO • The Synchronized Input and Output option
- 12614 ADV • The Advisory Information option
- 12615 TYM • The Typed Memory Objects option

12616 The following protection options shall be defined:

- 12617 code2 **PROT_READ** Page can be read.
- 12618 code2 **PROT_WRITE** Page can be written.
- 12619 code2 **PROT_EXEC** Page can be executed.
- 12620 code2 **PROT_NONE** Page cannot be accessed.

12621 The following *flag* options shall be defined:

- 12622 MF|SHM **MAP_SHARED** Share changes.
- 12623 MF|SHM **MAP_PRIVATE** Changes are private.
- 12624 MF|SHM **MAP_FIXED** Interpret *addr* exactly.

12625 The following flags shall be defined for *msync()*:

- 12626 MF|SIO **MS_ASYNC** Perform asynchronous writes.
- 12627 MF|SIO **MS_SYNC** Perform synchronous writes.
- 12628 MF|SIO **MS_INVALIDATE** Invalidate mappings.

12629 ML The following symbolic constants shall be defined for the *mlockall()* function:

- 12630 ML **MCL_CURRENT** Lock currently mapped pages.
- 12631 ML **MCL_FUTURE** Lock pages that become mapped.

12632 MF|SHM The symbolic constant **MAP_FAILED** shall be defined to indicate a failure from the *mmap()*
12633 function.12634 code1 Values for *advice* used by *posix_madvise()* are as follows:12635 **POSIX_MADV_NORMAL**12636 The application has no advice to give on its behavior with respect to the specified range. It
12637 is the default characteristic if no advice is given for a range of memory.

12638		POSIX_MADV_SEQUENTIAL
12639		The application expects to access the specified range sequentially from lower addresses to higher addresses.
12640		
12641		POSIX_MADV_RANDOM
12642		The application expects to access the specified range in a random order.
12643		POSIX_MADV_WILLNEED
12644		The application expects to access the specified range in the near future.
12645		POSIX_MADV_DONTNEED
12646		The application expects that it will not access the specified range in the near future.
12647		
12648	TYM	The following flags shall be defined for <i>posix_typed_mem_open()</i> :
12649		POSIX_TYPED_MEM_ALLOCATE
12650		Allocate on <i>mmap()</i> .
12651		POSIX_TYPED_MEM_ALLOCATE_CONTIG
12652		Allocate contiguously on <i>mmap()</i> .
12653		POSIX_TYPED_MEM_MAP_ALLOCATABLE Map on <i>mmap()</i> , without affecting allocatability.
12654		
12655		The mode_t , off_t , and size_t types shall be defined as described in <sys/types.h>.
12656	TYM	The <sys/mman.h> header shall define the structure posix_typed_mem_info , which includes at least the following member:
12657		
12658		<code>size_t posix_tmi_length</code> Maximum length which may be allocated
12659		from a typed memory object.
12660		
12661		The following shall be declared in <sys/mman.h> as functions and may also be defined as macros. Function prototypes shall be provided for use with an ISO C standard compiler.
12662		
12663	ML	<code>int mlock(const void *, size_t);</code>
12664		<code>int mlockall(int);</code>
12665	MF SHM	<code>void *mmap(void *, size_t, int, int, int, off_t);</code>
12666	MPR	<code>int mprotect(void *, size_t, int);</code>
12667	MF SIO	<code>int msync(void *, size_t, int);</code>
12668	ML	<code>int munlock(const void *, size_t);</code>
12669		<code>int munlockall(void);</code>
12670	MF SHM	<code>int munmap(void *, size_t);</code>
12671	ADV	<code>int posix_madvise(void *, size_t, int);</code>
12672	TYM	<code>int posix_mem_offset(const void *restrict, size_t, off_t *restrict,</code>
12673		<code>size_t *restrict, int *restrict);</code>
12674		<code>int posix_typed_mem_get_info(int, struct posix_typed_mem_info *);</code>
12675		<code>int posix_typed_mem_open(const char *, int, int);</code>
12676	SHM	<code>int shm_open(const char *, int, mode_t);</code>
12677		<code>int shm_unlink(const char *);</code>
12678		

12679 **APPLICATION USAGE**

12680 None.

12681 **RATIONALE**

12682 None.

12683 **FUTURE DIRECTIONS**

12684 None.

12685 **SEE ALSO**

12686 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *mlock()*, *mlockall()*,
12687 *mmap()*, *mprotect()*, *msync()*, *munlock()*, *munlockall()*, *munmap()*, *posix_mem_offset()*,
12688 *posix_typed_mem_get_info()*, *posix_typed_mem_open()*, *shm_open()*, *shm_unlink()*

12689 **CHANGE HISTORY**

12690 First released in Issue 4, Version 2.

12691 **Issue 5**

12692 Updated for alignment with the POSIX Realtime Extension.

12693 **Issue 6**

12694 The <sys/mman.h> header is marked as dependent on support for either the
12695 `_POSIX_MAPPED_FILES`, `_POSIX_MEMLOCK`, or `_POSIX_SHARED_MEMORY` options.

12696 The following changes are made for alignment with IEEE Std. 1003.1j-2000:

- 12697 • The TYM margin code is added to the list of margin codes for the <sys/mman.h> header line,
12698 as well as for other lines.
- 12699 • The `POSIX_TYPED_MEM_ALLOCATE`, `POSIX_TYPED_MEM_ALLOCATE_CONTIG`, and
12700 `POSIX_TYPED_MEM_MAP_ALLOCATABLE` flags are added.
- 12701 • The `posix_tmi_length` structure is added.
- 12702 • The *posix_mem_offset()*, *posix_typed_mem_get_info()*, and *posix_typed_mem_open()* functions
12703 are added.

12704 The `restrict` keyword is added to the prototype for *posix_mem_offset()*.

12705 IEEE PASC Interpretation 1003.1 #102 is applied adding the prototype for *posix_madvise()*.

12706 **NAME**

12707 sys/msg.h — XSI message queue structures

12708 **SYNOPSIS**12709 XSI

```
#include <sys/msg.h>
```

12710

12711 **DESCRIPTION**12712 The <sys/msg.h> header shall define the following constant and members of the structure
12713 **msqid_ds**.12714 The following data types shall be defined through **typedef**:12715 **msgqnum_t** Used for the number of messages in the message queue.12716 **msglen_t** Used for the number of bytes allowed in a message queue.12717 These types shall be unsigned integer types that are able to store values at least as large as a type
12718 **unsigned short**.

12719 Message operation flag:

12720 **MSG_NOERROR** No error if big message.12721 The **msqid_ds** structure shall contain the following members:

12722	struct ipc_perm	msg_perm	Operation permission structure.
12723	msgqnum_t	msg_qnum	Number of messages currently on queue.
12724	msglen_t	msg_qbytes	Maximum number of bytes allowed on queue.
12725	pid_t	msg_lspid	Process ID of last <i>msgsnd()</i> .
12726	pid_t	msg_lrpid	Process ID of last <i>msgrcv()</i> .
12727	time_t	msg_stime	Time of last <i>msgsnd()</i> .
12728	time_t	msg_rtime	Time of last <i>msgrcv()</i> .
12729	time_t	msg_ctime	Time of last change.

12730 The **pid_t**, **time_t**, **key_t**, **size_t**, and **ssize_t** types shall be defined as described in <sys/types.h>.12731 The following shall be declared as functions and may also be defined as macros. Function
12732 prototypes shall be provided for use with an ISO C standard compiler.

```
12733 int      msgctl(int, int, struct msqid_ds *);
12734 int      msgget(key_t, int);
12735 ssize_t  msgrcv(int, void *, size_t, long, int);
12736 int      msgsnd(int, const void *, size_t, int);
```

12737 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12738 **APPLICATION USAGE**

12739 None.

12740 **RATIONALE**

12741 None.

12742 **FUTURE DIRECTIONS**

12743 None.

12744 **SEE ALSO**12745 <sys/types.h>, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

12746 **CHANGE HISTORY**

12747 First released in Issue 2. Derived from System V Release 2.0.

12748 **Issue 4**

12749 The function declarations in this header are expanded to full ISO C standard prototypes.

12750 Reference to the <sys/types.h> header is added for the definitions of **pid_t**, **time_t**, **key_t**, and
12751 **size_t**.

12752 A statement is added indicating that all symbols in <sys/ipc.h> are defined when this header is
12753 included.

12754 **NAME**

12755 sys/resource.h — definitions for XSI resource operations

12756 **SYNOPSIS**

12757 xsi `#include <sys/resource.h>`

12758

12759 **DESCRIPTION**

12760 The <sys/resource.h> header shall define the following symbolic constants as possible values of
 12761 the *which* argument of *getpriority()* and *setpriority()*:

12762 **PRIO_PROCESS** Identifies the *who* argument as a process ID.

12763 **PRIO_PGRP** Identifies the *who* argument as a process group ID.

12764 **PRIO_USER** Identifies the *who* argument as a user ID.

12765 The following type shall be defined through **typedef**:

12766 **rlim_t** Unsigned integer type used for limit values.

12767 The following symbolic constants shall be defined:

12768 **RLIM_INFINITY** A value of **rlim_t** indicating no limit.

12769 **RLIM_SAVED_MAX** A value of type **rlim_t** indicating an unrepresentable saved hard
 12770 limit.

12771 **RLIM_SAVED_CUR** A value of type **rlim_t** indicating an unrepresentable saved soft limit.

12772 On implementations where all resource limits are representable in an object of type **rlim_t**,
 12773 **RLIM_SAVED_MAX** and **RLIM_SAVED_CUR** need not be distinct from **RLIM_INFINITY**.

12774 The following symbolic constants shall be defined as possible values of the *who* parameter of
 12775 *getrusage()*:

12776 **RUSAGE_SELF** Returns information about the current process.

12777 **RUSAGE_CHILDREN** Returns information about children of the current process.

12778 The <sys/resource.h> header shall define the **rlimit** structure that includes at least the following
 12779 members:

12780 `rlim_t rlim_cur` The current (soft) limit.

12781 `rlim_t rlim_max` The hard limit.

12782 The <sys/resource.h> header shall define the **rusage** structure that includes at least the following
 12783 members:

12784 `struct timeval ru_utime` User time used.

12785 `struct timeval ru_stime` System time used.

12786 The **timeval** structure shall be defined as described in <sys/time.h>.

12787 The following symbolic constants shall be defined as possible values for the *resource* argument of
 12788 *getrlimit()* and *setrlimit()*:

12789 **RLIMIT_CORE** Limit on size of core dump file.

12790 **RLIMIT_CPU** Limit on CPU time per process.

12791 **RLIMIT_DATA** Limit on data segment size.

12792 **RLIMIT_FSIZE** Limit on file size.

12793 RLIMIT_NOFILE Limit on number of open files.
12794 RLIMIT_STACK Limit on stack size.
12795 RLIMIT_AS Limit on address space size.
12796 The following are declared as functions and may also be defined as macros. Function prototypes
12797 shall be provided for use with an ISO C standard compiler.

12798 int getpriority(int, id_t);
12799 int getrlimit(int, struct rlimit *);
12800 int getrusage(int, struct rusage *);
12801 int setpriority(int, id_t, int);
12802 int setrlimit(int, const struct rlimit *);

12803 The **id_t** type shall be defined through **typedef** as described in **<sys/types.h>**.
12804 Inclusion of the **<sys/resource.h>** header may also make visible all symbols from **<sys/time.h>**.

12805 **APPLICATION USAGE**
12806 None.

12807 **RATIONALE**
12808 None.

12809 **FUTURE DIRECTIONS**
12810 None.

12811 **SEE ALSO**
12812 **<sys/time.h>**, **<sys/types.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x,
12813 *getpriority()*, *getrusage()*, *getrlimit()*

12814 **CHANGE HISTORY**
12815 First released in Issue 4, Version 2.

12816 **Issue 5**
12817 Large File System extensions are added.

12818 **NAME**

12819 sys/select.h — select types

12820 **SYNOPSIS**

12821 #include <sys/select.h>

12822 **DESCRIPTION**

12823 The <sys/select.h> header shall define the **timeval** structure that includes at least the following
 12824 members:

12825	time_t	tv_sec	Seconds.
12826	suseconds_t	tv_usec	Microseconds.

12827 The **time_t** and **suseconds_t** types shall be defined as described in <sys/types.h>.

12828 The **sigset_t** type shall be defined as described in <signal.h>.

12829 The **timespec** structure shall be defined as described in <time.h>.

12830 The <sys/select.h> header shall define the **fd_set** type as a structure that includes at least the
 12831 following member:

12832 long fds_bits[] Bit mask for open file descriptions.

12833 Each of the following may be declared as a function, or defined as a macro, or both:

12834 void FD_CLR(int *fd*, fd_set **fdset*)
 12835 Clears the bit for the file descriptor *fd* in the file descriptor set *fdset*.

12836 int FD_ISSET(int *fd*, fd_set **fdset*)
 12837 Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set by
 12838 *fdset*, and 0 otherwise.

12839 void FD_SET(int *fd*, fd_set **fdset*)
 12840 Sets the bit for the file descriptor *fd* in the file descriptor set *fdset*.

12841 void FD_ZERO(fd_set **fdset*)
 12842 Initializes the file descriptor set *fdset* to have zero bits for all file descriptors.

12843 **FD_SETSIZE**
 12844 Maximum number of file descriptors in an **fd_set** structure.

12845 If implemented as macros, these may evaluate their arguments more than once, so applications
 12846 should ensure that the arguments they supply are never expressions with side effects.

12847 The following shall be declared as functions and may also be defined as macros. Function
 12848 prototypes shall be provided for use with an ISO C standard compiler.

```
12849 int pselect(int, fd_set *, fd_set *, fd_set *, const struct timespec *,
12850             const sigset_t *);
12851 int select(int, fd_set *, fd_set *, fd_set *, struct timeval *);
```

12852 Inclusion of the <sys/select.h> header may make visible all symbols from the headers
 12853 <signal.h>, <sys/time.h>, and <time.h>.

12854 **APPLICATION USAGE**

12855 None.

12856 **RATIONALE**

12857 None.

12858 **FUTURE DIRECTIONS**

12859 None.

12860 **SEE ALSO**

12861 <signal.h>, <sys/time.h>, <sys/types.h>, <time.h>, the System Interfaces volume of
12862 IEEE Std. 1003.1-200x, *pselect()*, *select()*

12863 **CHANGE HISTORY**

12864 First released in Issue 6. Derived from IEEE Std. 1003.1g-2000.

12865 **NAME**

12866 sys/sem.h — XSI semaphore facility

12867 **SYNOPSIS**

12868 XSI #include <sys/sem.h>

12869

12870 **DESCRIPTION**

12871 The <sys/sem.h> header shall define the following constants and structures.

12872 Semaphore operation flags:

12873 SEM_UNDO Set up adjust on exit entry.

12874 Command definitions for the *semctl()* function shall be provided as follows:12875 GETNCNT Get *semncnt*.12876 GETPID Get *sempid*.12877 GETVAL Get *semval*.12878 GETALL Get all cases of *semval*.12879 GETZCNT Get *semzcnt*.12880 SETVAL Set *semval*.12881 SETALL Set all cases of *semval*.12882 The **semid_ds** structure shall contain the following members:

12883 struct ipc_perm sem_perm Operation permission structure.

12884 unsigned short sem_nsems Number of semaphores in set.

12885 time_t sem_otime Last *semop()* time.12886 time_t sem_ctime Last time changed by *semctl()*.12887 The **pid_t**, **time_t**, **key_t**, and **size_t** types shall be defined as described in <sys/types.h>.12888 A semaphore shall be represented by an anonymous structure containing the following
12889 members:

12890 unsigned short semval Semaphore value.

12891 pid_t sempid Process ID of last operation.

12892 unsigned short semncnt Number of processes waiting for *semval*
12893 to become greater than current value.12894 unsigned short semzcnt Number of processes waiting for *semval*
12895 to become 0.12896 The **sembuf** structure shall contain the following members:

12897 unsigned short sem_num Semaphore number.

12898 short sem_op Semaphore operation.

12899 short sem_flg Operation flags.

12900 The following shall be declared as functions and may also be defined as macros. Function
12901 prototypes shall be provided for use with an ISO C standard compiler.

12902 int semctl(int, int, int, ...);

12903 int semget(key_t, int, int);

12904 int semop(int, struct sembuf *, size_t);

12905 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12906 **APPLICATION USAGE**

12907 None.

12908 **RATIONALE**

12909 None.

12910 **FUTURE DIRECTIONS**

12911 None.

12912 **SEE ALSO**

12913 <sys/types.h>, *semctl()*, *semget()*, *semop()*

12914 **CHANGE HISTORY**

12915 First released in Issue 2. Derived from System V Release 2.0.

12916 **Issue 4**

12917 The function declarations in this header are expanded to full ISO C standard prototypes.

12918 Reference to the <sys/types.h> header is added for the definitions of **pid_t**, **time_t**, **key_t**, and
12919 **size_t**.

12920 A statement is added indicating that all symbols in <sys/ipc.h> are defined when this header is
12921 included.

12922 **NAME**

12923 sys/shm.h — XSI shared memory facility

12924 **SYNOPSIS**

12925 XSI `#include <sys/shm.h>`

12926

12927 **DESCRIPTION**

12928 The <sys/shm.h> header shall define the following symbolic constants:

12929 SHM_RDONLY Attach read-only (else read-write).

12930 SHM_RND Round attach address to SHMLBA.

12931 The <sys/shm.h> header shall define the following symbolic value:

12932 SHMLBA Segment low boundary address multiple.

12933 The following data types shall be defined through **typedef**:

12934 **shmatt_t** Unsigned integer used for the number of current attaches that must be able to
12935 store values at least as large as a type **unsigned short**.

12936 The **shmid_ds** structure shall contain the following members:

12937 struct ipc_perm shm_perm Operation permission structure.

12938 size_t shm_segsz Size of segment in bytes.

12939 pid_t shm_lpid Process ID of last shared memory operation.

12940 pid_t shm_cpid Process ID of creator.

12941 shmatt_t shm_nattch Number of current attaches.

12942 time_t shm_atime Time of last *shmat()*.

12943 time_t shm_dtime Time of last *shmdt()*.

12944 time_t shm_ctime Time of last change by *shmctl()*.

12945 The **pid_t**, **time_t**, **key_t**, and **size_t** types shall be defined as described in <sys/types.h>.

12946 The following shall be declared as functions and may also be defined as macros. Function
12947 prototypes shall be provided for use with an ISO C standard compiler.

12948 void *shmat(int, const void *, int);

12949 int shmctl(int, int, struct shmid_ds *);

12950 int shmdt(const void *);

12951 int shmget(key_t, size_t, int);

12952 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12953 **APPLICATION USAGE**

12954 None.

12955 **RATIONALE**

12956 None.

12957 **FUTURE DIRECTIONS**

12958 None.

12959 **SEE ALSO**

12960 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *shmat()*, *shmctl()*, *shmdt()*,

12961 *shmget()*

12962 **CHANGE HISTORY**

12963 First released in Issue 2. Derived from System V Release 2.0.

12964 **Issue 4**

12965 The function declarations in this header are expanded to full ISO C standard prototypes.

12966 Reference to the <sys/types.h> header is added for the definitions of **pid_t**, **time_t**, **key_t**, and
12967 **size_t**.

12968 A statement is added indicating that all symbols in <sys/ipc.h> are defined when this header is
12969 included.

12970 **Issue 5**

12971 The type of *shm_segsz* is changed from **int** to **size_t**.

12972 **NAME**

12973 sys/socket.h — main sockets header

12974 **SYNOPSIS**

12975 #include <sys/socket.h>

12976 **DESCRIPTION**

12977 The <sys/socket.h> header shall make available the type, **socklen_t**, which is an opaque integer
12978 type of length of at least 32 bits; see APPLICATION USAGE.

12979 The <sys/socket.h> header shall define the unsigned integer type **sa_family_t**.

12980 The <sys/socket.h> header shall define the **sockaddr** structure that includes at least the
12981 following members:

12982 sa_family_t sa_family Address family.
12983 char sa_data[] Socket address (variable-length data).

12984 The **sockaddr** structure is used to define a socket address which is used in the *bind()*, *connect()*,
12985 *getpeername()*, *getsockname()*, *recvfrom()*, and *sendto()* functions.

12986 The <sys/socket.h> header shall define the **sockaddr_storage** structure. This structure shall be:

- 12987 • Large enough to accommodate all supported protocol-specific address structures
- 12988 • Aligned at an appropriate boundary so that pointers to it can be cast as pointers to protocol-
12989 specific address structures and used to access the fields of those structures without
12990 alignment problems

12991 The **sockaddr_storage** structure shall contain at least the following members:

12992 sa_family_t ss_family

12993 When a **sockaddr_storage** structure is cast as a **sockaddr** structure, the *ss_family* field of the
12994 **sockaddr_storage** structure maps onto the *sa_family* field of the **sockaddr** structure. When a
12995 **sockaddr_storage** structure is cast as a protocol-specific address structure, the *ss_family* field
12996 maps onto a field of that structure that is of type **sa_family_t** and that identifies the protocol's
12997 address family.

12998 The <sys/socket.h> header shall define the **msghdr** structure that includes at least the following
12999 members:

13000 void *msg_name Optional address.
13001 socklen_t msg_namelen Size of address.
13002 struct iovec *msg_iov Scatter/gather array.
13003 int msg_iovlen Members in msg_iov.
13004 void *msg_control Ancillary data; see below.
13005 socklen_t msg_controllen Ancillary data buffer len.
13006 int msg_flags Flags on received message.

13007 The **msghdr** structure is used to minimize the number of directly supplied parameters to the
13008 *recvmsg()* and *sendmsg()* functions. This structure is used as a *value=result* parameter in the
13009 *recvmsg()* function and *value* only for the *sendmsg()* function.

13010 The **iovec** structure shall be defined through **typedef** as described in <sys/uio.h>.

13011 The <sys/socket.h> header shall define the **cmsghdr** structure that includes at least the following
13012 members:

13013 socklen_t cmsg_len Data byte count, including the cmsghdr.
13014 int cmsg_level Originating protocol.

13015 int *cmsg_type* Protocol-specific type.

13016 The **cmsghdr** structure is used for storage of ancillary data object information.

13017 Ancillary data consists of a sequence of pairs, each consisting of a **cmsghdr** structure followed
13018 by a data array. The data array contains the ancillary data message, and the **cmsghdr** structure
13019 contains descriptive information that allows an application to correctly parse the data.

13020 The values for *cmsg_level* shall be legal values for the *level* argument to the *getsockopt()* and
13021 *setsockopt()* functions. The system documentation shall specify the *cmsg_type* definitions for the
13022 supported protocols.

13023 Ancillary data is also possible at the socket level. The **<sys/socket.h>** header defines the
13024 following macro for use as the *cmsg_type* value when *cmsg_level* is SOL_SOCKET:

13025 SCM_RIGHTS Indicates that the data array contains the access rights to be sent or
13026 received.

13027 The **<sys/socket.h>** header defines the following macros to gain access to the data arrays in the
13028 ancillary data associated with a message header:

13029 MSG_DATA(*cmsg*)
13030 If the argument is a pointer to a **cmsghdr** structure, this macro returns an unsigned
13031 character pointer to the data array associated with the **cmsghdr** structure.

13032 MSG_NXTHDR(*mhdr, cmsg*)
13033 If the first argument is a pointer to a **msghdr** structure and the second argument is a pointer
13034 to a **cmsghdr** structure in the ancillary data pointed to by the *msg_control* field of that
13035 **msghdr** structure, this macro returns a pointer to the next **cmsghdr** structure, or a null
13036 pointer if this structure is the last **cmsghdr** in the ancillary data.

13037 MSG_FIRSTHDR(*mhdr*)
13038 If the argument is a pointer to a **msghdr** structure, this macro returns a pointer to the first
13039 **cmsghdr** structure in the ancillary data associated with this **msghdr** structure, or a null
13040 pointer if there is no ancillary data associated with the **msghdr** structure.

13041 The **<sys/socket.h>** header shall define the **linger** structure that includes at least the following
13042 members:

13043 int *l_onoff* Indicates whether linger option is enabled.
13044 int *l_linger* Linger time, in seconds.

13045 The **<sys/socket.h>** header shall define the following macros, with distinct integral values:

13046 SOCK_DGRAM Datagram socket.
13047 SOCK_STREAM Byte-stream socket.
13048 SOCK_SEQPACKET Sequenced-packet socket.

13049 The **<sys/socket.h>** header shall define the following macro for use as the *level* argument of
13050 *setsockopt()* and *getsockopt()*.

13051 SOL_SOCKET Options to be accessed at socket level, not protocol level.

13052 The **<sys/socket.h>** header shall define the following macros, with distinct integral values, for
13053 use as the *option_name* argument in *getsockopt()* or *setsockopt()* calls:

13054 SO_ACCEPTCONN Socket is accepting connections.
13055 SO_BROADCAST Transmission of broadcast messages is supported.

13056	SO_DEBUG	Debugging information is being recorded.
13057	SO_DONTROUTE	Bypass normal routing.
13058	SO_ERROR	Socket error status.
13059	SO_KEEPALIVE	Connections are kept alive with periodic messages.
13060	SO_LINGER	Socket lingers on close.
13061	SO_OOBINLINE	Out-of-band data is transmitted in line.
13062	SO_RCVBUF	Receive buffer size.
13063	SO_RCVLOWAT	Receive “low water mark”.
13064	SO_RCVTIMEO	Receive timeout.
13065	SO_REUSEADDR	Reuse of local addresses is supported.
13066	SO_SNDBUF	Send buffer size.
13067	SO_SNDLOWAT	Send “low water mark”.
13068	SO_SNDTIMEO	Send timeout.
13069	SO_TYPE	Socket type.
13070	The <sys/socket.h> header shall define the following macro as the maximum <i>backlog</i> queue	
13071	length which may be specified by the <i>backlog</i> field of the <i>listen()</i> function:	
13072	SOMAXCONN	The maximum <i>backlog</i> queue length.
13073	The <sys/socket.h> header shall define the following macros, with distinct integral values, for	
13074	use as the valid values for the <i>msg_flags</i> field in the msghdr structure, or the <i>flags</i> parameter in	
13075	<i>recvfrom()</i> , <i>recvmsg()</i> , <i>sendmsg()</i> , or <i>sendto()</i> calls:	
13076	MSG_CTRUNC	Control data truncated.
13077	MSG_DONTROUTE	Send without using routing tables.
13078	MSG_EOR	Terminates a record (if supported by the protocol).
13079	MSG_OOB	Out-of-band data.
13080	MSG_PEEK	Leave received data in queue.
13081	MSG_TRUNC	Normal data truncated.
13082	MSG_WAITALL	Wait for complete message.
13083	The <sys/socket.h> header shall define the following macros, with distinct integral values:	
13084	AF_UNIX	UNIX domain sockets.
13085	AF_UNSPEC	Unspecified .
13086	AF_INET	Internet domain sockets for use with IPv4 addresses.
13087	AF_INET6	Internet domain sockets for use with IPv6 addresses.
13088	The <sys/socket.h> header shall define the following macros, with distinct integral values:	
13089	SHUT_RD	Disables further receive operations.
13090	SHUT_WR	Disables further send operations.

13091 SHUT_RDWR Disables further send and receive operations.

13092 The following are declared as functions, and may also be defined as macros. Function prototypes
13093 shall be provided for use with an ISO C standard compiler.

```

13094       int        accept(int, struct sockaddr *restrict, socklen_t *restrict);
13095       int        bind(int, const struct sockaddr *, socklen_t);
13096       int        connect(int, const struct sockaddr *, socklen_t);
13097       int        getpeername(int, struct sockaddr *restrict, socklen_t *);
13098       int        getsockname(int, struct sockaddr *restrict, socklen_t *);
13099       int        getsockopt(int, int, int, void *restrict, socklen_t *restrict);
13100       int        listen(int, int);
13101       ssize_t    recv(int, void *, size_t, int);
13102       ssize_t    recvfrom(int, void *restrict, size_t, int,
13103                    struct sockaddr *restrict, socklen_t *restrict);
13104       ssize_t    recvmsg(int, struct msghdr *, int);
13105       ssize_t    send(int, const void *, size_t, int);
13106       ssize_t    sendmsg(int, const struct msghdr *, int);
13107       ssize_t    sendto(int, const void *, size_t, int, const struct sockaddr *,
13108                    socklen_t);
13109       int        setsockopt(int, int, int, const void *, socklen_t);
13110       int        shutdown(int, int);
13111       int        socket(int, int, int);
13112       int        socketpair(int, int, int, int);
13113            Inclusion of <sys/socket.h> may also make visible all symbols from <sys/uid.h>.
```

13114 APPLICATION USAGE

13115 To forestall portability problems, it is recommended that applications not use values larger than
13116 $2^{32} - 1$ for the **socklen_t** type.

13117 The **sockaddr_storage** structure solves the problem of declaring storage for automatic variables
13118 which is both large enough and aligned enough for storing the socket address data structure of
13119 any family. For example, code with a file descriptor and without the context of the address
13120 family can pass a pointer to a variable of this type, where a pointer to a socket address structure
13121 is expected in calls such as *getpeername()*, and determine the address family by accessing the
13122 received content after the call.

13123 An example implementation design of such a data structure would be as follows:

```

13124       /*
13125        *   Desired design of maximum size and alignment.
13126        */
13127       #define _SS_MAXSIZE 128
13128            /* Implementation-defined maximum size. */
13129       #define _SS_ALIGNSIZE (sizeof(int64_t))
13130            /* Implementation-defined desired alignment. */
13131       /*
13132        *   Definitions used for sockaddr_storage structure paddings design.
13133        */
13134       #define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
13135       #define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof(sa_family_t)+
13136                            _SS_PAD1SIZE + _SS_ALIGNSIZE))
13137       struct sockaddr_storage {
13138            sa_family_t   ss_family; /* Address family. */
```

```

13139  /*
13140  *  Following fields are implementation-defined. */
13141  */
13142  char _ss_pad1[_SS_PAD1SIZE];
13143  /* 6-byte pad; this is to make implementation-defined
13144  pad up to alignment field that follows explicit in
13145  the data structure. */
13146  int64_t _ss_align; /* Field to force desired structure
13147  storage alignment. */
13148  char _ss_pad2[_SS_PAD2SIZE];
13149  /* 112-byte pad to achieve desired size,
13150  _SS_MAXSIZE value minus size of ss_family
13151  __ss_pad1, __ss_align fields is 112. */
13152  };

```

13153 The above example illustrates a data structure which aligns on a 64-bit boundary. An
13154 implementation-defined field `_ss_align` along `_ss_pad1` is used to force a 64-bit alignment which
13155 covers proper alignment good enough for needs of `sockaddr_in6` (IPv6), `sockaddr_in` (IPv4)
13156 address data structures. The size of padding fields `_ss_pad1` depends on the chosen alignment
13157 boundary. The size of padding field `_ss_pad2` depends on the value of overall size chosen for the
13158 total size of the structure. This size and alignment are represented in the above example by
13159 implementation-defined (not required) constants `_SS_MAXSIZE` (chosen value 128) and
13160 `_SS_ALIGNMENT` (with chosen value 8). Constants `_SS_PAD1SIZE` (derived value 6) and
13161 `_SS_PAD2SIZE` (derived value 112) are also for illustration and not required. The
13162 implementation-defined definitions and structure field names above start with an underscore to
13163 denote implementation private name space. Portable code is not expected to access or reference
13164 those fields or constants.

13165 RATIONALE

13166 None.

13167 FUTURE DIRECTIONS

13168 None.

13169 SEE ALSO

13170 <sys/uio.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, `accept()`, `bind()`, `connect()`,
13171 `getpeername()`, `getsockname()`, `getsockopt()`, `listen()`, `recv()`, `recvfrom()`, `recvmsg()`, `send()`,
13172 `sendmsg()`, `sendto()`, `setsockopt()`, `shutdown()`, `socket()`, `socketpair()`

13173 CHANGE HISTORY

13174 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

13175 The **restrict** keyword is added to the prototypes for `accept()`, `getpeername()`, `getsockname()`,
13176 `getsockopt()`, and `recvfrom()`.

13177 NAME

13178 sys/stat.h — data returned by the stat() function

13179 SYNOPSIS

13180 #include <sys/stat.h>

13181 DESCRIPTION

13182 XSI The <sys/stat.h> header shall define the structure of the data returned by the functions *fstat()*,
13183 *lstat()*, and *stat()*.

13184 The **stat** structure shall contain at least the following members:

13185	dev_t	st_dev	ID of device containing file.
13186	ino_t	st_ino	File serial number.
13187	mode_t	st_mode	Mode of file (see below).
13188	nlink_t	st_nlink	Number of hard links to the file.
13189	uid_t	st_uid	User ID of file.
13190	gid_t	st_gid	Group ID of file.
13191 XSI	dev_t	st_rdev	Device ID (if file is character or block special).
13192	off_t	st_size	For regular files, the file size in bytes.
13193			For symbolic links, the length in bytes of the
13194			path name contained in the symbolic link.
13195			For other file types, the use of this field is
13196			unspecified
13197	time_t	st_atime	Time of last access.
13198	time_t	st_mtime	Time of last data modification.
13199	time_t	st_ctime	Time of last status change.
13200 XSI	blksize_t	st_blksize	A file system-specific preferred I/O block size for
13201			this object. In some file system types, this may
13202			vary from file to file.
13203	blkcnt_t	st_blocks	Number of blocks allocated for this object.
13204			

13205 File serial number and device ID taken together uniquely identify the file within the system. The
13206 **blkcnt_t**, **blksize_t**, **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types shall be
13207 defined as described in <sys/types.h>. Times shall be given in seconds since the Epoch.

13208 Unless otherwise specified, the structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atime*,
13209 *st_ctime*, and *st_mtime* shall have meaningful values for all file types defined in
13210 IEEE Std. 1003.1-200x.

13211 For symbolic links, the *st_mode* member shall contain meaningful information, which can be
13212 used with the file type macros described below, that take a *mode* argument. The *st_size* member
13213 shall contain the length, in bytes, of the path name contained in the symbolic link. File mode bits
13214 and the contents of the remaining members of the **stat** structure are unspecified. The value
13215 returned in the *st_size* field shall be the length of the contents of the symbolic link, and shall not
13216 count a trailing null if one is present.

13217 The following symbolic names for the values of type *mode_t* shall also be defined.

13218 File type:

13219 XSI	S_IFMT	Type of file.
13220	S_IFBLK	Block special.
13221	S_IFCHR	Character special.

13222	S_IFIFO	FIFO special.
13223	S_IFREG	Regular.
13224	S_IFDIR	Directory.
13225	S_IFLNK	Symbolic link.
13226	S_IFSOCK	Socket.
13227	File mode bits:	
13228	S_IRWXU	Read, write, execute/search by owner.
13229	S_IRUSR	Read permission, owner.
13230	S_IWUSR	Write permission, owner.
13231	S_IXUSR	Execute/search permission, owner.
13232	S_IRWXG	Read, write, execute/search by group.
13233	S_IRGRP	Read permission, group.
13234	S_IWGRP	Write permission, group.
13235	S_IXGRP	Execute/search permission, group.
13236	S_IRWXO	Read, write, execute/search by others.
13237	S_IROTH	Read permission, others.
13238	S_IWOTH	Write permission, others.
13239	S_IXOTH	Execute/search permission, others.
13240	S_ISUID	Set-user-ID on execution.
13241	S_ISGID	Set-group-ID on execution.
13242 XSI	S_ISVTX	On directories, restricted deletion flag.
13243	The bits defined by S_IRUSR , S_IWUSR , S_IXUSR , S_IRGRP , S_IWGRP , S_IXGRP , S_IROTH ,	
13244 XSI	S_IWOTH , S_IXOTH , S_ISUID , S_ISGID , and S_ISVTX shall be unique.	
13245	S_IRWXU is the bitwise-inclusive OR of S_IRUSR , S_IWUSR , and S_IXUSR .	
13246	S_IRWXG is the bitwise-inclusive OR of S_IRGRP , S_IWGRP , and S_IXGRP .	
13247	S_IRWXO is the bitwise-inclusive OR of S_IROTH , S_IWOTH , and S_IXOTH .	
13248	Implementations may OR other implementation-defined bits into S_IRWXU , S_IRWXG , and	
13249	S_IRWXO , but they shall not overlap any of the other bits defined in this volume of	
13250	IEEE Std. 1003.1-200x. The <i>file permission bits</i> are defined to be those corresponding to the	
13251	bitwise-inclusive OR of S_IRWXU , S_IRWXG , and S_IRWXO .	
13252	The following macros shall be provided to test whether a file is of the specified type. The value	
13253	<i>m</i> supplied to the macros is the value of <i>st_mode</i> from a stat structure. The macro shall evaluate	
13254	to a non-zero value if the test is true; 0 if the test is false.	
13255	S_ISBLK(<i>m</i>)	Test for a block special file.
13256	S_ISCHR(<i>m</i>)	Test for a character special file.
13257	S_ISDIR(<i>m</i>)	Test for a directory.

- 13258 S_ISFIFO(*m*) Test for a pipe or FIFO special file.
- 13259 S_ISREG(*m*) Test for a regular file.
- 13260 S_ISLNK(*m*) Test for a symbolic link.
- 13261 S_ISSOCK(*m*) Test for a socket.

13262 The implementation may implement message queues, semaphores, or shared memory objects as
 13263 distinct file types. The following macros shall be provided to test whether a file is of the
 13264 specified type. The value of the *buf* argument supplied to the macros is a pointer to a **stat**
 13265 structure. The macro shall evaluate to a non-zero value if the specified object is implemented as
 13266 a distinct file type and the specified file type is contained in the **stat** structure referenced by *buf*.
 13267 Otherwise, the macro shall evaluate to zero.

- 13268 S_TYPEISMQ(*buf*) Test for a message queue.
- 13269 S_TYPEISSEM(*buf*) Test for a semaphore.
- 13270 S_TYPEISSHM(*buf*) Test for a shared memory object.

13271 TYM The implementation may implement typed memory objects as distinct file types, and the
 13272 following macro shall test whether a file is of the specified type. The value of the *buf* argument
 13273 supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a non-zero
 13274 value if the specified object is implemented as a distinct file type and the specified file type is
 13275 contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to zero.

- 13276 S_TYPEISTMO(*buf*) Test macro for a typed memory object.

13278 The following shall be declared as functions and may also be defined as macros. Function
 13279 prototypes shall be provided for use with an ISO C standard compiler.

```

13280 int    chmod(const char *, mode_t);
13281 int    fchmod(int, mode_t);
13282 int    fstat(int, struct stat *);
13283 int    isfdtype(int, int);
13284 int    lstat(const char *restrict, struct stat *restrict);
13285 int    mkdir(const char *, mode_t);
13286 int    mkfifo(const char *, mode_t);
13287 XSI int    mknod(const char *, mode_t, dev_t);
13288 int    stat(const char *restrict, struct stat *restrict);
13289 mode_t umask(mode_t);
  
```

13290 **APPLICATION USAGE**

13291 Use of the macros is recommended for determining the type of a file.

13292 **RATIONALE**

13293 A conforming C-language application must include <sys/stat.h> for functions that have
 13294 arguments or return values of type **mode_t**, so that symbolic values for that type can be used.
 13295 An alternative would be to require that these constants are also defined by including
 13296 <sys/types.h>.

13297 The S_ISUID and S_ISGID bits may be cleared on any write, not just on *open()*, as some historical
 13298 implementations do it.

13299 System calls that update the time entry fields in the **stat** structure must be documented by the
 13300 implementors. POSIX-conforming systems should not update the time entry fields for functions
 13301 listed in the System Interfaces volume of IEEE Std. 1003.1-200x unless the standard requires that
 13302 they do, except in the case of documented extensions to the standard.

- 13303 Note that *st_dev* must be unique within a Local Area Network (LAN) in a “system” made up of
13304 multiple computers’ file systems connected by a LAN.
- 13305 Networked implementations of a POSIX-conforming system must guarantee that all files visible
13306 within the file tree (including parts of the tree that may be remotely mounted from other
13307 machines on the network) on each individual processor are uniquely identified by the
13308 combination of the *st_ino* and *st_dev* fields.
- 13309 **FUTURE DIRECTIONS**
- 13310 None.
- 13311 **SEE ALSO**
- 13312 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *chmod()*, *fchmod()*, *fstat()*,
13313 *lstat()*, *mkdir()*, *mkfifo()*, *mknod()*, *stat()*, *umask()*
- 13314 **CHANGE HISTORY**
- 13315 First released in Issue 1. Derived from Issue 1 of the SVID.
- 13316 **Issue 4**
- 13317 Reference to the <sys/types.h> header is added for the definitions of **dev_t**, **ino_t**, **mode_t**,
13318 **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t**. This has been marked as an extension.
- 13319 References to the S_IREAD, S_IWRITE, S_IEXEC file, and S_ISVTX modes are removed.
- 13320 The descriptions of the members of the **stat** structure in the DESCRIPTION are corrected.
- 13321 The following changes are incorporated for alignment with the ISO POSIX-1 standard:
- 13322 • The function declarations in this header are expanded to full ISO C standard prototypes.
 - 13323 • The DESCRIPTION is expanded to include:
 - 13324 — How files are uniquely identified within the system
 - 13325 — Times are given in units of seconds since the Epoch
 - 13326 — Rules governing the definition and use of the file mode bits
 - 13327 — Usage of the file type test macros
- 13328 **Issue 4, Version 2**
- 13329 The following changes are incorporated for X/OPEN UNIX conformance:
- 13330 • The *st_blksize* and *st_blocks* members are added to the **stat** structure.
 - 13331 • The S_IFLINK value of S_IFMT is defined.
 - 13332 • The S_ISVTX file mode bit and the S_ISLNK file type test macro is defined.
 - 13333 • The *fchmod()*, *lstat()*, and *mknod()* functions are added to the list of functions declared in this
13334 header.
- 13335 **Issue 5**
- 13336 The DESCRIPTION is updated for alignment with POSIX Realtime Extension.
- 13337 The type of *st_blksize* is changed from **long** to **blksize_t**; the type of *st_blocks* is changed from
13338 **long** to **blkcnt_t**.
- 13339 **Issue 6**
- 13340 The S_TYPEISMQ(), S_TYPEISSEM(), and S_TYPEISSHM() macros are unconditionally
13341 mandated.
- 13342 The Open Group corrigenda item U035/4 has been applied. In the DESCRIPTION, the types
13343 **blksize_t** and **blkcnt_t** have been described.

13344 The following new requirements on POSIX implementations derive from alignment with the
13345 Single UNIX Specification:

- 13346 • The **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types are mandated.

13347 The *isfdtype()* function, S_IFSOCK, and S_ISSOCK are added for sockets.

13348 The description of **stat** structure members is changed to reflect contents when file type is a
13349 symbolic link.

13350 The test macro S_TYPEISTMO is added for alignment with IEEE Std. 1003.1j-2000. |

13351 The **restrict** keyword is added to the prototypes for *lstat()* and *stat()*. |

13352 **NAME**

13353 sys/statvfs.h — VFS File System information structure

13354 **SYNOPSIS**

13355 XSI #include <sys/statvfs.h>

13356

13357 **DESCRIPTION**

13358 The <sys/statvfs.h> header shall define the **statvfs** structure that includes at least the following
 13359 members:

13360	unsigned long	f_bsize	File system block size.
13361	unsigned long	f_frsize	Fundamental file system block size.
13362	fsblkcnt_t	f_blocks	Total number of blocks on file system in units of <i>f_frsize</i> .
13363	fsblkcnt_t	f_bfree	Total number of free blocks.
13364	fsblkcnt_t	f_bavail	Number of free blocks available to non-privileged process.
13365			
13366	fsfilcnt_t	f_files	Total number of file serial numbers.
13367	fsfilcnt_t	f_ffree	Total number of free file serial numbers.
13368	fsfilcnt_t	f_favail	Number of file serial numbers available to non-privileged process.
13369			
13370	unsigned long	f_fsid	File system ID.
13371	unsigned long	f_flag	Bit mask of <i>f_flag</i> values.
13372	unsigned long	f_namemax	Maximum file name length.

13373 The **fsblkcnt_t** and **fsfilcnt_t** types shall be defined as described in <sys/types.h>.

13374 The following flags for the *f_flag* member shall be defined:

13375	ST_RDONLY	Read-only file system.
13376	ST_NOSUID	Does not support setuid/setgid semantics.

13377 The <sys/statvfs.h> header shall declare the following functions which may also be defined as
 13378 macros. Function prototypes shall be provided for use with an ISO C standard compiler.

```
13379 int statvfs(const char *restrict, struct statvfs *restrict);
13380 int fstatvfs(int, struct statvfs *);
```

13381 **APPLICATION USAGE**

13382 None.

13383 **RATIONALE**

13384 None.

13385 **FUTURE DIRECTIONS**

13386 None.

13387 **SEE ALSO**

13388 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *fstatvfs()*, *statvfs()*

13389 **CHANGE HISTORY**

13390 First released in Issue 4, Version 2.

13391 **Issue 5**

13392 The type of *f_blocks*, *f_bfree*, and *f_bavail* is changed from **unsigned long** to **fsblkcnt_t**; the type
 13393 of *f_files*, *f_ffree*, and *f_favail* is changed from **unsigned long** to **fsfilcnt_t**.

13394 **Issue 6**

13395 The Open Group corrigenda item U035/5 has been applied. In the DESCRIPTION, the types
13396 **fsblkcnt_t** and **fsfilcnt_t** have been described.

13397 The **restrict** keyword is added to the prototype for *statvfs()*.

13398 **NAME**

13399 sys/time.h — time types

13400 **SYNOPSIS**

13401 XSI `#include <sys/time.h>`

13402

13403 **DESCRIPTION**

13404 The <sys/time.h> header shall define the **timeval** structure that includes at least the following
 13405 members:

13406 `time_t` `tv_sec` Seconds.

13407 `suseconds_t` `tv_usec` Microseconds.

13408 The <sys/time.h> header shall define the **itimerval** structure that includes at least the following
 13409 members:

13410 `struct timeval it_interval` Timer interval.

13411 `struct timeval it_value` Current value.

13412 The **time_t** and **suseconds_t** types shall be defined as described in <sys/types.h>.

13413 The <sys/time.h> header shall define the **fd_set** type as a structure that includes at least the
 13414 following member:

13415 `long fds_bits[]` Bit mask for open file descriptions.

13416 The <sys/time.h> header shall define the following values for the *which* argument of *getitimer()*
 13417 and *setitimer()*:

13418 **ITIMER_REAL** Decrements in real time.

13419 **ITIMER_VIRTUAL** Decrements in process virtual time.

13420 **ITIMER_PROF** Decrements both in process virtual time and when the system is running
 13421 on behalf of the process.

13422 Each of the following may be declared as a function, or defined as a macro, or both:

13423 `void FD_CLR(int fd, fd_set *fdset)`

13424 Clears the bit for the file descriptor *fd* in the file descriptor set *fdset*.

13425 `int FD_ISSET(int fd, fd_set *fdset)`

13426 Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set by
 13427 *fdset*, and 0 otherwise.

13428 `void FD_SET(int fd, fd_set *fdset)`

13429 Sets the bit for the file descriptor *fd* in the file descriptor set *fdset*.

13430 `void FD_ZERO(fd_set *fdset)`

13431 Initializes the file descriptor set *fdset* to have zero bits for all file descriptors.

13432 **FD_SETSIZE**

13433 Maximum number of file descriptors in an **fd_set** structure.

13434 If implemented as macros, these may evaluate their arguments more than once, so that
 13435 arguments must never be expressions with side effects.

13436 The following shall be declared as functions and may also be defined as macros. Function
 13437 prototypes shall be provided for use with an ISO C standard compiler.

13438 `int getitimer(int, struct itimerval *);`

13439 `int gettimeofday(struct timeval *, void *);`

13440 int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
13441 struct timeval *restrict);
13442 int setitimer(int, const struct itimerval *restrict,
13443 struct itimerval *restrict);
13444 int utimes(const char *, const struct timeval [2]); (**LEGACY**)

13445 Inclusion of the <sys/time.h> header may make visible all symbols from the <sys/select.h>
13446 header.

13447 **APPLICATION USAGE**

13448 None.

13449 **RATIONALE**

13450 None.

13451 **FUTURE DIRECTIONS**

13452 None.

13453 **SEE ALSO**

13454 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *getitimer()*, *gettimeofday()*,
13455 *select()*, *setitimer()*

13456 **CHANGE HISTORY**

13457 First released in Issue 4, Version 2.

13458 **Issue 5**

13459 The type of *tv_usec* is changed from **long** to **suseconds_t**.

13460 **Issue 6**

13461 The **restrict** keyword is added to the prototypes for *select()* and *setitimer()*.

13462 The note is added that inclusion of this header may also make symbols visible from
13463 <sys/socket.h>.

13464 **NAME**

13465 sys/timeb.h — additional definitions for date and time

13466 **SYNOPSIS**

13467 XSI #include <sys/timeb.h>

13468

13469 **DESCRIPTION**13470 The <sys/timeb.h> header shall define the **timeb** structure that includes at least the following
13471 members:

13472	time_t	time	The seconds portion of the current time.
13473	unsigned short	millitm	The milliseconds portion of the current time.
13474	short	timezone	The local timezone in minutes west of Greenwich.
13475	short	dstflag	TRUE if Daylight Savings Time is in effect.

13476 The **time_t** type shall be defined as described in <sys/types.h>.13477 The <sys/timeb.h> header shall declare the following as a function which may also be defined as
13478 a macro. Function prototypes shall be provided for use with an ISO C standard compiler.13479 int ftime(struct timeb *); (**LEGACY**)13480 **APPLICATION USAGE**

13481 None.

13482 **RATIONALE**

13483 None.

13484 **FUTURE DIRECTIONS**

13485 None.

13486 **SEE ALSO**

13487 <sys/types.h>, <time.h>

13488 **CHANGE HISTORY**

13489 First released in Issue 4, Version 2.

13490 **Issue 6**13491 The *ftime()* function is marked LEGACY.

13492 **NAME**13493 `sys/times.h` — file access and modification times structure13494 **SYNOPSIS**13495 `#include <sys/times.h>`13496 **DESCRIPTION**13497 The **<sys/times.h>** header shall define the structure **tms**, which is returned by *times()* and
13498 includes at least the following members:13499 `clock_t tms_utime` User CPU time.13500 `clock_t tms_stime` System CPU time.13501 `clock_t tms_cutime` User CPU time of terminated child processes.13502 `clock_t tms_cstime` System CPU time of terminated child processes.13503 The **clock_t** type shall be defined as described in **<sys/types.h>**.13504 The following shall be declared as a function and may also be defined as a macro. Function
13505 prototypes shall be provided for use with an ISO C standard compiler.13506 `clock_t times(struct tms *);`13507 **APPLICATION USAGE**

13508 None.

13509 **RATIONALE**

13510 None.

13511 **FUTURE DIRECTIONS**

13512 None.

13513 **SEE ALSO**13514 **<sys/types.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, *times()*13515 **CHANGE HISTORY**

13516 First released in Issue 1. Derived from Issue 1 of the SVID.

13517 **Issue 4**13518 Reference to the **<sys/types.h>** header is added for the definitions of **clock_t**.13519 This issue states that the *times()* function can also be defined as a macro.

13520 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 13521
- The function declarations in this header are expanded to full ISO C standard prototypes.

13522 **NAME**

13523 sys/types.h — data types

13524 **SYNOPSIS**

13525 #include <sys/types.h>

13526 **DESCRIPTION**

13527 The <sys/types.h> header shall include definitions for at least the following types:

13528	blkcnt_t	Used for file block counts.
13529	blksize_t	Used for block sizes.
13530 XSI 13531	clock_t	Used for system times in clock ticks or CLOCKS_PER_SEC; see <time.h>.
13532 TMR	clockid_t	Used for clock ID type in the clock and timer functions.
13533	dev_t	Used for device IDs.
13534 XSI	fsblkcnt_t	Used for file system block counts.
13535 XSI	fsfilcnt_t	Used for file system file counts.
13536	gid_t	Used for group IDs.
13537 XSI 13538	id_t	Used as a general identifier; can be used to contain at least a pid_t , uid_t , or gid_t .
13539	ino_t	Used for file serial numbers.
13540 XSI	key_t	Used for XSI interprocess communication.
13541	mode_t	Used for some file attributes.
13542	nlink_t	Used for link counts.
13543	off_t	Used for file sizes.
13544	pid_t	Used for process IDs and process group IDs.
13545 THR	pthread_attr_t	Used to identify a thread attribute object.
13546 BAR	pthread_barrier_t	Used to identify a barrier.
13547 BAR	pthread_barrierattr_t	Used to define a barrier attributes object.
13548 THR	pthread_cond_t	Used for condition variables.
13549 THR	pthread_condattr_t	Used to identify a condition attribute object.
13550 THR	pthread_key_t	Used for thread-specific data keys.
13551 THR	pthread_mutex_t	Used for mutexes.
13552 THR	pthread_mutexattr_t	Used to identify a mutex attribute object.
13553 THR	pthread_once_t	Used for dynamic package initialization.
13554 THR	pthread_rwlock_t	Used for read-write locks.
13555 THR	pthread_rwlockattr_t	Used for read-write lock attributes.
13556 SPI	pthread_spinlock_t	Used to identify a spin lock.
13557 THR	pthread_t	Used to identify a thread.

13558	size_t	Used for sizes of objects.
13559	ssize_t	Used for a count of bytes or an error indication.
13560 XSI	suseconds_t	Used for time in microseconds
13561	time_t	Used for time in seconds.
13562 TMR	timer_t	Used for timer ID returned by <i>timer_create()</i> .
13563	uid_t	Used for user IDs.
13564 XSI	useconds_t	Used for time in microseconds.

13565 All of the types shall be defined as arithmetic types of an appropriate length, with the following
 13566 exceptions:

13567 XSI	key_t
13568 THR	pthread_attr_t
13569 BAR	pthread_barrier_t
13570	pthread_barrierattr_t
13571 THR	pthread_cond_t
13572	pthread_condattr_t
13573	pthread_key_t
13574	pthread_mutex_t
13575	pthread_mutexattr_t
13576	pthread_once_t
13577	pthread_rwlock_t
13578	pthread_rwlockattr_t
13579 SPI	pthread_spinlock_t
13580 TRC	trace_attr_t
13581	trace_event_id_t
13582 TRC TEF	trace_event_set_t
13583 TRC	trace_id_t
13584	

13585 Additionally:

- 13586 • **blkcnt_t** and **off_t** shall be signed integer types.
- 13587 XSI • **fsblkcnt_t**, **fsfilcnt_t**, and **ino_t** shall be defined as unsigned integer types.
- 13588 • **size_t** shall be an unsigned integer type.
- 13589 • **blksize_t**, **pid_t**, and **ssize_t** shall be signed integer types.

13590 XSI The type **ssize_t** shall be capable of storing values at least in the range $[-1, \{SSIZE_MAX\}]$. The
 13591 type **useconds_t** shall be an unsigned integer type capable of storing values at least in the range
 13592 $[0, 1\,000\,000]$. The type **suseconds_t** shall be a signed integer type capable of storing values at
 13593 least in the range $[-1, 1\,000\,000]$.

13594 There are no defined comparison or assignment operators for the following types:

13595 THR	pthread_attr_t
13596 BAR	pthread_barrier_t
13597	pthread_barrierattr_t
13598 THR	pthread_cond_t
13599	pthread_condattr_t
13600	pthread_mutex_t
13601	pthread_mutexattr_t

13602 **pthread_rwlock_t**
 13603 **pthread_rwlockattr_t**
 13604 SPI **pthread_spinlock_t**
 13605 TRC **trace_attr_t**
 13606

13607 **APPLICATION USAGE**

13608 None.

13609 **RATIONALE**

13610 None.

13611 **FUTURE DIRECTIONS**

13612 None.

13613 **SEE ALSO**

13614 <time.h>

13615 **CHANGE HISTORY**

13616 First released in Issue 1. Derived from Issue 1 of the SVID.

13617 **Issue 4**

13618 The **clock_t** type is marked as an extension.

13619 In the last paragraph of the DESCRIPTION, only the reference to type **key_t** is now marked as
 13620 an extension.

13621 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 13622 • The data type **ssize_t** is added.
- 13623 • The DESCRIPTION is expanded to indicate the required arithmetic types.

13624 **Issue 4, Version 2**

13625 The **id_t** and **useconds_t** types are defined for X/OPEN UNIX conformance. The capability of
 13626 the **useconds_t** type is described.

13627 **Issue 5**

13628 The **clockid_t** and **timer_t** types are defined for alignment with the POSIX Realtime Extension.

13629 The types **blkcnt_t**, **blksize_t**, **fsblkcnt_t**, **fsfilcnt_t**, and **suseconds_t** are added.

13630 Large File System extensions are added.

13631 Updated for alignment with the POSIX Threads Extension.

13632 **Issue 6**

13633 The **pthread_barrier_t**, **pthread_barrierattr_t**, and **pthread_spinlock_t** types are added for
 13634 alignment with IEEE Std. 1003.1j-2000.

13635 The margin code is changed from XSI to THR for the **pthread_rwlock_t** and
 13636 **pthread_rwlockattr_t** types as Read-Write Locks have been absorbed into the POSIX Threads
 13637 option. The threads types are now marked THR.

13638 **NAME**13639 `sys/uio.h` — definitions for vector I/O operations13640 **SYNOPSIS**13641 XSI `#include <sys/uio.h>`

13642

13643 **DESCRIPTION**13644 The `<sys/uio.h>` header shall define the `iovec` structure that includes at least the following
13645 members:13646 `void *iov_base` Base address of a memory region for input or output.13647 `size_t iov_len` The size of the memory pointed to by `iov_base`.13648 The `<sys/uio.h>` header uses the `iovec` structure for scatter/gather I/O.13649 The `ssize_t` and `size_t` types shall be defined as described in `<sys/types.h>`.13650 The following shall be declared as functions and may also be defined as macros. Function
13651 prototypes shall be provided for use with an ISO C standard compiler.13652 `ssize_t readv(int, const struct iovec *, int);`13653 `ssize_t writev(int, const struct iovec *, int);`13654 **APPLICATION USAGE**13655 The implementation can put a limit on the number of scatter/gather elements which can be
13656 processed in one call. The symbol `{IOV_MAX}` defined in `<limits.h>` should always be used to
13657 learn about the limits instead of assuming a fixed value.13658 **RATIONALE**13659 Traditionally, the maximum number of scatter/gather elements the system can process in one
13660 call were described by the symbolic value `{UIO_MAXIOV}`. In IEEE Std. 1003.1-200x this value
13661 was replaced by the constant `{IOV_MAX}` which can be found in `<limits.h>`.13662 **FUTURE DIRECTIONS**

13663 None.

13664 **SEE ALSO**13665 `<limits.h>`, `<sys/types.h>`, the System Interfaces volume of IEEE Std. 1003.1-200x, `read()`, `write()`13666 **CHANGE HISTORY**

13667 First released in Issue 4, Version 2.

13668 **Issue 6**

13669 Text referring to scatter/gather I/O is added to the DESCRIPTION.

13670 **NAME**

13671 sys/un.h — definitions for UNIX domain sockets

13672 **SYNOPSIS**

13673 #include <sys/un.h>

13674 **DESCRIPTION**

13675 The <sys/un.h> header shall define the **sockaddr_un** structure that includes at least the
13676 following members:

13677 sa_family_t sun_family Address family.
13678 char sun_path[] Socket path name.

13679 The **sockaddr_un** structure is used to store addresses for UNIX domain sockets. Values of this
13680 type shall be cast by applications to **struct sockaddr** for use with socket functions.

13681 The **sa_family_t** type shall be defined as described in <sys/socket.h>.

13682 **APPLICATION USAGE**

13683 The size of *sun_path* has intentionally been left undefined. This is because different
13684 implementations use different sizes. For example, BSD4.3 uses a size of 108, and BSD4.4 uses a
13685 size of 104. Since most implementations originate from BSD versions, the size is typically in the
13686 range 92 to 108.

13687 Applications should not assume a particular length for *sun_path* or assume that it can hold
13688 `_POSIX_PATH_MAX` characters (255).

13689 **RATIONALE**

13690 None.

13691 **FUTURE DIRECTIONS**

13692 None.

13693 **SEE ALSO**

13694 <sys/socket.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *bind()*, *socket()*,
13695 *socketpair()*

13696 **CHANGE HISTORY**

13697 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

13698 **NAME**

13699 sys/utsname.h — system name structure

13700 **SYNOPSIS**

13701 #include <sys/utsname.h>

13702 **DESCRIPTION**13703 The **<sys/utsname.h>** header shall define the structure **utsname** which shall include at least the
13704 following members:

13705 char sysname[] Name of this implementation of the operating system.
13706 char nodename[] Name of this node within an implementation-defined
13707 communications network.
13708 char release[] Current release level of this implementation.
13709 char version[] Current version level of this release.
13710 char machine[] Name of the hardware type on which the system is running.

13711 The character arrays are of unspecified size, but the data stored in them shall be terminated by a
13712 null byte.

13713 The following shall be declared as a function and may also be defined as a macro:

13714 int uname(struct utsname *);

13715 **APPLICATION USAGE**

13716 None.

13717 **RATIONALE**

13718 None.

13719 **FUTURE DIRECTIONS**

13720 None.

13721 **SEE ALSO**13722 The System Interfaces volume of IEEE Std. 1003.1-200x, *uname()*13723 **CHANGE HISTORY**

13724 First released in Issue 1. Derived from Issue 1 of the SVID.

13725 **Issue 4**

13726 The word “character” is replaced with the word “byte” in the DESCRIPTION.

13727 The function in this header can now also be defined as a macro.

13728 The following change is incorporated for alignment with the ISO C standard:

- 13729
- The function declarations in this header are expanded to full ISO C standard prototypes.

13730 **NAME**

13731 sys/wait.h — declarations for waiting

13732 **SYNOPSIS**

13733 #include <sys/wait.h>

13734 **DESCRIPTION**

13735 The <sys/wait.h> header shall define the following symbolic constants for use with *waitpid()*:

13736 WNOHANG Do not hang if no status is available; return immediately.

13737 WUNTRACED Report status of stopped child process.

13738 The <sys/wait.h> header shall define the following macros for analysis of process status values:

13739 WEXITSTATUS Return exit status.

13740 XSI WIFCONTINUED True if child has been continued

13741 WIFEXITED True if child exited normally.

13742 WIFSIGNALED True if child exited due to uncaught signal.

13743 WIFSTOPPED True if child is currently stopped.

13744 WSTOPSIG Return signal number that caused process to stop.

13745 WTERMSIG Return signal number that caused process to terminate.

13746 XSI The following symbolic constants shall be defined as possible values for the *options* argument to
13747 *waitid()*:

13748 WEXITED Wait for processes that have exited.

13749 WSTOPPED Status is returned for any child that has stopped upon receipt of a signal.

13750 WCONTINUED Status is returned for any child that was stopped and has been continued.

13751 WNOHANG Return immediately if there are no children to wait for.

13752 WNOWAIT Keep the process whose status is returned in *infp* in a waitable state.

13753 The type **idtype_t** shall be defined as an enumeration type whose possible values shall include
13754 at least the following:

13755 P_ALL

13756 P_PID

13757 P_PGID

13758

13759 The **id_t** and **pid_t** types shall be defined as described in <sys/types.h>.

13760 XSI The **siginfo_t** type shall be defined as described in <signal.h>.

13761 The **rusage** structure shall be defined as described in <sys/resource.h>.

13762 Inclusion of the <sys/wait.h> header may also make visible all symbols from <signal.h> and
13763 <sys/resource.h>.

13764 The following shall be declared as functions and may also be defined as macros. Function
13765 prototypes shall be provided for use with an ISO C standard compiler.

13766 pid_t wait(int *);

13767 XSI int waitid(idtype_t, id_t, siginfo_t *, int);

13768 pid_t waitpid(pid_t, int *, int);

13769 **APPLICATION USAGE**

13770 None.

13771 **RATIONALE**

13772 None.

13773 **FUTURE DIRECTIONS**

13774 None.

13775 **SEE ALSO**

13776 <signal.h>, <sys/resource.h>, <sys/types.h>, <sys/wait.h>, the System Interfaces volume of
13777 IEEE Std. 1003.1-200x, *wait()*, *waitid()*

13778 **CHANGE HISTORY**

13779 First released in Issue 3.

13780 Entry included for alignment with the POSIX.1-1988 standard.

13781 **Issue 4**

13782 Reference to the <sys/types.h> header is added for the definition of **pid_t** and marked as an
13783 extension.

13784 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 13785 • The function declarations in this header are expanded to full ISO C standard prototypes.

13786 **Issue 4, Version 2**

13787 The following changes are incorporated for X/OPEN UNIX conformance:

- 13788 • The WIFCONTINUED macro, the list of symbolic constants for the *options* argument to
13789 *waitid()*, and the description of the **idtype_t** enumeration type are added.
- 13790 • A statement is added indicated that inclusion of this header may also make visible constants
13791 from <signal.h> and <sys/resource.h>.
- 13792 • The *wait3()* and *waitid()* functions are added to the list of functions declared in this header.

13793 **Issue 6**

13794 The *wait3()* function is removed.

13795 **NAME**

13796 syslog — definitions for system error logging

13797 **SYNOPSIS**

13798 xSI #include <syslog.h>

13799

13800 **DESCRIPTION**

13801 The <syslog.h> header shall define the following symbolic constants, zero or more of which may
 13802 be OR'ed together to form the *logopt* option of *openlog()*:

13803 LOG_PID Log the process ID with each message.

13804 LOG_CONS Log to the system console on error.

13805 LOG_NDELAY Connect to syslog daemon immediately.

13806 LOG_ODELAY Delay open until *syslog()* is called.

13807 LOG_NOWAIT Do not wait for child processes.

13808 The following symbolic constants shall be defined as possible values of the *facility* argument to
 13809 *openlog()*:

13810 LOG_KERN Reserved for message generated by the system.

13811 LOG_USER Message generated by a process.

13812 LOG_MAIL Reserved for message generated by mail system.

13813 LOG_NEWS Reserved for message generated by news system.

13814 LOG_UUCP Reserved for message generated by UUCP system.

13815 LOG_DAEMON Reserved for message generated by system daemon.

13816 LOG_AUTH Reserved for message generated by authorization daemon.

13817 LOG_CRON Reserved for message generated by the clock daemon.

13818 LOG_LPR Reserved for message generated by printer system.

13819 LOG_LOCAL0 Reserved for local use.

13820 LOG_LOCAL1 Reserved for local use.

13821 LOG_LOCAL2 Reserved for local use.

13822 LOG_LOCAL3 Reserved for local use.

13823 LOG_LOCAL4 Reserved for local use.

13824 LOG_LOCAL5 Reserved for local use.

13825 LOG_LOCAL6 Reserved for local use.

13826 LOG_LOCAL7 Reserved for local use.

13827 The following shall be declared as macros for constructing the *maskpri* argument to *setlogmask()*.
 13828 The following macros expand to an expression of type **int** when the argument *pri* is an
 13829 expression of type **int**:

13830 LOG_MASK(*pri*) A mask for priority *pri*.

13831 The following constants shall be defined as possible values for the *priority* argument of *syslog()*:

13832 LOG_EMERG A panic condition was reported to all processes.

13833 LOG_ALERT A condition that should be corrected immediately.

13834 LOG_CRIT A critical condition.

13835 LOG_ERR An error message.

13836 LOG_WARNING A warning message.

13837 LOG_NOTICE A condition requiring special handling.

13838 LOG_INFO A general information message.

13839 LOG_DEBUG A message useful for debugging programs.

13840 The following shall be declared as functions and may also be defined as macros. Function

13841 prototypes shall be provided for use with an ISO C standard compiler.

13842 void closelog(void);

13843 void openlog(const char *, int, int);

13844 int setlogmask(int);

13845 void syslog(int, const char *, ...);

13846 **APPLICATION USAGE**

13847 None.

13848 **RATIONALE**

13849 None.

13850 **FUTURE DIRECTIONS**

13851 None.

13852 **SEE ALSO**

13853 The System Interfaces volume of IEEE Std. 1003.1-200x, *closelog()*

13854 **CHANGE HISTORY**

13855 First released in Issue 4, Version 2.

13856 **Issue 5**

13857 Moved to X/Open UNIX to BASE.

13858 **NAME**

13859 tar.h — extended tar definitions

13860 **SYNOPSIS**

13861 #include <tar.h>

13862 **DESCRIPTION**

13863 The <tar.h> header shall define header block definitions as follows.

13864 General definitions:

13865

13866

13867

13868

13869

13870

Name	Description	Value
TMAGIC	"ustar"	ustar plus null byte.
TMAGLEN	6	Length of the above.
TVERSION	"00"	00 without a null byte.
TVERSLEN	2	Length of the above.

13871

Typeflag field definitions:

13872

13873

13874

13875

13876

13877

13878

13879

13880

13881

13882

Name	Description	Value
REGTYPE	'0'	Regular file.
AREGTYPE	'\0'	Regular file.
LNKTYPE	'1'	Link.
SYMTYPE	'2'	Symbolic link.
CHRTYPE	'3'	Character special.
BLKTYPE	'4'	Block special.
DIRTYPE	'5'	Directory.
FIFOTYPE	'6'	FIFO special.
CONTTYPE	'7'	Reserved.

13883

Mode field bit definitions (octal):

13884

13885

13886

13887

13888 XSI

13889

13890

13891

13892

13893

13894

13895

13896

13897

Name	Description	Value
TSUID	04000	Set UID on execution.
TSGID	02000	Set GID on execution.
TSVTX	01000	On directories, restricted deletion flag.
TUREAD	00400	Read by owner.
TUWRITE	00200	Write by owner special.
TUEXEC	00100	Execute/search by owner.
TGREAD	00040	Read by group.
TGWRITE	00020	Write by group.
TGEXEC	00010	Execute/search by group.
TOREAD	00004	Read by other.
TOWRITE	00002	Write by other.
TOEXEC	00001	Execute/search by other.

13898 **APPLICATION USAGE**

13899 None.

13900 **RATIONALE**

13901 None.

13902 **FUTURE DIRECTIONS**

13903 None.

13904 **SEE ALSO**13905 The Shell and Utilities volume of IEEE Std. 1003.1-200x, *pax*13906 **CHANGE HISTORY**

13907 First released in Issue 3. Derived from the entry in the POSIX.1-1988 standard.

13908 **Issue 4**

13909 This entry is moved from the Headers Interface, Issue 3 specification.

13910 **Issue 4, Version 2**

13911 The following changes are incorporated for X/OPEN UNIX conformance:

13912 • The significance of SYMTYPE as the value of the *typeflag* field is explained.13913 • The value of TSVTX as the value of the *mode* field is explained.13914 **Issue 6**13915 The SEE ALSO section now refers to *pax* since the Shell and Utilities volume of
13916 IEEE Std. 1003.1-200x no longer contains the *tar* utility.

13917 **NAME**

13918 termios.h — define values for termios

13919 **SYNOPSIS**

13920 #include <termios.h>

13921 **DESCRIPTION**

13922 The <termios.h> header contains the definitions used by the terminal I/O interfaces (see
13923 Chapter 11 (on page 213) for the structures and names defined).

13924 **The termios Structure**

13925 The following data types shall be defined through **typedef**:

13926 **cc_t** Used for terminal special characters.

13927 **speed_t** Used for terminal baud rates.

13928 **tcflag_t** Used for terminal modes.

13929 The above types shall be all unsigned integer types.

13930 The **termios** structure shall be defined, and shall include at least the following members:

- 13931 tcflag_t c_iflag Input modes.
- 13932 tcflag_t c_oflag Output modes.
- 13933 tcflag_t c_cflag Control modes.
- 13934 tcflag_t c_lflag Local modes.
- 13935 cc_t c_cc[NCCS] Control characters.

13936 A definition shall be provided for:

13937 NCCS Size of the array *c_cc* for control characters.

13938 The following subscript names for the array *c_cc* shall be defined:

13939

13940

13941

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF		EOF character.
VEOL		EOL character.
VERASE		ERASE character.
VINTR	VINTR	INTR character.
VKILL		KILL character.
	VMIN	MIN value.
VQUIT	VQUIT	QUIT character.
VSTART	VSTART	START character.
VSTOP	VSTOP	STOP character.
VSUSP	VSUSP	SUSP character.
	VTIME	TIME value.

13942

13943

13944

13945

13946

13947

13948

13949

13950

13951

13952

13953 The subscript values shall be unique, except that the VMIN and VTIME subscripts may have the
13954 same values as the VEOF and VEOL subscripts, respectively.

13955 The following flags shall be provided.

13956 **Input Modes**

13957 The *c_iflag* field describes the basic terminal input control:

- 13958 BRKINT Signal interrupt on break.
- 13959 ICRNL Map CR to NL on input.
- 13960 IGNBRK Ignore break condition.
- 13961 IGNCR Ignore CR.
- 13962 IGNPAR Ignore characters with parity errors.
- 13963 INLCR Map NL to CR on input.
- 13964 INPCK Enable input parity check.
- 13965 ISTRIP Strip character.
- 13966 XSI IXANY Enable any character to restart output.
- 13967 IXOFF Enable start/stop input control.
- 13968 IXON Enable start/stop output control.
- 13969 PARMRK Mark parity errors.

13970 **Output Modes**

13971 The *c_oflag* field specifies the system treatment of output:

- 13972 OPOST Post-process output.
- 13973 XSI ONLCR Map NL to CR-NL on output.
- 13974 OCRNL Map CR to NL on output.
- 13975 ONOCR No CR output at column 0.
- 13976 ONLRET NL performs CR function.
- 13977 OFILL Use fill characters for delay.
- 13978 NLDLY Select newline delays:
- 13979 NL0 <newline> character type 0.
- 13980 NL1 <newline> character type 1.
- 13981 CRDLY Select carriage-return delays:
- 13982 CR0 Carriage-return delay type 0.
- 13983 CR1 Carriage-return delay type 1.
- 13984 CR2 Carriage-return delay type 2.
- 13985 CR3 Carriage-return delay type 3.
- 13986 TABDLY Select horizontal-tab delays:
- 13987 TAB0 Horizontal-tab delay type 0.
- 13988 TAB1 Horizontal-tab delay type 1.
- 13989 TAB2 Horizontal-tab delay type 2.

13990	TAB3	Expand tabs to spaces.
13991	BSDLY	Select backspace delays:
13992	BS0	Backspace-delay type 0.
13993	BS1	Backspace-delay type 1.
13994	VTDLY	Select vertical-tab delays:
13995	VT0	Vertical-tab delay type 0.
13996	VT1	Vertical-tab delay type 1.
13997	FFDLY	Select form-feed delays:
13998	FF0	Form-feed delay type 0.
13999	FF1	Form-feed delay type 1.

14000 **Baud Rate Selection**

14001 The input and output baud rates are stored in the **termios** structure. These are the valid values
 14002 for objects of type **speed_t**. The following values shall be defined, but not all baud rates need be
 14003 supported by the underlying hardware.

14004	B0	Hang up
14005	B50	50 baud
14006	B75	75 baud
14007	B110	110 baud
14008	B134	134.5 baud
14009	B150	150 baud
14010	B200	200 baud
14011	B300	300 baud
14012	B600	600 baud
14013	B1200	1200 baud
14014	B1800	1800 baud
14015	B2400	2400 baud
14016	B4800	4800 baud
14017	B9600	9600 baud
14018	B19200	19200 baud
14019	B38400	38400 baud

14020 Control Modes

14021 The *c_cflag* field describes the hardware control of the terminal; not all values specified are
14022 required to be supported by the underlying hardware:

14023	CSIZE	Character size:
14024		CS5 5 bits
14025		CS6 6 bits
14026		CS7 7 bits
14027		CS8 8 bits
14028	CSTOPB	Send two stop bits, else one.
14029	CREAD	Enable receiver.
14030	PARENB	Parity enable.
14031	PARODD	Odd parity, else even.
14032	HUPCL	Hang up on last close.
14033	CLOCAL	Ignore modem status lines.

14034 Local Modes

14035 The *c_lflag* field of the argument structure is used to control various terminal functions:

14036	ECHO	Enable echo.
14037	ECHOE	Echo erase character as error-correcting backspace.
14038	ECHOK	Echo KILL.
14039	ECHONL	Echo NL.
14040	ICANON	Canonical input (erase and kill processing).
14041	IEXTEN	Enable extended input character processing.
14042	ISIG	Enable signals.
14043	NOFLSH	Disable flush after interrupt or quit.
14044	TOSTOP	Send SIGTTOU for background output.

14045 Attribute Selection

14046 The following symbolic constants for use with *tcsetattr()* are defined:

14047	TCSANOW	Change attributes immediately.
14048	TCSADRAIN	Change attributes when output has drained.
14049	TCSAFLUSH	Change attributes when output has drained; also flush pending input.

14050 **Line Control**

14051 The following symbolic constants for use with *tflush()* shall be defined:

- 14052 TCIFLUSH Flush pending input. Flush untransmitted output.
- 14053 TCIOFLUSH Flush both pending input and untransmitted output.
- 14054 TCOFLUSH Flush untransmitted output.

14055 The following symbolic constants for use with *tflow()* shall be defined:

- 14056 TCIOFF Transmit a STOP character, intended to suspend input data.
- 14057 TCION Transmit a START character, intended to restart input data.
- 14058 TCOOFF Suspend output.
- 14059 TCOON Restart output.

14060 The following shall be declared as functions and may also be defined as macros. Function
14061 prototypes shall be provided for use with an ISO C standard compiler.

```

14062 speed_t cfgetispeed(const struct termios *);
14063 speed_t cfgetospeed(const struct termios *);
14064 int cfsetispeed(struct termios *, speed_t);
14065 int cfsetospeed(struct termios *, speed_t);
14066 int tcdrain(int);
14067 int tcflow(int, int);
14068 int tcf flush(int, int);
14069 int tcgetattr(int, struct termios *);
14070 xSI pid_t tcgetsid(int);
14071 int tcsendbreak(int, int);
14072 int tcsetattr(int, int, struct termios *);
    
```

14073 **APPLICATION USAGE**

14074 The following names are commonly used as extensions to the above, therefore portable
14075 applications must not use them:

14076 xSI	CBAUD	EXTB	VDSUSP
14077	DEFECHO	FLUSHO	VLNEXT
14078	ECHOCTL	LOBLK	VREPRINT
14079	ECHOKE	PENDIN	VSTATUS
14080	ECHOPRT	SWTCH	VWERASE
14081	EXTA	VDISCARD	

14082 **Note:** These names are not used in IEEE Std. 1003.1-200x, but are reserved for historical use.

14083 **RATIONALE**

14084 None.

14085 **FUTURE DIRECTIONS**

14086 None.

14087 **SEE ALSO**

14088 The System Interfaces volume of IEEE Std. 1003.1-200x, *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*,
14089 *cfsetospeed()*, *tcdrain()*, *tcflow()*, *tcf flush()*, *tcgetattr()*, *tcgetsid()*, *tcsendbreak()*, *tcsetattr()*, Chapter
14090 11 (on page 213)

14091 **CHANGE HISTORY**

14092 First released in Issue 3.

14093 Entry included for alignment with the ISO POSIX-1 standard.

14094 **Issue 4**

14095 The following words are removed from the description of the `c_cc` array: “Implementations that
14096 do not support the job control option, may ignore the SUSP character value in the `c_cc` array
14097 indexed by the VSUSP subscript.” This is because job control is defined as mandatory for Issue 4
14098 conforming implementations.

14099 The mask name symbols IUCLC and OLCUC are marked LEGACY.

14100 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 14101 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 14102 • Some minor rewording of the DESCRIPTION is done to align the text more exactly with the
14103 ISO POSIX-1 standard. No functional differences are implied by these changes.
- 14104 • The list of mask name symbols for the `c_oflag` field have all been marked as extensions, with
14105 the exception of OPOST.

14106 **Issue 4, Version 2**

14107 For X/OPEN UNIX conformance, the `tcgetsid()` function is added to the list of functions declared
14108 in this header.

14109 **Issue 6**

14110 The LEGACY symbols IUCLC, ULCUC, and XCASE are removed.

14111 NAME

14112 tgmath.h — type-generic macros

14113 SYNOPSIS

14114 #include <tgmath.h>

14115 DESCRIPTION

14116 cx The functionality described on this reference page extends the ISO C standard. Applications
 14117 shall define the appropriate feature test macro (see the System Interfaces volume of
 14118 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 14119 symbols in this header.

14120 The <tgmath.h> header shall include the headers <math.h> and <complex.h> and shall define
 14121 several type-generic macros.

14122 Of the functions contained within the <math.h> and <complex.h> headers without an *f* (**float**) or
 14123 *l* (**long double**) suffix, several have one or more parameters whose corresponding real type is
 14124 **double**. For each such function, except *modf()*, there shall be a corresponding type-generic
 14125 macro. The parameters whose corresponding real type is **double** in the function synopsis are
 14126 generic parameters. Use of the macro invokes a function whose corresponding real type and
 14127 type domain are determined by the arguments for the generic parameters.

14128 Use of the macro invokes a function whose generic parameters have the corresponding real type
 14129 determined as follows:

- 14130 • First, if any argument for generic parameters has type **long double**, the type determined is
 14131 **long double**.
- 14132 • Otherwise, if any argument for generic parameters has type **double** or is of integer type, the
 14133 type determined is **double**.
- 14134 • Otherwise, the type determined is **float**.

14135 For each unsuffixed function in the <math.h> header for which there is a function in the
 14136 <complex.h> header with the same name except for a *c* prefix, the corresponding type-generic
 14137 macro (for both functions) has the same name as the function in the <math.h> header. The
 14138 corresponding type-generic macro for *fabs()* and *cabs()* is *fabs()*.

14139
14140
14141
14142
14143
14144
14145
14146
14147
14148
14149
14150
14151
14152
14153
14154
14155
14156
14157
14158

<math.h> Function	<complex.h> Function	Type-Generic Macro
<i>acos()</i>	<i>cacos()</i>	<i>acos()</i>
<i>asin()</i>	<i>casin()</i>	<i>asin()</i>
<i>atan()</i>	<i>catan()</i>	<i>atan()</i>
<i>acosh()</i>	<i>cacosh()</i>	<i>acosh()</i>
<i>asinh()</i>	<i>casinh()</i>	<i>asinh()</i>
<i>atanh()</i>	<i>catanh()</i>	<i>atanh()</i>
<i>cos()</i>	<i>ccos()</i>	<i>cos()</i>
<i>sin()</i>	<i>csin()</i>	<i>sin()</i>
<i>tan()</i>	<i>ctan()</i>	<i>tan()</i>
<i>cosh()</i>	<i>ccosh()</i>	<i>cosh()</i>
<i>sinh()</i>	<i>csinh()</i>	<i>sinh()</i>
<i>tanh()</i>	<i>ctanh()</i>	<i>tanh()</i>
<i>exp()</i>	<i>cexp()</i>	<i>exp()</i>
<i>log()</i>	<i>clog()</i>	<i>log()</i>
<i>pow()</i>	<i>cpow()</i>	<i>pow()</i>
<i>sqrt()</i>	<i>csqrt()</i>	<i>sqrt()</i>
<i>fabs()</i>	<i>cabs()</i>	<i>fabs()</i>

14159 If at least one argument for a generic parameter is complex, then use of the macro invokes a
14160 complex function; otherwise, use of the macro invokes a real function.

14161 For each unsuffixed function in the <math.h> header without a c-prefixed counterpart in the
14162 <complex.h> header, the corresponding type-generic macro has the same name as the function.
14163 These type-generic macros are:

14164	<i>atan2()</i>	<i>fma()</i>	<i>llround()</i>	<i>remainder()</i>
14165	<i>cbrt()</i>	<i>fmax()</i>	<i>log10()</i>	<i>remquo()</i>
14166	<i>ceil()</i>	<i>fmin()</i>	<i>log1p()</i>	<i>rint()</i>
14167	<i>copysign()</i>	<i>fmod()</i>	<i>log2()</i>	<i>round()</i>
14168	<i>erf()</i>	<i>frexp()</i>	<i>logb()</i>	<i>scalbn()</i>
14169	<i>erfc()</i>	<i>hypot()</i>	<i>lrint()</i>	<i>scalbln()</i>
14170	<i>exp2()</i>	<i>ilogb()</i>	<i>lround()</i>	<i>tgamma()</i>
14171	<i>expm1()</i>	<i>ldexp()</i>	<i>nearbyint()</i>	<i>trunc()</i>
14172	<i>fdim()</i>	<i>lgamma()</i>	<i>nextafter()</i>	
14173	<i>floor()</i>	<i>llrint()</i>	<i>nexttoward()</i>	

14174 If all arguments for generic parameters are real, then use of the macro invokes a real function;
14175 otherwise, use of the macro results in undefined behavior.

14176 For each unsuffixed function in the <complex.h> header that is not a c-prefixed counterpart to a
14177 function in the <math.h> header, the corresponding type-generic macro has the same name as
14178 the function. These type-generic macros are:

14179	<i>carg()</i>
14180	<i>cimag()</i>
14181	<i>conj()</i>
14182	<i>cproj()</i>
14183	<i>creal()</i>

14184 Use of the macro with any real or complex argument invokes a complex function.

14185 **APPLICATION USAGE**

14186 With the declarations:

```

14187 #include <tgmath.h>
14188 int n;
14189 float f;
14190 double d;
14191 long double ld;
14192 float complex fc;
14193 double complex dc;
14194 long double complex ldc;
    
```

14195 functions invoked by use of type-generic macros are shown in the following table:

Macro	Use Invokes
<i>exp(n)</i>	<i>exp(n)</i> , the function
<i>acosh(f)</i>	<i>acoshf(f)</i>
<i>sin(d)</i>	<i>sin(d)</i> , the function
<i>atan(ld)</i>	<i>atanl(ld)</i>
<i>log(fc)</i>	<i>clogf(fc)</i>
<i>sqrt(dc)</i>	<i>csqrt(dc)</i>
<i>pow(ldc,f)</i>	<i>cpowl(ldc, f)</i>
<i>remainder(n,n)</i>	<i>remainder(n, n)</i> , the function
<i>nextafter(d,f)</i>	<i>nextafter(d, f)</i> , the function
<i>nexttoward(f,ld)</i>	<i>nexttowardf(f, ld)</i>
<i>copysign(n,ld)</i>	<i>copysignl(n, ld)</i>
<i>ceil(fc)</i>	Undefined behavior
<i>rint(dc)</i>	Undefined behavior
<i>fmax(ldc,ld)</i>	Undefined behavior
<i>carg(n)</i>	<i>carg(n)</i> , the function
<i>cproj(f)</i>	<i>cprojf(f)</i>
<i>creal(d)</i>	<i>creal(d)</i> , the function
<i>cimag(ld)</i>	<i>cimagl(ld)</i>
<i>cabs(fc)</i>	<i>cabsf(fc)</i>
<i>carg(dc)</i>	<i>carg(dc)</i> , the function
<i>cproj(ldc)</i>	<i>cprojl(ldc)</i>

14218 **RATIONALE**

14219 Type-generic macros allow calling a function whose type is determined by the argument type, as
 14220 is the case for C operators such as '+' and '*'. For example, with a type-generic *cos()* macro,
 14221 the expression *cos((float)x)* will have type **float**. This feature enables writing more portably
 14222 efficient code and alleviates need for awkward casting and suffixing in the process of porting or
 14223 adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

14224 The only arguments that affect the type resolution are the arguments corresponding to the
 14225 parameters that have type **double** in the synopsis. Hence the type of a type-generic call to
 14226 *nexttoward()*, whose second parameter is **long double** in the synopsis, is determined solely by
 14227 the type of the first argument.

14228 The term “type-generic” was chosen over the proposed alternatives of intrinsic and overloading.
 14229 The term is more specific than intrinsic, which already is widely used with a more general
 14230 meaning, and reflects a closer match to Fortran’s generic functions than to C++ overloading.

14231 The macros are placed in their own header in order not to silently break old programs that
 14232 include the <math.h> header; for example, with:

14233 `printf ("%e", sin(x))`

14234 *modf(double, double*)* is excluded because no way was seen to make it safe without
14235 complicating the type resolution.

14236 The implementation might, as an extension, endow appropriate ones of the macros that
14237 IEEE Std. 1003.1-200x specifies only for real arguments with the ability to invoke the complex
14238 functions.

14239 IEEE Std. 1003.1-200x does not prescribe any particular implementation mechanism for generic
14240 macros. It could be implemented simply with built-in macros. The generic macro for *sqrt()*, for
14241 example, could be implemented with:

14242 `#undef sqrt`

14243 `#define sqrt(x) __BUILTIN_GENERIC_sqrt(x)`

14244 Generic macros are designed for a useful level of consistency with C++ overloaded math
14245 functions.

14246 The great majority of existing C programs are expected to be unaffected when the **<tgmath.h>**
14247 header is included instead of the **<math.h>** or **<complex.h>** headers. Generic macros are similar
14248 to the ISO/IEC 9899:1999 standard library masking macros, though the semantic types of return
14249 values differ.

14250 The ability to overload on integer as well as floating types would have been useful for some
14251 functions; for example, *copysign()*. Overloading with different numbers of arguments would
14252 have allowed reusing names; for example, *remainder()* for *remquo()*. However, these facilities
14253 would have complicated the specification; and their natural consistent use, such as for a floating
14254 *abs()* or a two-argument *atan()*, would have introduced further inconsistencies with the
14255 ISO/IEC 9899:1999 standard for insufficient benefit.

14256 The ISO C standard in no way limits the implementation's options for efficiency, including
14257 inlining library functions.

14258 **FUTURE DIRECTIONS**

14259 None.

14260 **SEE ALSO**

14261 **<math.h>**, **<complex.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, *cabs()*, *fabs()*,
14262 *modf()*

14263 **CHANGE HISTORY**

14264 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

14265 **NAME**

14266 time.h — time types

14267 **SYNOPSIS**

14268 #include <time.h>

14269 **DESCRIPTION**

14270 CX The functionality described on this reference page extends the ISO C standard. Applications
 14271 shall define the appropriate feature test macro (see the System Interfaces volume of
 14272 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 14273 symbols in this header.

14274 The <time.h> header shall declare the structure **tm**, which shall include at least the following
 14275 members:

14276	int	tm_sec	Seconds [0,60].
14277	int	tm_min	Minutes [0,59].
14278	int	tm_hour	Hour [0,23].
14279	int	tm_mday	Day of month [1,31].
14280	int	tm_mon	Month of year [0,11].
14281	int	tm_year	Years since 1900.
14282	int	tm_wday	Day of week [0,6] (Sunday =0).
14283	int	tm_yday	Day of year [0,365].
14284	int	tm_isdst	Daylight savings flag.

14285 The value of *tm_isdst* shall be positive if Daylight Saving Time is in effect, 0 if Daylight Saving
 14286 Time is not in effect, and negative if the information is not available.

14287 The <time.h> header shall define the following symbolic names:

14288	NULL	Null pointer constant.
14289	CLOCKS_PER_SEC	A number used to convert the value returned by the <i>clock()</i> function into 14290 seconds.

14291	TMR CPT	CLOCK_PROCESS_CPUTIME_ID
14292		The identifier of the CPU-time clock associated with the process making a 14293 <i>clock()</i> or <i>timer*()</i> function call.

14294	TMR TCT	CLOCK_THREAD_CPUTIME_ID
14295		The identifier of the CPU-time clock associated with the thread making a 14296 <i>clock()</i> or <i>timer*()</i> function call.

14297 TMR The <time.h> header shall declare the structure **timespec**, which has at least the following
 14298 members:

14299	time_t	tv_sec	Seconds.
14300	long	tv_nsec	Nanoseconds.

14301 The <time.h> header shall also declare the **itimerspec** structure, which has at least the following
 14302 members:

14303	struct timespec	it_interval	Timer period.
14304	struct timespec	it_value	Timer expiration.

14305 The following manifest constants shall be defined:

14306	CLOCK_REALTIME	The identifier of the system-wide realtime clock.
14307	TIMER_ABSTIME	Flag indicating time is absolute with respect to the clock associated with a 14308 timer.

14309 MON **CLOCK_MONOTONIC**
 14310 The identifier for the system-wide monotonic clock, which is defined as a
 14311 clock whose value cannot be set via *clock_settime()* and which cannot
 14312 have backward clock jumps. The maximum possible clock jump shall be
 14313 implementation-defined.

14314 TMR The **clock_t**, **size_t**, **time_t**, **clockid_t**, and **timer_t** types shall be defined as described in
 14315 **<sys/types.h>**.

14316 XSI Although the value of **CLOCKS_PER_SEC** is required to be 1 million on all XSI-conformant
 14317 systems, it may be variable on other systems, and it should not be assumed that
 14318 **CLOCKS_PER_SEC** is a compile-time constant.

14319 XSI The **<time.h>** header shall provide a declaration for *getdate_err*.

14320 The following shall be declared as functions and may also be defined as macros. Function
 14321 prototypes shall be provided for use with an ISO C standard compiler.

14322 char *asctime(const struct tm *);
 14323 TSF char *asctime_r(const struct tm *restrict, char *restrict);
 14324 clock_t clock(void);
 14325 CPT int clock_getcpuclockid(pid_t, clockid_t *);
 14326 TMR int clock_getres(clockid_t, struct timespec *);
 14327 int clock_gettime(clockid_t, struct timespec *);
 14328 CS int clock_nanosleep(clockid_t, int, const struct timespec *,
 14329 struct timespec *);
 14330 TMR int clock_settime(clockid_t, const struct timespec *);
 14331 char *ctime(const time_t *);
 14332 TSF char *ctime_r(const time_t *, char *);
 14333 double difftime(time_t, time_t);
 14334 XSI struct tm *getdate(const char *);
 14335 struct tm *gmtime(const time_t *);
 14336 struct tm *gmtime_r(const time_t *restrict, struct tm *restrict);
 14337 struct tm *localtime(const time_t *);
 14338 TSF struct tm *localtime_r(const time_t *restrict, struct tm *restrict);
 14339 time_t mktime(struct tm *);
 14340 TMR int nanosleep(const struct timespec *, struct timespec *);
 14341 size_t strftime(char *restrict, size_t, const char *restrict,
 14342 const struct tm *restrict);
 14343 XSI char *strptime(const char *restrict, const char *restrict,
 14344 struct tm *restrict);
 14345 time_t time(time_t *);
 14346 TMR int timer_create(clockid_t, struct sigevent *restrict,
 14347 timer_t *restrict);
 14348 int timer_delete(timer_t);
 14349 int timer_gettime(timer_t, struct itimerspec *);
 14350 int timer_getoverrun(timer_t);
 14351 int timer_settime(timer_t, int, const struct itimerspec *restrict,
 14352 struct itimerspec *restrict);
 14353 void tzset(void);

14354 The following shall be declared as variables:

14355 XSI extern int daylight;
 14356 extern long timezone;
 14357 extern char *tzname[];

14358 **APPLICATION USAGE**

14359 The range [0,61] for *tm_sec* allows for the occasional leap second or double leap second.

14360 *tm_year* is a signed value; therefore, years before 1900 may be represented.

14361 To obtain the number of clock ticks per second returned by the *times()* function, applications
14362 should call *sysconf(_SC_CLK_TCK)*.

14363 **RATIONALE**

14364 None.

14365 **FUTURE DIRECTIONS**

14366 None.

14367 **SEE ALSO**

14368 <**sys/types.h**>, the System Interfaces volume of IEEE Std. 1003.1-200x, *asctime()*, *clock()*,
14369 *clock_getcpuclockid()*, *clock_getres()*, *clock_nanosleep()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*,
14370 *localtime()*, *mktime()*, *nanosleep()*, *strftime()*, *strptime()*, *sysconf()*, *time()*, *timer_create()*,
14371 *timer_delete()*, *timer_getoverrun()*, *tzname()*, *tzset()*, *utime()*, the Shell and Utilities volume of
14372 IEEE Std. 1003.1-200x, *daylight*, *timezone*

14373 **CHANGE HISTORY**

14374 First released in Issue 1. Derived from Issue 1 of the SVID.

14375 **Issue 4**

14376 The symbolic name CLK_TCK is marked as an extension and LEGACY. Warnings about its use
14377 are also added to the DESCRIPTION.

14378 Reference to the <**sys/types.h**> header is added for the definitions of **clock_t**, **size_t**, and **time_t**.

14379 References to CLK_TCK are changed to CLOCKS_PER_SEC in part of the DESCRIPTION. The
14380 fact that CLOCKS_PER_SEC is always one millionth of a second on XSI-conformant systems is
14381 also marked as an extension.

14382 External declarations for *daylight*, *timezone*, and *tzname* are added. The first two are marked as
14383 extensions.

14384 The *strptime()* function is added to the list of functions declared in this header.

14385 A note about the settings of *tm_sec* is added to the APPLICATION USAGE section.

14386 The following changes are incorporated for alignment with the ISO C standard:

- 14387 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 14388 • The range of *tm_min* is changed from [0,61] to [0,59].
- 14389 • Possible settings of *tm_isdst* and their meanings are added.
- 14390 • The *clock()* and *difftime()* functions are added to the list of functions declared in this header.

14391 **Issue 4, Version 2**

14392 The following changes are incorporated for X/OPEN UNIX conformance:

- 14393 • The <**time.h**> header provides a declaration for *getdate_err*.
- 14394 • The *getdate()* function is added to the list of functions declared in this header.

14395 **Issue 5**

14396 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
14397 Threads Extension.

14398 **Issue 6**

14399 The Open Group corrigenda item U035/6 has been applied. In the DESCRIPTION, the types
14400 **clockid_t** and **timer_t** have been described.

14401 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 14402 • The POSIX timer-related functions are now marked as part of the Timers option.

14403 The symbolic name CLK_TCK is removed. Application usage is added describing how its
14404 equivalent functionality can be obtained using *sysconf()*.

14405 The *clock_getcpuclockid()* function and manifest constants CLOCK_PROCESS_CPUTIME_ID and
14406 CLOCK_THREAD_CPUTIME_ID are added for alignment with IEEE Std. 1003.1d-1999.

14407 The manifest constant CLOCK_MONOTONIC and the *clock_nanosleep()* function are added for
14408 alignment with IEEE Std. 1003.1j-2000.

14409 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- 14410 • The range for seconds is changed from 0,61 to 0.60.
- 14411 • The **restrict** keyword is added to the prototypes for *asctime_r()*, *gmtime_r()*, *localtime_r()*,
14412 *strftime()*, *strptime()*, *timer_create()*, and *timer_settime()*.

14413 **NAME**

14414 trace.h — tracing

14415 **SYNOPSIS**

14416 TRC #include <tracing.h>

14417

14418 **DESCRIPTION**

14419 The <trace.h> header shall define the **posix_trace_event_info** structure that includes at least the
 14420 following members:

14421	trace_event_id_t	posix_event_id
14422	pid_t	posix_pid
14423	void	*posix_prog_address
14424	int	posix_truncation_status
14425	struct timespec	posix_timestamp
14426 THR	pthread_t	posix_thread_id

14427

14428 The <trace.h> header shall define the **posix_trace_status_info** structure that includes at least the
 14429 following members:

14430	int	posix_stream_status
14431	int	posix_stream_full_status
14432	int	posix_stream_overrun_status
14433 TRL	int	posix_stream_flush_status
14434	int	posix_stream_flush_error
14435	int	posix_log_overrun_status
14436	int	posix_log_full_status

14437

14438 The <trace.h> header shall define the following symbols:

14439	POSIX_TRACE_RUNNING
14440	POSIX_TRACE_SUSPENDED
14441	POSIX_TRACE_FULL
14442	POSIX_TRACE_NOT_FULL
14443	POSIX_TRACE_NO_OVERRUN
14444	POSIX_TRACE_OVERRUN
14445 TRL	POSIX_TRACE_FLUSHING
14446	POSIX_TRACE_NOT_FLUSHING
14447	POSIX_TRACE_NOT_TRUNCATED
14448	POSIX_TRACE_TRUNCATED_READ
14449	POSIX_TRACE_TRUNCATED_RECORD
14450 TRL	POSIX_TRACE_FLUSH
14451	POSIX_TRACE_LOOP
14452	POSIX_TRACE_UNTIL_FULL
14453 TRI	POSIX_TRACE_CLOSE_FOR_CHILD
14454	POSIX_TRACE_INHERITED
14455 TRL	POSIX_TRACE_APPEND
14456	POSIX_TRACE_LOOP
14457	POSIX_TRACE_UNTIL_FULL
14458 TEF	POSIX_TRACE_FILTER
14459 TRL	POSIX_TRACE_FLUSH_START
14460	POSIX_TRACE_FLUSH_STOP
14461	POSIX_TRACE_OVERFLOW

14462 POSIX_TRACE_RESUME
 14463 POSIX_TRACE_START
 14464 POSIX_TRACE_STOP
 14465 POSIX_TRACE_UNNAMED_USER_EVENT

14466 The following types shall be defined as described in **<sys/types.h>**:

14467 **trace_attr_t**
 14468 **trace_id_t**
 14469 **trace_event_id_t**
 14470 TEF **trace_event_set_t**
 14471

14472 The following shall be declared as functions and may also be declared as macros. Function
 14473 prototypes shall be provided for use with an ISO C standard compiler.

```

14474 int posix_trace_attr_destroy(trace_attr_t *);
14475 int posix_trace_attr_getclockres(const trace_attr_t *,
14476     struct timespec *);
14477 int posix_trace_attr_getcreatetime(const trace_attr_t *,
14478     struct timespec *);
14479 int posix_trace_attr_getgenversion(const trace_attr_t *, char *);
14480 TRI int posix_trace_attr_getinherited(const trace_attr_t *, int *);
14481 TRL int posix_trace_attr_getlogfullpolicy(const trace_attr_t *, int *);
14482 int posix_trace_attr_getlogsize(const trace_attr_t *, size_t *);
14483 int posix_trace_attr_getmaxdatasize(const trace_attr_t *, size_t *);
14484 int posix_trace_attr_getmaxsystemevents(size_t *,
14485     size_t *);
14486 int posix_trace_attr_getmaxuserevents(size_t *,
14487     size_t *);
14488 int posix_trace_attr_getname(const trace_attr_t *, char *);
14489 int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *, int *);
14490 int posix_trace_attr_getstreamsize(const trace_attr_t *, size_t *);
14491 int posix_trace_attr_init(trace_attr_t *);
14492 TRI int posix_trace_attr_setinherited(trace_attr_t *, int);
14493 TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *, int);
14494 int posix_trace_attr_setlogsize(trace_attr_t *, size_t);
14495 int posix_trace_attr_setmaxdatasize(trace_attr_t *, size_t);
14496 int posix_trace_attr_setname(trace_attr_t *, const char *);
14497 int posix_trace_attr_setstreamsize(trace_attr_t *, size_t);
14498 int posix_trace_attr_setstreamfullpolicy(trace_attr_t *, int);
14499 int posix_trace_clear(trace_id_t);
14500 TRL int posix_trace_close(trace_id_t);
14501 int posix_trace_create(pid_t, const trace_attr_t *, trace_id_t *);
14502 TRL int posix_trace_create_withlog(pid_t, const trace_attr_t *, int,
14503     trace_id_t *);
14504 void posix_trace_event(trace_event_id_t, const void *, size_t);
14505 int posix_trace_eventid_equal(trace_id_t, trace_eventid_t,
14506     trace_eventid_t);
14507 int posix_trace_eventid_get_name(trace_id_t, trace_eventid_t, char *);
14508 int posix_trace_eventid_open(const char *, trace_event_id_t *);
14509 int posix_trace_eventtypelist_getnext_id(trace_id_t, trace_eventid_t *,
14510     int *);
14511 int posix_trace_eventtypelist_rewind(trace_id_t);

```

```

14512 TEF int posix_trace_eventset_add(trace_event_id_t, trace_event_set_t *);
14513 int posix_trace_eventset_del(trace_event_id_t, trace_event_set_t *);
14514 int posix_trace_eventset_empty(trace_event_set_t *);
14515 int posix_trace_eventset_fill(trace_event_set_t *, int);
14516 int posix_trace_eventset_ismember(trace_event_id_t,
14517     const trace_event_set_t *, int *);
14518 int posix_trace_flush(trace_id_t);
14519 int posix_trace_get_attr(trace_id_t, trace_attr_t *);
14520 TEF int posix_trace_get_filter(trace_id_t, trace_event_set_t *);
14521 int posix_trace_get_status(trace_id_t,
14522     struct posix_trace_status_info *);
14523 int posix_trace_getnext_event(trace_id_t,
14524     struct posix_trace_event_info *, void *, size_t, size_t *,
14525     int *);
14526 TRL int posix_trace_open(int, trace_id_t *);
14527 int posix_trace_rewind(trace_id_t);
14528 TEF int posix_trace_set_filter(trace_id_t, const trace_event_set_t *, int);
14529 int posix_trace_shutdown(trace_id_t);
14530 int posix_trace_start(trace_id_t);
14531 int posix_trace_stop(trace_id_t);
14532 TMO int posix_trace_timedgetnext_event(trace_id_t,
14533     struct posix_trace_event_info *, void *, size_t, size_t *,
14534     int *, const struct timespec *);
14535 TEF int posix_trace_trid_eventid_open(trace_id_t, const char *,
14536     trace_eventid_t *);
14537 int posix_trace_trygetnext_event(trace_id_t,
14538     struct posix_trace_event_info *, void *, size_t, size_t *,
14539     int *);

```

14540 **APPLICATION USAGE**

14541 None.

14542 **RATIONALE**

14543 None.

14544 **FUTURE DIRECTIONS**

14545 None.

14546 **SEE ALSO**

14547 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, Section 2.11, Tracing, the
14548 System Interfaces volume of IEEE Std. 1003.1-200x, *posix_trace_attr_destroy()*,
14549 *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*, *posix_trace_attr_getgenversion()*,
14550 *posix_trace_attr_getinherited()*, *posix_trace_attr_getlogfullpolicy()*, *posix_trace_attr_getlogsize()*,
14551 *posix_trace_attr_getmaxdatasize()*, *posix_trace_attr_getmaxsystemeventsizesize()*,
14552 *posix_trace_attr_getmaxusereventsizesize()*, *posix_trace_attr_getname()*,
14553 *posix_trace_attr_getstreamfullpolicy()*, *posix_trace_attr_getstreamsize()*, *posix_trace_attr_init()*,
14554 *posix_trace_attr_setinherited()*, *posix_trace_attr_setlogfullpolicy()*, *posix_trace_attr_setlogsize()*,
14555 *posix_trace_attr_setmaxdatasize()*, *posix_trace_attr_setname()*, *posix_trace_attr_setstreamsize()*,
14556 *posix_trace_attr_setstreamfullpolicy()*, *posix_trace_clear()*, *posix_trace_close()*, *posix_trace_create()*,
14557 *posix_trace_create_withlog()*, *posix_trace_event()*, *posix_trace_eventid_equal()*,
14558 *posix_trace_eventid_get_name()*, *posix_trace_eventid_open()*, *posix_trace_eventtypelist_getnext_id()*,
14559 *posix_trace_eventtypelist_rewind()*, *posix_trace_eventset_add()*, *posix_trace_eventset_del()*,
14560 *posix_trace_eventset_empty()*, *posix_trace_eventset_fill()*, *posix_trace_eventset_ismember()*,
14561 *posix_trace_flush()*, *posix_trace_get_attr()*, *posix_trace_get_filter()*, *posix_trace_get_status()*,

14562 *posix_trace_getnext_event(), posix_trace_open(), posix_trace_rewind(), posix_trace_set_filter(),*
14563 *posix_trace_shutdown(), posix_trace_start(), posix_trace_stop(), posix_trace_timedgetnext_event(),*
14564 *posix_trace_trid_eventid_open(), posix_trace_trygetnext_event()*

14565 **CHANGE HISTORY**

14566 First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

14567 **NAME**

14568 ucontext.h — user context

14569 **SYNOPSIS**

14570 XSI `#include <ucontext.h>`

14571

14572 **DESCRIPTION**

14573 The <ucontext.h> header shall define the **mcontext_t** type through **typedef**.

14574 The <ucontext.h> header shall define the **ucontext_t** type as a structure that shall include at least
14575 the following members:

14576	ucontext_t *uc_link	Pointer to the context that is resumed
14577		when this context returns.
14578	sigset_t uc_sigmask	The set of signals that are blocked when this
14579		context is active.
14580	stack_t uc_stack	The stack used by this context.
14581	mcontext_t uc_mcontext	A machine-specific representation of the saved
14582		context.

14583 The types **sigset_t** and **stack_t** shall be defined as in <signal.h>.

14584 The following shall be declared as functions and may also be defined as macros, Function
14585 prototypes shall be provided for use with an ISO C standard compiler.

```
14586 int getcontext(ucontext_t *);
14587 int setcontext(const ucontext_t *);
14588 void makecontext(ucontext_t *, void (*)(void), int, ...);
14589 int swapcontext(ucontext_t *restrict, const ucontext_t *restrict);
```

14590 **APPLICATION USAGE**

14591 None.

14592 **RATIONALE**

14593 None.

14594 **FUTURE DIRECTIONS**

14595 None.

14596 **SEE ALSO**

14597 <signal.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *getcontext()*, *makecontext()*,
14598 *sigaction()*, *sigprocmask()*, *sigaltstack()*

14599 **CHANGE HISTORY**

14600 First released in Issue 4, Version 2.

14601 **NAME**

14602 ulimit.h — ulimit commands

14603 **SYNOPSIS**

14604 XSI #include <ulimit.h>

14605

14606 **DESCRIPTION**14607 The <ulimit.h> header shall define the symbolic constants used by the *ulimit()* function.

14608 Symbolic constants:

14609 UL_GETFSIZE Get maximum file size.

14610 UL_SETFSIZE Set maximum file size.

14611 The following shall be declared as a function and may also be defined as a macro. Function
14612 prototypes shall be provided for use with an ISO C standard compiler.

14613 long ulimit(int, ...);

14614 **APPLICATION USAGE**

14615 None.

14616 **RATIONALE**

14617 None.

14618 **FUTURE DIRECTIONS**

14619 None.

14620 **SEE ALSO**14621 The System Interfaces volume of IEEE Std. 1003.1-200x, *ulimit()*14622 **CHANGE HISTORY**

14623 First released in Issue 3.

14624 **Issue 4**

14625 The function declarations in this header are expanded to full ISO C standard prototypes.

14626 **NAME**

14627 unistd.h — standard symbolic constants and types

14628 **SYNOPSIS**

14629 #include <unistd.h>

14630 **DESCRIPTION**

14631 The <unistd.h> header defines miscellaneous symbolic constants and types, and declares
 14632 miscellaneous functions. The actual value of the constants are unspecified except as shown. The
 14633 contents of this header are shown below.

14634 **Version Test Macros**

14635 The following symbolic constants shall be defined:

14636 `_POSIX_VERSION`

14637 Integer value indicating version of IEEE Std. 1003.1-200x (C-language binding). The value is
 14638 200xxxL. This value shall be used for systems that conform to IEEE Std. 1003.1-200x.

14639 `_POSIX2_VERSION`

14640 Integer value indicating version of the Shell and Utilities volume of IEEE Std. 1003.1-200x.

14641 XSI `_XOPEN_VERSION`

14642 Integer value indicating version of the X/Open Portability Guide to which the
 14643 implementation conforms. The value is 600.

14644 XSI `_XOPEN_XCU_VERSION` is defined as an integer value indicating the version of the Shell and
 14645 Utilities volume of IEEE Std. 1003.1-200x to which the implementation conforms. If the value is
 14646 -1, no commands and utilities are provided on the implementation. If the value is greater than
 14647 or equal to 4, the functionality associated with the following symbols is also supported (see
 14648 **Constants for Options and Option Groups** (on page 438) and **Constants for Profiling Option**
 14649 **Groups** (on page 444)):

14650 `_POSIX2_C_BIND`

14651 `_POSIX2_CHAR_TERM`

14652 `_POSIX2_LOCALEDEF`

14653 `_POSIX2_UPE`

14654 `_POSIX2_VERSION`

14655 If `_XOPEN_XCU_VERSION` is not defined, use the `sysconf()` function to determine which
 14656 features are supported.

14657 Each of the following symbolic constants shall be defined only if the implementation supports
 14658 the indicated version of the X/Open Portability Guide:

14659 XSI `_XOPEN_UNIX`

14660 X/Open CAE Specification, January 1997, System Interfaces and Headers, Issue 5
 14661 (ISBN: 1-85912-181-0, C606).

14662 `_XOPEN_XPG2`

14663 X/Open Portability Guide, Volume 2, January 1987, XVS System Calls and Libraries
 14664 (ISBN: 0-444-70175-3).

14665 `_XOPEN_XPG3`

14666 X/Open Specification, February 1992, System Interfaces and Headers, Issue 3
 14667 (ISBN: 1-872630-37-5, C212); this specification was formerly X/Open Portability Guide,
 14668 Issue 3, Volume 2, January 1989, XSI System Interface and Headers (ISBN: 0-13-685843-0,
 14669 XO/XPG/89/003).

14670 **_XOPEN_XPG4**
 14671 X/Open CAE Specification, July 1992, System Interfaces and Headers, Issue 4
 14672 (ISBN: 1-872630-47-2, C202).

14673 **Constants for Options and Option Groups**

14674 The following symbolic constants, if defined in <unistd.h>, shall have a value of -1, 0, or greater,
 14675 unless otherwise specified below. If these are undefined, the *sysconf()* function can be used to
 14676 determine whether the option is provided for a particular invocation of the application.

14677 If a symbolic constant is defined with the value -1, the option is not supported. Headers, data
 14678 types, and function interfaces required only for the option need not be supplied. An application
 14679 that attempts to use anything associated only with the option is considered to be requiring an
 14680 extension.

14681 If a symbolic constant is defined with a value greater than zero, the option shall always be
 14682 supported when the application is executed. All headers, data types, and functions shall be
 14683 present and shall operate as specified.

14684 If a symbolic constant is defined with the value zero, all headers, data types, and functions shall
 14685 be present. The application must check at runtime to see whether the option is supported by
 14686 calling *sysconf()* with the indicated *name* parameter.

14687 Unless explicitly specified otherwise, the behavior of functions associated with an unsupported
 14688 option is unspecified, and an application that uses such functions without first checking
 14689 *sysconf()* is considered to be requiring an extension.

14690 For conformance requirements, refer to Chapter 2 (on page 19).

14691 ADV **_POSIX_ADVISORY_INFO**
 14692 The implementation supports the Advisory Information option. If this symbol has a value
 14693 other than -1, it shall have the value 200ymmL, the date of approval of
 14694 IEEE Std. 1003.1-200x.

14695 AIO **_POSIX_ASYNCHRONOUS_IO**
 14696 The implementation supports the Asynchronous Input and Output option. If this symbol
 14697 has a value other than -1, it shall have the value 200ymmL, the date of approval of
 14698 IEEE Std. 1003.1-200x.

14699 BAR **_POSIX_BARRIERS**
 14700 The implementation supports the Barriers option. If this symbol has a value other than -1, it
 14701 shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.

14702 **_POSIX_CHOWN_RESTRICTED**
 14703 The use of *chown()* and *fchown()* is restricted to a process with appropriate privileges, and
 14704 to changing the group ID of a file only to the effective group ID of the process or to one of
 14705 its supplementary group IDs.

14706 CS **_POSIX_CLOCK_SELECTION**
 14707 The implementation supports the Clock Selection option. If this symbol has a value other
 14708 than -1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.

14709 CPT **_POSIX_CPUTIME**
 14710 The implementation supports the Process CPU-Time Clocks option. If this symbol has a
 14711 value other than -1, it shall have the value 200ymmL, the date of approval of
 14712 IEEE Std. 1003.1-200x.

14713 FSC **_POSIX_FSYNC**
 14714 The implementation supports the File Synchronization option. If this symbol has a value

14715	other than -1, it shall have the value 200ymmL, the date of approval of
14716	IEEE Std. 1003.1-200x.
14717	_POSIX_JOB_CONTROL
14718	The implementation supports job control. This is always set to a value greater than zero.
14719 MF	_POSIX_MAPPED_FILES
14720	The implementation supports the Memory Mapped Files option. If this symbol has a value
14721	other than -1, it shall have the value 200ymmL, the date of approval of
14722	IEEE Std. 1003.1-200x.
14723 ML	_POSIX_MEMLOCK
14724	The implementation supports the Process Memory Locking option. If this symbol has a
14725	value other than -1, it shall have the value 200ymmL, the date of approval of
14726	IEEE Std. 1003.1-200x.
14727 MLR	_POSIX_MEMLOCK_RANGE
14728	The implementation supports the Range Memory Locking option. If this symbol has a value
14729	other than -1, it shall have the value 200ymmL, the date of approval of
14730	IEEE Std. 1003.1-200x.
14731 MPR	_POSIX_MEMORY_PROTECTION
14732	The implementation supports the Memory Protection option. If this symbol has a value
14733	other than -1, it shall have the value 200ymmL, the date of approval of
14734	IEEE Std. 1003.1-200x.
14735 MSG	_POSIX_MESSAGE_PASSING
14736	The implementation supports the Message Passing option. If this symbol has a value other
14737	than -1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.
14738 MON	_POSIX_MONOTONIC_CLOCK
14739	The implementation supports the Monotonic Clock option. If this symbol has a value other
14740	than -1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.
14741	_POSIX_NO_TRUNC
14742	Path name components longer than {NAME_MAX} generate an error.
14743 PIO	_POSIX_PRIORITIZED_IO
14744	The implementation supports the Prioritized Input and Output option. If this symbol has a
14745	value other than -1, it shall have the value 200ymmL, the date of approval of
14746	IEEE Std. 1003.1-200x.
14747 PS	_POSIX_PRIORITY_SCHEDULING
14748	The implementation supports the Process Scheduling option. If this symbol has a value
14749	other than -1, it shall have the value 200ymmL, the date of approval of
14750	IEEE Std. 1003.1-200x.
14751 THR	_POSIX_READER_WRITER_LOCKS
14752	The implementation supports the Read-Write Locks option. This is always set to a value
14753	greater than zero if the Threads option is supported. If this symbol has a value other than
14754	-1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.
14755 RTS	_POSIX_REALTIME_SIGNALS
14756	The implementation supports the Realtime Signals Extension option. If this symbol has a
14757	value other than -1, it shall have the value 200ymmL, the date of approval of
14758	IEEE Std. 1003.1-200x.
14759	_POSIX_REGEX
14760	The implementation supports the Regular Expression Handling option. This is always set

14761	to a value greater than zero.
14762	_POSIX_SAVED_IDS
14763	Each process has a saved set-user-ID and a saved set-group-ID. The behavior of the <i>setuid()</i> ,
14764	<i>setgid()</i> , and <i>kill()</i> functions shall be dependent on the values of the saved set-user-ID and
14765	the saved get-group-ID, respectively. This is always set to a value greater than zero.
14766 SEM	_POSIX_SEMAPHORES
14767	The implementation supports the Semaphores option. If this symbol has a value other than
14768	-1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.
14769 SHM	_POSIX_SHARED_MEMORY_OBJECTS
14770	The implementation supports the Shared Memory Objects option. If this symbol has a value
14771	other than -1, it shall have the value 200ymmL, the date of approval of
14772	IEEE Std. 1003.1-200x.
14773 SH	_POSIX_SHELL
14774	The implementation supports the POSIX shell. This is always set to a value greater than
14775	zero.
14776 SPN	_POSIX_SPAWN
14777	The implementation supports the Spawn option. If this symbol has a value other than -1, it
14778	shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.
14779 SPI	_POSIX_SPIN_LOCKS
14780	The implementation supports the Spin Locks option. If this symbol has a value other than
14781	-1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.
14782 SS	_POSIX_SPORADIC_SERVER
14783	The implementation supports the Process Sporadic Server option. If this symbol has a value
14784	other than -1, it shall have the value 200ymmL, the date of approval of
14785	IEEE Std. 1003.1-200x.
14786 SIO	_POSIX_SYNCHRONIZED_IO
14787	The implementation supports the Synchronized Input and Output option. If this symbol
14788	has a value other than -1, it shall have the value 200ymmL, the date of approval of
14789	IEEE Std. 1003.1-200x.
14790 TSA	_POSIX_THREAD_ATTR_STACKADDR
14791	The implementation supports the Thread Stack Address Attribute option. If this symbol
14792	has a value other than -1, it shall have the value 200ymmL, the date of approval of
14793	IEEE Std. 1003.1-200x.
14794 TSS	_POSIX_THREAD_ATTR_STACKSIZE
14795	The implementation supports the Thread Stack Address Size option. If this symbol has a
14796	value other than -1, it shall have the value 200ymmL, the date of approval of
14797	IEEE Std. 1003.1-200x.
14798 TCT	_POSIX_THREAD_CPUTIME
14799	The implementation supports the Thread CPU-Time Clocks option. If this symbol has a
14800	value other than -1, it shall have the value 200ymmL, the date of approval of
14801	IEEE Std. 1003.1-200x.
14802 TPI	_POSIX_THREAD_PRIO_INHERIT
14803	The implementation supports the Threads Priority Inheritance option. If this symbol has a
14804	value other than -1, it shall have the value 200ymmL, the date of approval of
14805	IEEE Std. 1003.1-200x.

14806	TPP	_POSIX_THREAD_PRIO_PROTECT
14807		The implementation supports the Thread Priority Protection option. If this symbol has a
14808		value other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14809		IEEE Std. 1003.1-200x.
14810	TPS	_POSIX_THREAD_PRIORITY_SCHEDULING
14811		The implementation supports the Thread Execution Scheduling option. If this symbol has a
14812		value other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14813		IEEE Std. 1003.1-200x.
14814	TSH	_POSIX_THREAD_PROCESS_SHARED
14815		The implementation supports the Thread Process-Shared Synchronization option. If this
14816		symbol has a value other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14817		IEEE Std. 1003.1-200x.
14818	TSF	_POSIX_THREAD_SAFE_FUNCTIONS
14819		The implementation supports the Thread-Safe Functions option. If this symbol has a value
14820		other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14821		IEEE Std. 1003.1-200x.
14822	TSP	_POSIX_THREAD_SPORADIC_SERVER
14823		The implementation supports the Thread Sporadic Server option. If this symbol has a value
14824		other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14825		IEEE Std. 1003.1-200x.
14826	THR	_POSIX_THREADS
14827		The implementation supports the Threads option. If this symbol has a value other than <code>-1</code> , it
14828		shall have the value <code>200ymmL</code> , the date of approval of IEEE Std. 1003.1-200x.
14829	TMR	_POSIX_TIMERS
14830		The implementation supports the Timers option. If this symbol has a value other than <code>-1</code> , it
14831		shall have the value <code>200ymmL</code> , the date of approval of IEEE Std. 1003.1-200x.
14832	TMO	_POSIX_TIMEOUTS
14833		The implementation supports the Timeouts option. If this symbol has a value other than <code>-1</code> ,
14834		it shall have the value <code>200ymmL</code> , the date of approval of IEEE Std. 1003.1-200x.
14835	TRC	_POSIX_TRACE
14836		The implementation supports the Trace option.
14837	TEF	_POSIX_TRACE_EVENT_FILTER
14838		The implementation supports the Trace Event Filter option.
14839	TRL	_POSIX_TRACE_LOG
14840		The implementation supports the Trace Log option.
14841	TRI	_POSIX_TRACE_INHERIT
14842		The implementation supports the Trace Inherit option.
14843	TYM	_POSIX_TYPED_MEMORY_OBJECTS
14844		The implementation supports the Typed Memory Objects option. If this symbol has a value
14845		other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14846		IEEE Std. 1003.1-200x.
14847		_POSIX_VDISABLE
14848		Terminal special characters defined in <termios.h> can be disabled using this character
14849		value.

14850 _POSIX2_C_BIND
14851 The implementation supports the C-Language Binding option. This always has the value
14852 200ymmL, the date of approval of IEEE Std. 1003.1-200x.

14853 CD _POSIX2_C_DEV
14854 The implementation supports the C-Language Development Utilities option. If this symbol
14855 has a value other than -1, it shall have the value 200ymmL, the date of approval of
14856 IEEE Std. 1003.1-200x.

14857 _POSIX2_CHAR_TERM
14858 The implementation supports at least one terminal type.

14859 FD _POSIX2_FORT_DEV
14860 The implementation supports the FORTRAN Development Utilities option. If this symbol
14861 has a value other than -1, it shall have the value 200ymmL, the date of approval of
14862 IEEE Std. 1003.1-200x.

14863 FR _POSIX2_FORT_RUN
14864 The implementation supports the FORTRAN Runtime Utilities option. If this symbol has a
14865 value other than -1, it shall have the value 200ymmL, the date of approval of
14866 IEEE Std. 1003.1-200x.

14867 _POSIX2_LOCALEDEF
14868 The implementation supports the creation of locales by the *localedef* utility. If this symbol
14869 has a value other than -1, it shall have the value 200ymmL, the date of approval of
14870 IEEE Std. 1003.1-200x.

14871 BE _POSIX2_PBS
14872 The implementation supports the Batch Environment Services and Utilities option. If this
14873 symbol has a value other than -1, it shall have the value 200ymmL, the date of approval of
14874 IEEE Std. 1003.1-200x.

14875 BE _POSIX2_PBS_ACCOUNTING
14876 The implementation supports the Batch Accounting option. If this symbol has a value other
14877 than -1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.

14878 BE _POSIX2_PBS_CHECKPOINT
14879 The implementation supports the Batch Checkpoint/Restart option. If this symbol has a
14880 value other than -1, it shall have the value 200ymmL, the date of approval of
14881 IEEE Std. 1003.1-200x.

14882 BE _POSIX2_PBS_LOCATE
14883 The implementation supports the Locate Batch Job Request option. If this symbol has a
14884 value other than -1, it shall have the value 200ymmL, the date of approval of
14885 IEEE Std. 1003.1-200x.

14886 BE _POSIX2_PBS_MESSAGE
14887 The implementation supports the Batch Job Message Request option. If this symbol has a
14888 value other than -1, it shall have the value 200ymmL, the date of approval of
14889 IEEE Std. 1003.1-200x.

14890 BE _POSIX2_PBS_TRACK
14891 The implementation supports the Track Batch Job Request option. If this symbol has a value
14892 other than -1, it shall have the value 200ymmL, the date of approval of
14893 IEEE Std. 1003.1-200x.

14894 SD _POSIX2_SW_DEV
14895 The implementation supports the Software Development Utilities option. If this symbol has

14896	a value other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14897	IEEE Std. 1003.1-200x.
14898 UP	_POSIX2_UPE
14899	The implementation supports the User Portability Utilities option. If this symbol has a value
14900	other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14901	IEEE Std. 1003.1-200x.
14902	_V6_ILP32_OFF32
14903	The implementation provides a C-language compilation environment with 32-bit int , long ,
14904	pointer , and off_t types.
14905	_V6_ILP32_OFFBIG
14906	The implementation provides a C-language compilation environment with 32-bit int , long ,
14907	and pointer types and an off_t type using at least 64 bits.
14908	_V6_LP64_OFF64
14909	The implementation provides a C-language compilation environment with 32-bit int and
14910	64-bit long , pointer , and off_t types.
14911	_V6_LPBIG_OFFBIG
14912	The implementation provides a C-language compilation environment with an int type
14913	using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14914 XSI	_XBS5_ILP32_OFF32 (LEGACY)
14915	The implementation provides a C-language compilation environment with 32-bit int , long ,
14916	pointer , and off_t types.
14917 XSI	_XBS5_ILP32_OFFBIG (LEGACY)
14918	The implementation provides a C-language compilation environment with 32-bit int , long ,
14919	and pointer types and an off_t type using at least 64 bits.
14920 XSI	_XBS5_LP64_OFF64 (LEGACY)
14921	The implementation provides a C-language compilation environment with 32-bit int and
14922	64-bit long , pointer , and off_t types.
14923 XSI	_XBS5_LPBIG_OFFBIG (LEGACY)
14924	The implementation provides a C-language compilation environment with an int type
14925	using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14926 XSI	_XOPEN_CRYPT
14927	The implementation supports the X/Open Encryption Option Group.
14928	_XOPEN_ENH_I18N
14929	The implementation supports the Issue 4, Version 2 Enhanced Internationalization Option
14930	Group. This is always set to a value other than <code>-1</code> .
14931	_XOPEN_LEGACY
14932	The implementation supports the Legacy Option Group.
14933	_XOPEN_REALTIME
14934	The implementation supports the X/Open Realtime Option Group.
14935	_XOPEN_REALTIME_THREADS
14936	The implementation supports the X/Open Realtime Threads Option Group.
14937	_XOPEN_SHM
14938	The implementation supports the Issue 4, Version 2 Shared Memory Option Group. This is
14939	always set to a value other than <code>-1</code> .

14940 **XOPEN_STREAMS**
14941 The implementation supports the XSI STREAMS Option Group.

14942 **Constants for Profiling Option Groups**

14943 The following symbolic constants shall be defined to have the value -1 if the implementation
14944 never provides the Profiling Option Group, and to have a value other than -1 if the
14945 implementation always provides the Profiling Option Group. If these are undefined, the
14946 *sysconf()* function can be used to determine whether the Profiling Option Group is provided for
14947 a particular invocation of the application.

14948 For conformance requirements, refer to Chapter 2 (on page 19).

- 14949 • `_POSIX_BASE`
- 14950 • `_POSIX_C_LANG_SUPPORT`
- 14951 • `_POSIX_C_LANG_SUPPORT_R`
- 14952 • `_POSIX_DEVICE_IO`
- 14953 • `_POSIX_DEVICE_SPECIFIC`
- 14954 • `_POSIX_DEVICE_SPECIFIC_R`
- 14955 • `_POSIX_FD_MGMT`
- 14956 • `_POSIX_FIFO`
- 14957 • `_POSIX_FILE_ATTRIBUTES`
- 14958 • `_POSIX_FILE_LOCKING`
- 14959 • `_POSIX_FILE_SYSTEM`
- 14960 • `_POSIX_JOB_CONTROL`
- 14961 • `_POSIX_MULTIPLE_PROCESS`
- 14962 • `_POSIX_NETWORKING`
- 14963 • `_POSIX_PIPE`
- 14964 • `_POSIX_SIGNALS`
- 14965 • `_POSIX_SINGLE_PROCESS`
- 14966 • `_POSIX_SYSTEM_DATABASE`
- 14967 • `_POSIX_SYSTEM_DATABASE_R`
- 14968 • `_POSIX_USER_GROUPS`
- 14969 • `_POSIX_USER_GROUPS_R`

14970 **Execution-Time Symbolic Constants**

14971 If any of the following constants are not defined in the <unistd.h> header, the value shall vary
14972 depending on the file to which it is applied.

14973 If any of the following constants are defined to have value -1 in the <unistd.h> header, the
14974 implementation shall not provide the option on any file; if any are defined to have a value other
14975 than -1 in the <unistd.h> header, the implementation shall provide the option on all applicable
14976 files.

14977 All of the following constants, whether defined in <unistd.h> or not, may be queried with
 14978 respect to a specific file using the *pathconf()* or *fpathconf()* functions:

14979 `_POSIX_ASYNC_IO`

14980 Asynchronous input or output operations may be performed for the associated file.

14981 `_POSIX_PRIO_IO`

14982 Prioritized input or output operations may be performed for the associated file.

14983 `_POSIX_SYNC_IO`

14984 Synchronized input or output operations may be performed for the associated file.

14985 **Constants for Functions**

14986 The following symbolic constant shall be defined:

14987 `NULL` Null pointer

14988 The following symbolic constants shall be defined for the *access()* function:

14989 `F_OK` Test for existence of file.

14990 `R_OK` Test for read permission.

14991 `W_OK` Test for write permission.

14992 `X_OK` Test for execute (search) permission.

14993 The constants `F_OK`, `R_OK`, `W_OK`, and `X_OK` and the expressions `R_OK | W_OK`, `R_OK | X_OK`,
 14994 and `R_OK | W_OK | X_OK` shall all have distinct values.

14995 The following symbolic constants shall be defined for the *confstr()* function:

14996 `_CS_PATH`

14997 This is the value for the *PATH* environment variable that finds all standard utilities.

14998 `_CS_V6_ILP32_OFF32_CFLAGS`

14999 If *sysconf*(`_SC_V6_ILP32_OFF32`) returns `-1`, the meaning of this value is unspecified.
 15000 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
 15001 build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t**
 15002 types.

15003 `_CS_V6_ILP32_OFF32_LDFLAGS`

15004 If *sysconf*(`_SC_V6_ILP32_OFF32`) returns `-1`, the meaning of this value is unspecified.
 15005 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
 15006 an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15007 `_CS_V6_ILP32_OFF32_LIBS`

15008 If *sysconf*(`_SC_V6_ILP32_OFF32`) returns `-1`, the meaning of this value is unspecified.
 15009 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
 15010 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15011 `_CS_V6_ILP32_OFF32_LINTFLAGS`

15012 If *sysconf*(`_SC_V6_ILP32_OFF32`) returns `-1`, the meaning of this value is unspecified.
 15013 Otherwise, this value is the set of options to be given to the *lint* utility to check application
 15014 source using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15015 `_CS_V6_ILP32_OFFBIG_CFLAGS`

15016 If *sysconf*(`_SC_V6_ILP32_OFFBIG`) returns `-1`, the meaning of this value is unspecified.
 15017 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
 15018 build an application using a programming model with 32-bit **int**, **long**, and **pointer** types,

15019 and an **off_t** type using at least 64 bits.

15020 _CS_V6_ILP32_OFFBIG_LDFLAGS
15021 If *sysconf*(_SC_V6_ILP32_OFFBIG) returns -1, the meaning of this value is unspecified.
15022 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
15023 an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15024 **off_t** type using at least 64 bits.

15025 _CS_V6_ILP32_OFFBIG_LIBS
15026 If *sysconf*(_SC_V6_ILP32_OFFBIG) returns -1, the meaning of this value is unspecified.
15027 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
15028 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15029 **off_t** type using at least 64 bits.

15030 _CS_V6_ILP32_OFFBIG_LINTFLAGS
15031 If *sysconf*(_SC_V6_ILP32_OFFBIG) returns -1, the meaning of this value is unspecified.
15032 Otherwise, this value is the set of options to be given to the *lint* utility to check an
15033 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15034 **off_t** type using at least 64 bits.

15035 _CS_V6_LP64_OFF64_CFLAGS
15036 If *sysconf*(_SC_V6_LP64_OFF64) returns -1, the meaning of this value is unspecified.
15037 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
15038 build an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t**
15039 types.

15040 _CS_V6_LP64_OFF64_LDFLAGS
15041 If *sysconf*(_SC_V6_LP64_OFF64) returns -1, the meaning of this value is unspecified.
15042 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
15043 an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

15044 _CS_V6_LP64_OFF64_LIBS
15045 If *sysconf*(_SC_V6_LP64_OFF64) returns -1, the meaning of this value is unspecified.
15046 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
15047 application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

15048 _CS_V6_LP64_OFF64_LINTFLAGS
15049 If *sysconf*(_SC_V6_LP64_OFF64) returns -1, the meaning of this value is unspecified.
15050 Otherwise, this value is the set of options to be given to the *lint* utility to check application
15051 source using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

15052 _CS_V6_LPBIG_OFFBIG_CFLAGS
15053 If *sysconf*(_SC_V6_LPBIG_OFFBIG) returns -1, the meaning of this value is unspecified.
15054 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
15055 build an application using a programming model with an **int** type using at least 32 bits and
15056 **long**, **pointer**, and **off_t** types using at least 64 bits.

15057 _CS_V6_LPBIG_OFFBIG_LDFLAGS
15058 If *sysconf*(_SC_V6_LPBIG_OFFBIG) returns -1, the meaning of this value is unspecified.
15059 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
15060 an application using a programming model with an **int** type using at least 32 bits and **long**,
15061 **pointer**, and **off_t** types using at least 64 bits.

15062 _CS_V6_LPBIG_OFFBIG_LIBS
15063 If *sysconf*(_SC_V6_LPBIG_OFFBIG) returns -1, the meaning of this value is unspecified.
15064 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
15065 application using a programming model with an **int** type using at least 32 bits and **long**,

15066 **pointer**, and **off_t** types using at least 64 bits.

15067 `_CS_V6_LPBIG_OFFBIG_LINTFLAGS`
15068 If `sysconf(_SC_V6_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15069 Otherwise, this value is the set of options to be given to the `lint` utility to check application
15070 source using a programming model with an **int** type using at least 32 bits and **long**, **pointer**,
15071 and **off_t** types using at least 64 bits.

15072 XSI `_CS_XBS5_ILP32_OFF32_CFLAGS (LEGACY)`
15073 If `sysconf(_SC_XBS5_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
15074 Otherwise, this value is the set of initial options to be given to the `cc` and `c99` utilities to
15075 build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t**
15076 types.

15077 XSI `_CS_XBS5_ILP32_OFF32_LDFLAGS (LEGACY)`
15078 If `sysconf(_SC_XBS5_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
15079 Otherwise, this value is the set of final options to be given to the `cc` and `c99` utilities to build
15080 an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15081 XSI `_CS_XBS5_ILP32_OFF32_LIBS (LEGACY)`
15082 If `sysconf(_SC_XBS5_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
15083 Otherwise, this value is the set of libraries to be given to the `cc` and `c99` utilities to build an
15084 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15085 XSI `_CS_XBS5_ILP32_OFF32_LINTFLAGS (LEGACY)`
15086 If `sysconf(_SC_XBS5_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
15087 Otherwise, this value is the set of options to be given to the `lint` utility to check application
15088 source using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15089 XSI `_CS_XBS5_ILP32_OFFBIG_CFLAGS (LEGACY)`
15090 If `sysconf(_SC_XBS5_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15091 Otherwise, this value is the set of initial options to be given to the `cc` and `c99` utilities to
15092 build an application using a programming model with 32-bit **int**, **long**, and **pointer** types,
15093 and an **off_t** type using at least 64 bits.

15094 XSI `_CS_XBS5_ILP32_OFFBIG_LDFLAGS (LEGACY)`
15095 If `sysconf(_SC_XBS5_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15096 Otherwise, this value is the set of final options to be given to the `cc` and `c99` utilities to build
15097 an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15098 **off_t** type using at least 64 bits.

15099 XSI `_CS_XBS5_ILP32_OFFBIG_LIBS (LEGACY)`
15100 If `sysconf(_SC_XBS5_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15101 Otherwise, this value is the set of libraries to be given to the `cc` and `c99` utilities to build an
15102 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15103 **off_t** type using at least 64 bits.

15104 XSI `_CS_XBS5_ILP32_OFFBIG_LINTFLAGS (LEGACY)`
15105 If `sysconf(_SC_XBS5_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15106 Otherwise, this value is the set of options to be given to the `lint` utility to check an
15107 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15108 **off_t** type using at least 64 bits.

15109 XSI `_CS_XBS5_LP64_OFF64_CFLAGS (LEGACY)`
15110 If `sysconf(_SC_XBS5_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
15111 Otherwise, this value is the set of initial options to be given to the `cc` and `c99` utilities to
15112 build an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t**

15113 types.

15114 XSI **_CS_XBS5_LP64_OFF64_LDFLAGS (LEGACY)**
 15115 If *sysconf*(_SC_XBS5_LP64_OFF64) returns -1, the meaning of this value is unspecified.
 15116 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
 15117 an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

15118 XSI **_CS_XBS5_LP64_OFF64_LIBS (LEGACY)**
 15119 If *sysconf*(_SC_XBS5_LP64_OFF64) returns -1, the meaning of this value is unspecified.
 15120 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
 15121 application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

15122 XSI **_CS_XBS5_LP64_OFF64_LINTFLAGS (LEGACY)**
 15123 If *sysconf*(_SC_XBS5_LP64_OFF64) returns -1, the meaning of this value is unspecified.
 15124 Otherwise, this value is the set of options to be given to the *lint* utility to check application
 15125 source using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

15126 XSI **_CS_XBS5_LPBIG_OFFBIG_CFLAGS (LEGACY)**
 15127 If *sysconf*(_SC_XBS5_LPBIG_OFFBIG) returns -1, the meaning of this value is unspecified.
 15128 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
 15129 build an application using a programming model with an **int** type using at least 32 bits and
 15130 **long**, **pointer**, and **off_t** types using at least 64 bits.

15131 XSI **_CS_XBS5_LPBIG_OFFBIG_LDFLAGS (LEGACY)**
 15132 If *sysconf*(_SC_XBS5_LPBIG_OFFBIG) returns -1, the meaning of this value is unspecified.
 15133 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
 15134 an application using a programming model with an **int** type using at least 32 bits and **long**,
 15135 **pointer**, and **off_t** types using at least 64 bits.

15136 XSI **_CS_XBS5_LPBIG_OFFBIG_LIBS (LEGACY)**
 15137 If *sysconf*(_SC_XBS5_LPBIG_OFFBIG) returns -1, the meaning of this value is unspecified.
 15138 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
 15139 application using a programming model with an **int** type using at least 32 bits and **long**,
 15140 **pointer**, and **off_t** types using at least 64 bits.

15141 XSI **_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS (LEGACY)**
 15142 If *sysconf*(_SC_XBS5_LPBIG_OFFBIG) returns -1, the meaning of this value is unspecified.
 15143 Otherwise, this value is the set of options to be given to the *lint* utility to check application
 15144 source using a programming model with an **int** type using at least 32 bits and **long**, **pointer**,
 15145 and **off_t** types using at least 64 bits.

15146 The following symbolic constants shall be defined for the *lseek*() and *fcntl*() functions (they have
 15147 distinct values):

15148 {SEEK_CUR} Set file offset to current plus *offset*.
 15149 {SEEK_END} Set file offset to EOF plus *offset*.
 15150 {SEEK_SET} Set file offset to *offset*.

15151 The following symbolic constants shall be defined for *sysconf*():

15152 **_SC_2_C_BIND**
 15153 **_SC_2_C_DEV**
 15154 **_SC_2_C_VERSION**
 15155 **_SC_2_FORT_DEV**
 15156 **_SC_2_FORT_RUN**
 15157 **_SC_2_LOCALEDEF**
 15158 **_SC_2_PBS**

```

15159     _SC_2_PBS_ACCOUNTING
15160     _SC_2_PBS_CHECKPOINT
15161     _SC_2_PBS_LOCATE
15162     _SC_2_PBS_MESSAGE
15163     _SC_2_PBS_TRACK
15164     _SC_2_SW_DEV
15165     _SC_2_UPE
15166     _SC_2_VERSION
15167     _SC_ARG_MAX
15168     _SC_AIO_LISTIO_MAX
15169     _SC_AIO_MAX
15170     _SC_AIO_PRIO_DELTA_MAX
15171     _SC_ASYNCHRONOUS_IO
15172 XSI   _SC_ATEXIT_MAX
15173 BAR   _SC_BARRIERS
15174     _SC_BASE
15175     _SC_BC_BASE_MAX
15176     _SC_BC_DIM_MAX
15177     _SC_BC_SCALE_MAX
15178     _SC_BC_STRING_MAX
15179     _SC_C_LANG_SUPPORT
15180     _SC_C_LANG_SUPPORT_R
15181     _SC_CHILD_MAX
15182     _SC_CLK_TCK
15183 CS    _SC_CLOCK_SELECTION
15184     _SC_COLL_WEIGHTS_MAX
15185     _SC_DELAYTIMER_MAX
15186     _SC_DEVICE_IO
15187     _SC_DEVICE_SPECIFIC
15188     _SC_DEVICE_SPECIFIC_R
15189     _SC_EXPR_NEST_MAX
15190     _SC_FD_MGMT
15191     _SC_FIFO
15192     _SC_FILE_ATTRIBUTES
15193     _SC_FILE_LOCKING
15194     _SC_FILE_SYSTEM
15195     _SC_FSYNC
15196     _SC_GETGR_R_SIZE_MAX
15197     _SC_GETPW_R_SIZE_MAX
15198 XSI   _SC_IOV_MAX
15199     _SC_JOB_CONTROL
15200     _SC_LINE_MAX
15201     _SC_LOGIN_NAME_MAX
15202     _SC_MAPPED_FILES
15203     _SC_MEMLOCK
15204     _SC_MEMLOCK_RANGE
15205     _SC_MEMORY_PROTECTION
15206     _SC_MESSAGE_PASSING
15207 MON   _SC_MONOTONIC_CLOCK
15208     _SC_MQ_OPEN_MAX
15209     _SC_MQ_PRIO_MAX
15210     _SC_MULTIPLE_PROCESS

```

15211		_SC_NETWORKING
15212		_SC_NGROUPS_MAX
15213		_SC_OPEN_MAX
15214	XSI	_SC_PAGE_SIZE
15215		_SC_PAGESIZE
15216		_SC_PIPE
15217		_SC_PRIORITIZED_IO
15218		_SC_PRIORITY_SCHEDULING
15219		_SC_RE_DUP_MAX
15220	THR	_SC_READER_WRITER_LOCKS
15221		_SC_REALTIME_SIGNALS
15222		_SC_REGEX
15223		_SC_RTSIG_MAX
15224		_SC_SAVED_IDS
15225		_SC_SEMAPHORES
15226		_SC_SEM_NSEMS_MAX
15227		_SC_SEM_VALUE_MAX
15228		_SC_SHARED_MEMORY_OBJECTS
15229		_SC_SHELL
15230		_SC_SIGNALS
15231		_SC_SIGQUEUE_MAX
15232		_SC_SINGLE_PROCESS
15233	SPI	_SC_SPIN_LOCKS
15234		_SC_STREAM_MAX
15235		_SC_SYNCHRONIZED_IO
15236		_SC_SYSTEM_DATABASE
15237		_SC_SYSTEM_DATABASE_R
15238		_SC_THREAD_ATTR_STACKADDR
15239		_SC_THREAD_ATTR_STACKSIZE
15240		_SC_THREAD_DESTRUCTOR_ITERATIONS
15241		_SC_THREAD_KEYS_MAX
15242		_SC_THREAD_PRIO_INHERIT
15243		_SC_THREAD_PRIO_PROTECT
15244		_SC_THREAD_PRIORITY_SCHEDULING
15245		_SC_THREAD_PROCESS_SHARED
15246		_SC_THREAD_SAFE_FUNCTIONS
15247		_SC_THREAD_STACK_MIN
15248		_SC_THREAD_THREADS_MAX
15249		_SC_THREADS
15250		_SC_TIMER_MAX
15251		_SC_TIMERS
15252	TRC	_SC_TRACE
15253	TEF	_SC_TRACE_EVENT_FILTER
15254	TRL	_SC_TRACE_LOG
15255	TRI	_SC_TRACE_INHERIT
15256		_SC_TTY_NAME_MAX
15257	TYM	_SC_TYPED_MEMORY_OBJECTS
15258		_SC_TZNAME_MAX
15259		_SC_USER_GROUPS
15260		_SC_USER_GROUPS_R
15261		_SC_V6_ILP32_OFF32
15262		_SC_V6_ILP32_OFFBIG

```

15263     _SC_V6_LP64_OFF64
15264     _SC_V6_LP64_OFF64
15265     _SC_VERSION
15266 XSI   _SC_XBS5_ILP32_OFF32 (LEGACY)
15267     _SC_XBS5_ILP32_OFFBIG (LEGACY)
15268     _SC_XBS5_LP64_OFF64 (LEGACY)
15269     _SC_XBS5_LP64_OFF64 (LEGACY)
15270     _SC_XOPEN_CRYPT
15271     _SC_XOPEN_ENH_I18N
15272     _SC_XOPEN_LEGACY
15273     _SC_XOPEN_REALTIME
15274     _SC_XOPEN_REALTIME_THREADS
15275     _SC_XOPEN_SHM
15276     _SC_XOPEN_STREAMS
15277     _SC_XOPEN_UNIX
15278     _SC_XOPEN_VERSION
15279     _SC_XOPEN_XCU_VERSION
15280

```

15281 The two constants `_SC_PAGESIZE` and `_SC_PAGE_SIZE` may be defined to have the same
15282 value.

15283 The following symbolic constants shall be defined as possible values for the *function* argument
15284 to the `lockf()` function:

```

15285     F_LOCK           Lock a section for exclusive use.
15286     F_TEST           Test section for locks by other processes.
15287     F_TLOCK         Test and lock a section for exclusive use.
15288     F_ULOCK         Unlock locked sections.

```

15289 The following symbolic constants shall be defined for `pathconf()`:

```

15290 ADV   _PC_ALLOC_SIZE_MIN
15291 AIO   _PC_ASYNC_IO
15292     _PC_CHOWN_RESTRICTED
15293     _PC_FILESIZEBITS
15294     _PC_LINK_MAX
15295     _PC_MAX_CANON
15296     _PC_MAX_INPUT
15297     _PC_NAME_MAX
15298     _PC_NO_TRUNC
15299     _PC_PATH_MAX
15300     _PC_PIPE_BUF
15301     _PC_PRIO_IO
15302 ADV   _PC_REC_INCR_XFER_SIZE
15303     _PC_REC_MAX_XFER_SIZE
15304     _PC_REC_MIN_XFER_SIZE
15305     _PC_REC_XFER_ALIGN
15306     _PC_SYNC_IO
15307     _PC_VDISABLE

```

15308 The following symbolic constants shall be defined for file streams:

15309 `STDERR_FILENO` File number of *stderr*; 2.
 15310 `STDIN_FILENO` File number of *stdin*; 0.
 15311 `STDOUT_FILENO` File number of *stdout*; 1.

15312 **Type Definitions**

15313 The `size_t`, `ssize_t`, `uid_t`, `gid_t`, `off_t`, and `pid_t` types shall be defined as described in
 15314 `<sys/types.h>`.

15315 The `useconds_t` type shall be defined as described in `<sys/types.h>`.

15316 The `intptr_t` type shall be defined as described in `<inttypes.h>`.

15317 The `socklen_t` type shall be defined as described in `<sys/socket.h>`.

15318 **Declarations**

15319 The following shall be declared as functions and may also be defined as macros. Function
 15320 prototypes shall be provided for use with an ISO C standard compiler.

```

15321 int      access(const char *, int);
15322 unsigned alarm(unsigned);
15323 XSI int    brk(void *);
15324 int      chdir(const char *);
15325 int      chown(const char *, uid_t, gid_t);
15326 int      close(int);
15327 size_t   confstr(int, char *, size_t);
15328 XSI char *crypt(const char *, const char *);
15329 char *ctermid(char *);
15330 int      dup(int);
15331 int      dup2(int, int);
15332 XSI void  encrypt(char[64], int);
15333 int      execl(const char *, const char *, ...);
15334 int      execlp(const char *, const char *, ...);
15335 int      execlp(const char *, const char *, ...);
15336 int      execv(const char *, char *const []);
15337 int      execve(const char *, char *const [], char *const []);
15338 int      execvp(const char *, char *const []);
15339 void     _exit(int);
15340 XSI int   fchown(int, uid_t, gid_t);
15341 int      fchdir(int);
15342 SIO int   fdatsync(int);
15343 pid_t    fork(void);
15344 long     fpathconf(int, int);
15345 int      fsync(int);
15346 int      ftruncate(int, off_t);
15347 char *getcwd(char *, size_t);
15348 gid_t    getegid(void);
15349 uid_t    geteuid(void);
15350 gid_t    getgid(void);
15351 int      getgroups(int, gid_t []);
15352 XSI long  gethostid(void);
15353 int      gethostname(char *, socklen_t);

```



```

15354     char        *getlogin(void);
15355     int         getlogin_r(char *, size_t);
15356     int         getopt(int, char * const [], const char *);
15357 XSI    pid_t      getpgid(pid_t);
15358     pid_t      getpgrp(void);
15359     pid_t      getpid(void);
15360     pid_t      getppid(void);
15361 XSI    pid_t      getsid(pid_t);
15362     uid_t      getuid(void);
15363 XSI    char        *getwd(char *); (LEGACY)
15364     int         isatty(int);
15365 XSI    int         lchown(const char *, uid_t, gid_t);
15366     int         link(const char *, const char *);
15367 XSI    int         lockf(int, int, off_t);
15368     off_t      lseek(int, off_t, int);
15369 XSI    int         nice(int);
15370     long        pathconf(const char *, int);
15371     int         pause(void);
15372     int         pipe(int [2]);
15373 XSI    ssize_t     pread(int, void *, size_t, off_t);
15374     ssize_t     pwrite(int, const void *, size_t, off_t);
15375     ssize_t     read(int, void *, size_t);
15376     ssize_t     readlink(const char *restrict, char *restrict, size_t);
15377     int         rmdir(const char *);
15378 XSI    void        *sbrk(intptr_t);
15379     int         setegid(gid_t);
15380     int         seteuid(uid_t);
15381     int         setgid(gid_t);
15382     int         setpgid(pid_t, pid_t);
15383 XSI    pid_t      setpgrp(void);
15384     int         setregid(gid_t, gid_t);
15385     int         setreuid(uid_t, uid_t);
15386     pid_t      setsid(void);
15387     int         setuid(uid_t);
15388     unsigned    sleep(unsigned);
15389 XSI    void        swab(const void *restrict, void *restrict, ssize_t);
15390     int         symlink(const char *, const char *);
15391     void        sync(void);
15392     long        sysconf(int);
15393     pid_t      tcgetpgrp(int);
15394     int         tcsetpgrp(int, pid_t);
15395 XSI    int         truncate(const char *, off_t);
15396     char        *ttyname(int);
15397     int         ttyname_r(int, char *, size_t);
15398 XSI    useconds_t  ualarm(useconds_t, useconds_t);
15399     int         unlink(const char *);
15400 XSI    int         usleep(useconds_t);
15401     pid_t      vfork(void);
15402     ssize_t     write(int, const void *, size_t);

15403     Implementations may also include the pthread_atfork() prototype as defined in <pthread.h> (on
15404     page 322).

```

15405 The following external variables shall be declared:

```
15406 extern char *optarg;  
15407 extern int optind, opterr, optopt;
```

15408 **APPLICATION USAGE**

15409 None.

15410 **RATIONALE**

15411 As IEEE Std. 1003.1-200x evolved, certain options became sufficiently standardized that it was
15412 concluded that simply requiring one of the option choices was simpler than retaining the option.
15413 However, for backwards compatibility, the option flags (with required constant values) are
15414 retained.

15415 **Version Test Macros**

15416 The standard developers considered altering the definition of `_POSIX_VERSION` and removing
15417 `_SC_VERSION` from the specification of `sysconf()` since the utility to an application was deemed
15418 by some to be minimal, and since the implementation of the functionality is potentially
15419 problematic.

15420 Applications are allowed the ability to adapt to various versions of IEEE Std. 1003.1-200x at
15421 compile time by conditionally compiling different code depending on the value of
15422 `_POSIX_VERSION`. For example, an application which expects the semantics of the
15423 POSIX.1-1988 standard `cuserid()` but also wishes to compile and run on a system which
15424 conforms to the POSIX.1-1990 standard might be coded as in the following program fragment:

```
15425 #if _POSIX_VERSION == 198808L  
15426 val = cuserid();  
15427 #else  
15428 {  
15429 struct passwd *pwp;  
15430 pwp = getpwuid(geteuid());  
15431 val = pwp->pw_name;  
15432 }  
15433 #endif
```

15434 While POSIX does not make any attempt to define application binary interaction with the
15435 underlying operating system, the standard developers recognized that support for existing
15436 application binaries is a concern to manufacturers, application developers, and the users of
15437 implementations conforming to IEEE Std. 1003.1-200x. To that end, an application can query
15438 `_SC_VERSION` at runtime via `sysconf()` to determine whether the current version of the
15439 operating system supports the necessary functionality as in the following program fragment:

```
15440 if(sysconf(_SC_VERSION) != 200xxxL) {  
15441 fprintf(stderr, "POSIX.1-1990 system required, terminating\n")  
15442 exit(1);  
15443 }
```

15444 While the above example does not provide the greatest degree of imaginable utility to the
15445 application developer or user, it is arguably better than a core dump or some other equally
15446 obscure result. (It is also possible for implementations to encode and recognize application
15447 binaries compiled in various POSIX.1-conforming environments, and modify the semantics of
15448 the underlying system to conform to the expectations of the application.) For the reasons
15449 outlined in the preceding paragraphs, the standard developers elected to retain the
15450 `_POSIX_VERSION` and `_SC_VERSION` functionality.

15451 **Compile-Time Symbolic Constants for System-Wide Options**

15452 IEEE Std. 1003.1-200x now includes support in certain areas for the newly adopted policy
15453 governing options and stubs.

15454 This policy provides flexibility for implementations in how they support options. It also
15455 specifies how conforming applications can adapt to different implementations that support
15456 different sets of options. It allows the following:

- 15457 1. If an implementation has no interest in supporting an option, it does not have to provide
15458 anything associated with that option beyond the announcement that it does not support it.
- 15459 2. An implementation can support a partial or incompatible version of an option (as a non-
15460 standard extension) as long as it does not claim to support the option.
- 15461 3. An application can determine whether the option is supported. A strictly conforming
15462 application must check this announcement mechanism before first using anything
15463 associated with the option.

15464 There is an important implication of this policy. IEEE Std. 1003.1-200x cannot dictate the
15465 behavior of interfaces associated with an option when the implementation does not claim to
15466 support the option. In particular, it cannot require that a function associated with an
15467 unsupported option will fail if it does not perform as specified. However, this policy does not
15468 prevent a standard from requiring certain functions to always be present, but that they shall
15469 always fail on some implementations. The *setpgid()* function in the POSIX.1-1990 standard, for
15470 example, is considered appropriate.

15471 The POSIX standards include various options, and the C language binding support for an option
15472 implies that the implementation must supply data types and function interfaces. An application
15473 must be able to discover whether the implementation supports each option.

15474 Any application must consider the following three cases for each option:

- 15475 1. Option never supported.
15476 The implementation advertises at compile time that the option will never be supported. In
15477 this case, it is not necessary for the implementation to supply any of the data types or
15478 function interfaces that are provided only as part of the option. The implementation might
15479 provide data types and functions that are similar to those defined by IEEE Std. 1003.1-200x,
15480 but there is no guarantee for any particular behavior.
- 15481 2. Option always supported.
15482 The implementation advertises at compile time that the option will always be supported.
15483 In this case, all data types and function interfaces shall be available and shall operate as
15484 specified.
- 15485 3. Option might or might not be supported.
15486 Some implementations might not provide a mechanism to specify support of options at
15487 compile time. In addition, the implementation might be unable or unwilling to specify
15488 support or non-support at compile time. In either case, any application that might use the
15489 option at runtime must be able to compile and execute. The implementation must provide,
15490 at compile time, all data types and function interfaces that are necessary to allow this. In
15491 this situation, there must be a mechanism that allows the application to query, at runtime,
15492 whether the option is supported. If the application attempts to use the option when it is
15493 not supported, the result is unspecified unless explicitly specified otherwise in
15494 IEEE Std. 1003.1-200x.

15495 **FUTURE DIRECTIONS**

15496 None.

15497 **SEE ALSO**

15498 <inttypes.h>, <limits.h>, <sys/socket.h>, <sys/types.h>, <termios.h>, the System Interfaces
15499 volume of IEEE Std. 1003.1-200x, *access()*, *alarm()*, *chdir()*, *chown()*, *close()*, *crypt()*, *ctermid()*,
15500 *dup()*, *encrypt()*, *environ()*, *exec()*, *exit()*, *fchdir()*, *fchown()*, *fcntl()*, *fork()*, *fpathconf()*, *fsync()*,
15501 *ftruncate()*, *getcwd()*, *getegid()*, *geteuid()*, *getgid()*, *getgroups()*, *gethostid()*, *gethostname()*,
15502 *getlogin()*, *getpgid()*, *getpgrp()*, *getpid()*, *getppid()*, *getsid()*, *getuid()*, *isatty()*, *lchown()*, *link()*,
15503 *lockf()*, *lseek()*, *nice()*, *pathconf()*, *pause()*, *pipe()*, *read()*, *readlink()*, *rmdir()*, *setgid()*, *setpgid()*,
15504 *setpgrp()*, *setregid()*, *setreuid()*, *setsid()*, *setuid()*, *sleep()*, *swab()*, *symlink()*, *sync()*, *sysconf()*,
15505 *tcgetpgrp()*, *tcsetpgrp()*, *truncate()*, *ttyname()*, *ualarm()*, *unlink()*, *usleep()*, *vfork()*, *write()*

15506 **CHANGE HISTORY**

15507 First released in Issue 1. Derived from Issue 1 of the SVID.

15508 **Issue 4**

15509 The symbolic constants F_ULOCK, F_LOCK, F_TLOCK, F_TEST, GF_PATH, IF_PATH, and
15510 PF_PATH are withdrawn.

15511 The required value of _XOPEN_VERSION is defined and the constant is marked as an extension.

15512 The constants _XOPEN_XPG2, _XOPEN_XPG3, and _XOPEN_XPG4 are added.

15513 The constants _POSIX2_* are added.

15514 Reference to the <sys/types.h> header is added for the definitions of **size_t**, **ssize_t**, **uid_t**, **gid_t**,
15515 **off_t**, and **pid_t**. These are marked as extensions.

15516 The names *chroot()*, *crypt()*, *encrypt()*, *fsync()*, *getopt()*, *getpass()*, *nice()*, and *swab()* are added to
15517 the list of functions declared in this header. With the exception of *getopt()*, these are all marked
15518 as extensions.

15519 The APPLICATION USAGE section is removed.

15520 The following changes are incorporated for alignment with the ISO POSIX-1 standard and the
15521 ISO POSIX-2 standard:

- 15522 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 15523 • A large number of new constants are defined for the *sysconf()* function, including all those
- 15524 with prefixes *_SC_2* and *_SC_BC*, plus:

15525 *_SC_COLL_WEIGHTS_MAX*

15526 *_SC_EXPR_NEST_MAX*

15527 *_SC_LINE_MAX*

15528 *_SC_RE_DUP_MAX*

15529 *_SC_STREAM_MAX*

15530 *_SC_TZNAME_MAX*

- 15531 • The *confstr()* function is added to the list of functions declared in this header, complete with
- 15532 a new set of constants for alignment with the ISO POSIX-2 standard.

15533 The following change is incorporated for alignment with the FIPS requirements:

- 15534 • The following symbolic constants are always defined:

15535 _POSIX_CHOWN_RESTRICTED
 15536 _POSIX_NO_TRUNC
 15537 _POSIX_VDISABLE
 15538 _POSIX_SAVED_IDS
 15539 _POSIX_JOB_CONTROL

15540 In Issue 3, they are only defined if the associated option is present.

15541 **Issue 4, Version 2**

15542 The following changes are incorporated for X/OPEN UNIX conformance:

15543 • The Option Group constant `_XOPEN_UNIX` is defined.

15544 • The `sysconf()` symbolic constants `_SC_ATEXIT_MAX`, `_SC_IOV_MAX`, `_SC_PAGESIZE`, and
 15545 `_SC_PAGE_SIZE` are defined.

15546 • The `brk()`, `fchown()`, `fchdir()`, `ftruncate()`, `gethostid()`, `getpagesize()`, `getpgid()`, `getsid()`, `getwd()`,
 15547 `lchown()`, `lockf()`, `readlink()`, `sbrk()`, `setpgrp()`, `setregid()`, `setreuid()`, `symlink()`, `sync()`,
 15548 `truncate()`, `ualarm()`, `usleep()`, and `vfork()` functions are added to the list of functions
 15549 declared in this header.

15550 • The symbolic constants `F_ULOCK`, `F_LOCK`, `F_TLOCK`, and `F_TEST` are added.

15551 **Issue 5**

15552 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 15553 Threads Extension.

15554 The symbolic constants `_XOPEN_REALTIME` and `_XOPEN_REALTIME_THREADS` are added.
 15555 `_POSIX2_C_BIND`, `_XOPEN_ENH_I18N`, and `_XOPEN_SHM` must now be set to a value other
 15556 than `-1` by a conforming implementation.

15557 Large File System extensions are added.

15558 The type of the argument to `sbrk()` is changed from `int` to `intptr_t`.

15559 `_XBS_` constants are added to the list of constants for Options and Option Groups, to the list of
 15560 constants for the `confstr()` function, and to the list of constants to the `sysconf()` function. These
 15561 are all marked EX.

15562 **Issue 6**

15563 `_POSIX2_C_VERSION` is removed.

15564 The Open Group corrigenda item U026/4 has been applied, adding the prototype for `fdatasync()`.

15565 The Open Group corrigenda item U026/1 has been applied, adding the symbols
 15566 `_SC_XOPEN_LEGACY`, `_SC_XOPEN_REALTIME`, and `_SC_XOPEN_REALTIME_THREADS`.

15567 The symbols `_XOPEN_STREAMS` and `_SC_XOPEN_STREAMS` are added to support the XSI
 15568 STREAMS Option Group.

15569 Constants for profiling options are added.

15570 Text in the DESCRIPTION relating to conformance requirements is moved elsewhere in
 15571 IEEE Std. 1003.1-200x.

15572 The legacy symbol `_SC_PASS_MAX` is removed.

15573 The following new requirements on POSIX implementations derive from alignment with the
 15574 Single UNIX Specification:

15575 • The `_CS_XBS5_*` constants are added for the `confstr()` function.

- 15576 • The `_SC_XBS5_*` constants are added for the `sysconf()` function.
- 15577 • The symbolic constants `F_ULOCK`, `F_LOCK`, `F_TLOCK`, and `F_TEST` are added.
- 15578 • The `uid_t`, `gid_t`, `off_t`, `pid_t`, and `useconds_t` types are mandated.
- 15579 The `gethostname()` prototype is added for sockets.
- 15580 New section added for System Wide Options.
- 15581 Function prototypes for `setegid()` and `seteuid()` are added.
- 15582 Option symbolic constants are added for `_POSIX_ADVISORY_INFO`, `_POSIX_CPUTIME`,
15583 `_POSIX_SPAWN`, `_POSIX_SPORADIC_SERVER`, `_POSIX_THREAD_CPUTIME`,
15584 `_POSIX_THREAD_SPORADIC_SERVER`, and `_POSIX_TIMEOUTS`, and `pathconf()` variables are
15585 added for `_PC_ALLOC_SIZE_MIN`, `_PC_REC_INCR_XFER_SIZE`, `_PC_REC_MAX_XFER_SIZE`,
15586 `_PC_REC_MIN_XFER_SIZE`, and `_PC_REC_XFER_ALIGN` for alignment with
15587 IEEE Std. 1003.1d-1999.
- 15588 The following are added for alignment with IEEE Std. 1003.1j-2000:
 - 15589 • Option symbolic constants `_POSIX_BARRIERS`, `_POSIX_CLOCK_SELECTION`,
15590 `_POSIX_MONOTONIC_CLOCK`, `_POSIX_READER_WRITER_LOCKS`,
15591 `_POSIX_SPIN_LOCKS`, and `_POSIX_TYPED_MEMORY_OBJECTS`
 - 15592 • `sysconf()` variables `_SC_BARRIERS`, `_SC_CLOCK_SELECTION`,
15593 `_SC_MONOTONIC_CLOCK`, `_SC_READER_WRITER_LOCKS`, `_SC_SPIN_LOCKS`, and
15594 `_SC_TYPED_MEMORY_OBJECTS`
- 15595 The `_SC_XBS5` macros associated with the ISO/IEC 9899:1990 standard are marked `LEGACY`,
15596 and new equivalent `_SC_V6` macros associated with the ISO/IEC 9899:1999 standard are
15597 introduced.
- 15598 The `getwd()` function is marked `LEGACY`.
- 15599 The `restrict` keyword is added to the prototypes for `realink()` and `swab()`.
- 15600 Constants for options are now harmonized, so when supported they take the year of approval of
15601 IEEE Std. 1003.1-200x as the value.
- 15602 The following are added for alignment with IEEE Std. 1003.1q-2000:
 - 15603 • Optional symbolic constants `_POSIX_TRACE`, `_POSIX_TRACE_EVENT_FILTER`,
15604 `_POSIX_TRACE_LOG`, and `_POSIX_TRACE_INHERIT`
 - 15605 • The `sysconf()` symbolic constants `_SC_TRACE`, `_SC_TRACE_EVENT_FILTER`,
15606 `_SC_TRACE_LOG`, and `_SC_TRACE_INHERIT`.

15607 **NAME**

15608 utime.h — access and modification times structure

15609 **SYNOPSIS**

15610 #include <utime.h>

15611 **DESCRIPTION**

15612 The <utime.h> header shall declare the structure **utimbuf**, which shall include the following
15613 members:

15614 time_t actime Access time.
15615 time_t modtime Modification time.

15616 The times shall be measured in seconds since the Epoch.

15617 The type **time_t** shall be defined as described in <sys/types.h>.

15618 The following shall be declared as a function and may also be defined as a macro. Function
15619 prototypes shall be provided for use with an ISO C standard compiler.

15620 int utime(const char *, const struct utimbuf *);

15621 **APPLICATION USAGE**

15622 None.

15623 **RATIONALE**

15624 None.

15625 **FUTURE DIRECTIONS**

15626 None.

15627 **SEE ALSO**

15628 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *utime()*

15629 **CHANGE HISTORY**

15630 First released in Issue 3.

15631 **Issue 4**

15632 Reference to the <sys/types.h> header is added for the definition of **time_t**. This is marked as an
15633 extension.

15634 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 15635 • The function declarations in this header are expanded to full ISO C standard prototypes.

15636 **Issue 6**

15637 The following new requirements on POSIX implementations derive from alignment with the
15638 Single UNIX Specification:

- 15639 • The **time_t** type is defined.

15640 **NAME**

15641 utmpx.h — user accounting database definitions

15642 **SYNOPSIS**15643 XSI `#include <utmpx.h>`

15644

15645 **DESCRIPTION**15646 The **<utmpx.h>** header shall define the **utmpx** structure that shall include at least the following
15647 members:

15648	char	ut_user[]	User login name.
15649	char	ut_id[]	Unspecified initialization process identifier.
15650	char	ut_line[]	Device name.
15651	pid_t	ut_pid	Process ID.
15652	short	ut_type	Type of entry.
15653	struct timeval	ut_tv	Time entry was made.

15654 The **pid_t** type shall be defined through **typedef** as described in **<sys/types.h>**.15655 The **timeval** structure shall be defined as described in **<sys/time.h>**.15656 Inclusion of the **<utmpx.h>** header may also make visible all symbols from **<sys/time.h>**.15657 The following symbolic constants shall be defined as possible values for the *ut_type* member of
15658 the **utmpx** structure:

15659	EMPTY	No valid user accounting information.
15660	BOOT_TIME	Identifies time of system boot.
15661	OLD_TIME	Identifies time when system clock changed.
15662	NEW_TIME	Identifies time after system clock changed.
15663	USER_PROCESS	Identifies a process.
15664	INIT_PROCESS	Identifies a process spawned by the init process.
15665	LOGIN_PROCESS	Identifies the session leader of a logged in user.
15666	DEAD_PROCESS	Identifies a session leader who has exited.

15667 The following shall be declared as functions and may also be defined as macros. Function
15668 prototypes shall be provided for use with an ISO C standard compiler.

15669	void	endutxent(void);
15670	struct utmpx	*getutxent(void);
15671	struct utmpx	*getutxid(const struct utmpx *);
15672	struct utmpx	*getutxline(const struct utmpx *);
15673	struct utmpx	*pututxline(const struct utmpx *);
15674	void	setutxent(void);

15675 **APPLICATION USAGE**

15676 None.

15677 **RATIONALE**

15678 None.

15679 **FUTURE DIRECTIONS**

15680 None.

15681 **SEE ALSO**15682 <sys/time.h>, <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *endutxent()*15683 **CHANGE HISTORY**

15684 First released in Issue 4, Version 2.

15685 **NAME**15686 **wchar.h** — wide-character types15687 **SYNOPSIS**

15688 #include <wchar.h>

15689 **DESCRIPTION**

15690 **CX** The functionality described on this reference page extends the ISO C standard. Applications
 15691 shall define the appropriate feature test macro (see the System Interfaces volume of
 15692 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 15693 symbols in this header.

15694 The <**wchar.h**> header shall define the following data types through **typedef**:

15695 **wchar_t** As described in <**stddef.h**>.

15696 **wint_t** An integer type capable of storing any valid value of **wchar_t** or WEOF.

15697 **wctype_t** A scalar type of a data object that can hold values which represent locale-
 15698 specific character classification.

15699 **mbstate_t** An object type other than an array type that can hold the conversion state
 15700 information necessary to convert between sequences of (possibly multibyte)
 15701 **XSI** characters and wide characters. If a codeset is being used such that an
 15702 **mbstate_t** needs to preserve more than 2 levels of reserved state, the results
 15703 are unspecified.

15704 **XSI** **FILE** As described in <**stdio.h**>.

15705 **size_t** As described in <**stddef.h**>.

15706 The <**wchar.h**> header shall declare the following as functions and may also define them as
 15707 macros. Function prototypes shall be provided for use with an ISO C standard compiler.

```

15708        wint_t         btowc(int);
15709        wint_t         fgetwc(FILE *);
15710        wchar_t       *fgetws(wchar_t *restrict, int, FILE *restrict);
15711        wint_t         fputwc(wchar_t, FILE *);
15712        int            fputws(const wchar_t *restrict, FILE *restrict);
15713        int            fwide(FILE *, int);
15714        int            fwprintf(FILE *restrict, const wchar_t *restrict, ...);
15715        int            fwscanf(FILE *restrict, const wchar_t *restrict, ...);
15716        wint_t         getwc(FILE *);
15717        wint_t         getwchar(void);
15718        int            iswalnum(wint_t);
15719        int            iswalpha(wint_t);
15720        int            iswcntrl(wint_t);
15721        int            iswctype(wint_t, wctype_t);
15722        int            iswdigit(wint_t);
15723        int            iswgraph(wint_t);
15724        int            iswlower(wint_t);
15725        int            iswprint(wint_t);
15726        int            iswpunct(wint_t);
15727        int            iswspace(wint_t);
15728        int            iswupper(wint_t);
15729        int            iswxdigit(wint_t);
15730        size_t         mbrlen(const char *restrict, size_t, mbstate_t *restrict);
15731        size_t         mbrtowc(wchar_t *restrict, const char *restrict, size_t,
```

```

15732         mbstate_t *restrict);
15733 int      mbsinit(const mbstate_t *);
15734 size_t   mbsrtowcs(wchar_t *restrict, const char **restrict, size_t,
15735               mbstate_t *restrict);
15736 wint_t   putwc(wchar_t, FILE *);
15737 wint_t   putwchar(wchar_t);
15738 int      swprintf(wchar_t *restrict, size_t,
15739               const wchar_t *restrict, ...);
15740 int      swscanf(const wchar_t *restrict,
15741               const wchar_t *restrict, ...);
15742 wint_t   tolower(wint_t);
15743 wint_t   toupper(wint_t);
15744 wint_t   ungetwc(wint_t, FILE *);
15745 int      vfwprintf(FILE *restrict, const wchar_t *restrict, va_list);
15746 int      vwscanf(FILE *restrict, const wchar_t *restrict, va_list);
15747 int      vwprintf(const wchar_t *restrict, va_list);
15748 int      vswprintf(wchar_t *restrict, size_t,
15749               const wchar_t *restrict, va_list);
15750 int      vswscanf(const wchar_t *restrict, const wchar_t *restrict,
15751               va_list);
15752 int      vwscanf(const wchar_t *restrict, va_list);
15753 size_t   wcrctomb(char *restrict, wchar_t, mbstate_t *restrict);
15754 wchar_t  *wcsconcat(wchar_t *restrict, const wchar_t *restrict);
15755 wchar_t  *wcschr(const wchar_t *, wchar_t);
15756 int      wcscmp(const wchar_t *, const wchar_t *);
15757 int      wscoll(const wchar_t *, const wchar_t *);
15758 wchar_t  *wcscpy(wchar_t *restrict, const wchar_t *restrict);
15759 size_t   wcscspn(const wchar_t *, const wchar_t *);
15760 size_t   wcsftime(wchar_t *restrict, size_t,
15761               const wchar_t *restrict, const struct tm *restrict);
15762 size_t   wcslen(const wchar_t *);
15763 wchar_t  *wcsncat(wchar_t *restrict, const wchar_t *restrict, size_t);
15764 int      wcsncmp(const wchar_t *, const wchar_t *, size_t);
15765 wchar_t  *wcsncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15766 wchar_t  *wcpbrk(const wchar_t *, const wchar_t *);
15767 wchar_t  *wcsrchr(const wchar_t *, wchar_t);
15768 size_t   wcsrtombs(char *restrict, const wchar_t **restrict,
15769               size_t, mbstate_t *restrict);
15770 size_t   wcsspncpy(const wchar_t *, const wchar_t *);
15771 wchar_t  *wcssstr(const wchar_t *restrict, const wchar_t *restrict);
15772 double  wcstod(const wchar_t *restrict, wchar_t **restrict);
15773 float   wcstof(const wchar_t *restrict, wchar_t **restrict);
15774 wchar_t  *wcstok(wchar_t *restrict, const wchar_t *restrict,
15775               wchar_t **restrict);
15776 long    wcstol(const wchar_t *restrict, wchar_t **restrict, int);
15777 long double wcstold(const wchar_t *restrict, wchar_t **restrict);
15778 long long wcstoll(const wchar_t *restrict, wchar_t **restrict, int);
15779 unsigned long wcstoul(const wchar_t *restrict, wchar_t **restrict, int);
15780 unsigned long long wcstoull(const wchar_t *restrict, wchar_t **restrict, int);
15781
15782 XSI wchar_t  *wcsvcs(const wchar_t *, const wchar_t *);
15783 int      wcswidth(const wchar_t *, size_t);

```

```

15784     size_t      wcsxfrm(wchar_t *restrict, const wchar_t *restrict, size_t);
15785     int         wctob(wint_t);
15786     wctype_t    wctype(const char *);
15787 XSI     int         wcwidth(wchar_t);
15788     wchar_t     *wmemchr(const wchar_t *, wchar_t, size_t);
15789     int         wmemcmp(const wchar_t *, const wchar_t *, size_t);
15790     wchar_t     *wmemcpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15791     wchar_t     *wmemmove(wchar_t *, const wchar_t *, size_t);
15792     wchar_t     *wmemset(wchar_t *, wchar_t, size_t);
15793     int         wprintf(const wchar_t *restrict, ...);
15794     int         wscanf(const wchar_t *restrict, ...);

```

15795 The **<wchar.h>** header shall define the following macro names:

```

15796     WCHAR_MAX   The maximum value representable by an object of type wchar_t.
15797     WCHAR_MIN   The minimum value representable by an object of type wchar_t.
15798     WEOF        Constant expression of type wint_t that is returned by several WP functions
15799               to indicate end-of-file.
15800     NULL        As described in <stddef.h>.
15801     The tag tm shall be declared as naming an incomplete structure type, the contents of which are
15802               described in the header <time.h>.
15803     Inclusion of the <wchar.h> header may make visible all symbols from the headers <ctype.h>,
15804               <stdio.h>, <stdarg.h>, <stdlib.h>, <string.h>, <stddef.h>, and <time.h>.

```

15805 APPLICATION USAGE

15806 None.

15807 RATIONALE

15808 None.

15809 FUTURE DIRECTIONS

15810 None.

15811 SEE ALSO

15812 **<ctype.h>**, **<stdarg.h>**, **<stddef.h>**, **<stdio.h>**, **<stdlib.h>**, **<string.h>**, **<time.h>**, the System
15813 Interfaces volume of IEEE Std. 1003.1-200x, *btowc()*, *fgetwc()*, *fgetws()*, *fputwc()*, *fputws()*,
15814 *fwide()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *iswalnum()*, *iswalphalpha()*, *iswcntrl()*, *iswctype()*,
15815 *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*,
15816 *iswctype()*, *mbsinit()*, *mbrlen()*, *mbrtowc()*, *mbsrtowcs()*, *putwc()*, *putwchar()*, *swprintf()*, *swscanf()*,
15817 *towlower()*, *towupper()*, *ungetwc()*, *vfwprintf()*, *vfwscanf()*, *vswprintf()*, *vswscanf()*, *vscanf()*,
15818 *wcrtomb()*, *wcsrtombs()*, *wscat()*, *wcschr()*, *wscmp()*, *wscoll()*, *wscopy()*, *wscspn()*, *wcsftime()*,
15819 *wcslen()*, *wcsncat()*, *wcsncmp()*, *wcsncpy()*, *wcspbrk()*, *wcsrchr()*, *wcsspn()*, *wcsstr()*, *wcstod()*,
15820 *wcstof()*, *wcstok()*, *wcstol()*, *wcstold()*, *wcstoll()*, *wcstoul()*, *wcstoull()*, *wcswcs()*, *wcswidth()*,
15821 *wcsxfrm()*, *wctob()*, *wctype()*, *wcwidth()*, *wmemchr()*, *wmemcmp()*, *wmemcpy()*, *wmemmove()*,
15822 *wmemset()*, *wprintf()*, *wscanf()*

15823 CHANGE HISTORY

15824 First released in Issue 4.

15825 Issue 5

15826 Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15827 **Issue 6**

15828 The Open Group corrigenda item U021/10 has been applied. The prototypes for *wcswidth()* and
15829 *wcwidth()* are marked as extensions.

15830 The Open Group corrigenda item U028/5 has been applied, correcting the prototype for the
15831 *mbsinit()* function.

15832 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15833 • Various function prototypes are updated to add the **restrict** keyword.
- 15834 • The functions *vwscanf()*, *vswscanf()*, *wcstof()*, *wctold()*, *wctoll()*, and *wcstoull()* are added.

15835 **NAME**

15836 wctype.h — wide-character classification and mapping utilities

15837 **SYNOPSIS**

15838 #include <wctype.h>

15839 **DESCRIPTION**

15840 cx The functionality described on this reference page extends the ISO C standard. Applications
 15841 shall define the appropriate feature test macro (see the System Interfaces volume of
 15842 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 15843 symbols in this header.

15844 The **<wctype.h>** header shall define the following data types through **typedef**:

15845 **wint_t** As described in **<wchar.h>**.

15846 **wctrans_t** A scalar type that can hold values which represent locale-specific character
 15847 mappings.

15848 **wctype_t** As described in **<wchar.h>**.

15849 The **<wctype.h>** header shall declare the following as functions and may also define them as
 15850 macros. Function prototypes shall be provided for use with an ISO C standard compiler.

```

15851 int      iswalnum(wint_t);
15852 int      iswalpha(wint_t);
15853 int      iswblank(wint_t);
15854 int      iswcntrl(wint_t);
15855 int      iswdigit(wint_t);
15856 int      iswgraph(wint_t);
15857 int      iswlower(wint_t);
15858 int      iswprint(wint_t);
15859 int      iswpunct(wint_t);
15860 int      iswspace(wint_t);
15861 int      iswupper(wint_t);
15862 int      iswxdigit(wint_t);
15863 int      iswctype(wint_t, wctype_t);
15864 wint_t   towctrans(wint_t, wctrans_t);
15865 wint_t   tolower(wint_t);
15866 wint_t   toupper(wint_t);
15867 wctrans_t wctrans(const char *);
15868 wctype_t wctype(const char *);

```

15869 The **<wctype.h>** header shall define the following macro name:

15870 **WEOF** Constant expression of type **wint_t** that is returned by several MSE functions
 15871 to indicate end-of-file.

15872 For all functions described in this header that accept an argument of type **wint_t**, the value is
 15873 representable as a **wchar_t** or equals the value of **WEOF**. If this argument has any other value,
 15874 the behavior is undefined.

15875 The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.

15876 Inclusion of the **<wctype.h>** header may make visible all symbols from the headers **<ctype.h>**,
 15877 **<stdio.h>**, **<stdarg.h>**, **<stdlib.h>**, **<string.h>**, **<stddef.h>**, **<time.h>**, and **<wchar.h>**.

15878 **APPLICATION USAGE**

15879 None.

15880 **RATIONALE**

15881 None.

15882 **FUTURE DIRECTIONS**

15883 None.

15884 **SEE ALSO**

15885 <locale.h>, <wchar.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *iswalnum()*,
15886 *iswalpha()*, *iswblank()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
15887 *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *towctrans()*, *towlower()*, *towupper()*,
15888 *wctrans()*, *wctype()*

15889 **CHANGE HISTORY**

15890 First released in Issue 5. Derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15891 **Issue 6**15892 The *iswblank()* function is added for alignment with the ISO/IEC 9899:1999 standard.

15893 **NAME**

15894 wordexp.h — word-expansion types

15895 **SYNOPSIS**

15896 #include <wordexp.h>

15897 **DESCRIPTION**15898 The **<wordexp.h>** header shall define the structures and symbolic constants used by the
15899 *wordexp()* and *wordfree()* functions.15900 The structure type **wordexp_t** shall contain at least the following members:

15901	size_t	we_wordc	Count of words matched by <i>words</i> .
15902	char	**we_wordv	Pointer to list of expanded words.
15903	size_t	we_offs	Slots to reserve at the beginning of <i>we_wordv</i> .

15904 The *flags* argument to the *wordexp()* function shall be the bitwise-inclusive OR of the following
15905 flags:

15906	WRDE_APPEND	Append words to those previously generated.
15907	WRDE_DOOFFS	Number of null pointers to prepend to <i>we_wordv</i> .
15908	WRDE_NOCMD	Fail if command substitution is requested.
15909	WRDE_REUSE	The <i>pwordexp</i> argument was passed to a previous successful call to <i>wordexp()</i> , and has not been passed to <i>wordfree()</i> . The result is the same as if the application had called <i>wordfree()</i> and then called <i>wordexp()</i> without WRDE_REUSE.
15910		
15911		
15912		
15913	WRDE_SHOWERR	Do not redirect <i>stderr</i> to <i>/dev/null</i> .
15914	WRDE_UNDEF	Report error on an attempt to expand an undefined shell variable.

15915 The following constants shall be defined as error return values:

15916	WRDE_BADCHAR	One of the unquoted characters—<newline>, ' ', ' & ', ' ; ', ' < ', ' > ', ' (', ') ', ' { ', ' } '—appears in <i>words</i> in an inappropriate context.
15917		
15918	WRDE_BADVAL	Reference to undefined shell variable when WRDE_UNDEF is set in <i>flags</i> .
15919	WRDE_CMDSUB	Command substitution requested when WRDE_NOCMD was set in <i>flags</i> .
15920	WRDE_NOSPACE	Attempt to allocate memory failed.
15921	WRDE_NOSYS	The implementation does not support the function.
15922	WRDE_SYNTAX	Shell syntax error, such as unbalanced parentheses or unterminated string.
15923		

15924 The following shall be declared as functions and may also be declared as macros. Function
15925 prototypes shall be provided for use with an ISO C standard compiler.

```
15926 int wordexp(const char *restrict, wordexp_t *restrict, int);
15927 void wordfree(wordexp_t *);
```

15928 The implementation may define additional macros or constants using names beginning with
15929 WRDE_.

15930 **APPLICATION USAGE**

15931 None.

15932 **RATIONALE**

15933 None.

15934 **FUTURE DIRECTIONS**

15935 None.

15936 **SEE ALSO**15937 The System Interfaces volume of IEEE Std. 1003.1-200x, *wordexp()*, the Shell and Utilities volume
15938 of IEEE Std. 1003.1-200x15939 **CHANGE HISTORY**

15940 First released in Issue 4. Derived from the ISO POSIX-2 standard.

15941 **Issue 6**15942 The **restrict** keyword is added to the prototype for *wordexp()*.

Index

1

2	(time) resolution	96	<poll.h>.....	320
3	/	211	<pthread.h>	322
4	/dev	211	<pwd.h>	327
5	/dev/console	211	<regex.h>.....	329
6	/dev/null.....	211	<sched.h>	331
7	/dev/tty	211	<search.h>.....	333
8	/tmp	211	<semaphore.h>	335
9	<aio.h>	232	<setjmp.h>	336
10	<alert>	42	<signal.h>.....	338
11	<arpa/inet.h>	234	<space>	101
12	<assert.h>	235	<spawn.h>	346
13	<backspace>	47	<stdarg.h>	348
14	<blank>	53	<stdbool.h>	350
15	<carriage-return>.....	56	<stddef.h>	351
16	<complex.h>	236	<stdint.h>	352
17	<cpio.h>	240	<stdio.h>.....	358
18	<ctype.h>.....	242	<stdlib.h>	362
19	<dirent.h>.....	244	<string.h>	366
20	<dlfcn.h>	246	<strings.h>	368
21	<errno.h>.....	247	<stropts.h>	369
22	<fcntl.h>	251	<sys/dir.h>	244
23	<fenv.h>.....	254	<sys/ipc.h>.....	374
24	<float.h>	259	<sys/mman.h>.....	376
25	<fmtmsg.h>	263	<sys/msg.h>.....	379
26	<fnmatch.h>	265	<sys/resource.h>	381
27	<form-feed>	73	<sys/select.h>	383
28	<ftw.h>	266	<sys/sem.h>	385
29	<glob.h>.....	268	<sys/shm.h>.....	387
30	<grp.h>	270	<sys/socket.h>	389
31	<iconv.h>.....	272	<sys/stat.h>	394
32	<inttypes.h>.....	273	<sys/statvfs.h>	399
33	<iso646.h>	276	<sys/time.h>	401
34	<langinfo.h>	277	<sys/timeb.h>.....	403
35	<libgen.h>	280	<sys/times.h>	404
36	<limits.h>	281	<sys/types.h>	405
37	<locale.h>	296	<sys/uio.h>.....	408
38	<math.h>	298	<sys/un.h>.....	409
39	<monetary.h>	305	<sys/utsname.h>.....	410
40	<mqueue.h>.....	306	<sys/wait.h>	411
41	<ndbm.h>.....	308	<syslog.h>	413
42	<net/if.h>.....	309	<tab>	107
43	<netdb.h>	310	<tar.h>.....	415
44	<netinet/in.h>.....	314	<termios.h>.....	417
45	<netinet/tcp.h>.....	318	<tgmath.h>	423
46	<newline>.....	82	<time.h>	427
47	<nl_types.h>	319	<trace.h>.....	431

48	<ucontext.h>.....	435	_POSIX2_LOCALEDEF	437, 442
49	<ulimit.h>.....	436	_POSIX2_PBS	442
50	<unistd.h>.....	437	_POSIX2_PBS_ACCOUNTING	442
51	<utime.h>.....	459	_POSIX2_PBS_CHECKPOINT	442
52	<utmpx.h>.....	460	_POSIX2_PBS_LOCATE.....	442
53	<vertical-tab>.....	116	_POSIX2_PBS_MESSAGE.....	442
54	<wchar.h>.....	462	_POSIX2_PBS_TRACK.....	442
55	<wctype.h>.....	466	_POSIX2_RE_DUP_MAX.....	283, 286, 290
56	<wordexp.h>.....	468	_POSIX2_SW_DEV	442
57	±0.....	118	_POSIX2_UPE.....	437, 443
58	_CS_PATH	445	_POSIX2_VERSION.....	437
59	_CS_XBS5_ILP32_OFF32_CFLAGS.....	447	_POSIX_ADVISORY_INFO.....	21, 32, 438
60	_CS_XBS5_ILP32_OFF32_LDFLAGS.....	447	_POSIX_AIO_LISTIO_MAX.....	282, 287
61	_CS_XBS5_ILP32_OFF32_LIBS.....	447	_POSIX_AIO_MAX.....	282, 287
62	_CS_XBS5_ILP32_OFF32_LINTFLAGS.....	447	_POSIX_ARG_MAX	282, 287
63	_CS_XBS5_ILP32_OFFBIG_CFLAGS.....	447	_POSIX_ASYNCHRONOUS_IO.....	21, 28, 32, 438
64	_CS_XBS5_ILP32_OFFBIG_LDFLAGS.....	447	_POSIX_ASYNC_IO	445
65	_CS_XBS5_ILP32_OFFBIG_LIBS.....	447	_POSIX_BARRIERS.....	21, 30, 32, 438
66	_CS_XBS5_ILP32_OFFBIG_LINTFLAGS.....	447	_POSIX_BASE.....	20, 444
67	_CS_XBS5_LP64_OFF64_CFLAGS.....	447	_POSIX_CHILD_MAX	287
68	_CS_XBS5_LP64_OFF64_LDFLAGS.....	448	_POSIX_CHOWN_RESTRICTED.....	21, 438, 457
69	_CS_XBS5_LP64_OFF64_LIBS.....	448	_POSIX_CLOCKRES_MIN.....	286
70	_CS_XBS5_LP64_OFF64_LINTFLAGS.....	448	_POSIX_CLOCK_SELECTION.....	21, 32, 438
71	_CS_XBS5_LPBIG_OFFBIG_CFLAGS.....	448	_POSIX_CPUTIME.....	21, 32, 438
72	_CS_XBS5_LPBIG_OFFBIG_LDFLAGS.....	448	_POSIX_C_LANG_SUPPORT.....	20, 25, 444
73	_CS_XBS5_LPBIG_OFFBIG_LIBS.....	448	_POSIX_C_LANG_SUPPORT_R.....	21, 24-25, 444
74	_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS.....	448	_POSIX_DELAYTIMER_MAX	282, 287
75	_IOFBF	358	_POSIX_DEVICE_IO.....	20, 25, 444
76	_IOLBF	358	_POSIX_DEVICE_SPECIFIC	20, 25, 444
77	_IONBF	358	_POSIX_DEVICE_SPECIFIC_R.....	25, 444
78	_MIN	281	_POSIX_FD_MGMT	20, 25, 444
79	_PC constants		_POSIX_FIFO.....	20, 25, 444
80	defined in <unistd.h>.....	451	_POSIX_FILE_ATTRIBUTES	20, 25, 444
81	_POSIX.....	281	_POSIX_FILE_LOCKING	21, 24-25, 444
82	_POSIX maximum values		_POSIX_FILE_SYSTEM	20, 25, 444
83	in <limits.h>.....	286	_POSIX_FSYNC	21, 24, 28, 32, 438
84	_POSIX minimum values		_POSIX_IPV6.....	21, 32
85	in <limits.h>.....	286	_POSIX_JOB_CONTROL...20-21, 26, 439, 444, 457	
86	_POSIX2_BC_BASE_MAX	285, 289	_POSIX_LINK_MAX.....	284, 287
87	_POSIX2_BC_DIM_MAX	285, 290	_POSIX_LOGIN_NAME_MAX	282, 287
88	_POSIX2_BC_SCALE_MAX	285, 290	_POSIX_MAPPED_FILES	21, 24, 28, 32, 439
89	_POSIX2_BC_STRING_MAX	286, 290	_POSIX_MAPPED_FILES,.....	28
90	_POSIX2_CHARCLASS_NAME_MAX.....	286, 290	_POSIX_MAX_CANON	284, 287
91	_POSIX2_CHAR_TERM.....	437, 442	_POSIX_MAX_INPUT	284, 287
92	_POSIX2_COLL_WEIGHTS_MAX	286, 290	_POSIX_MEMLOCK.....	21, 28, 33, 439
93	_POSIX2_C_BIND.....	437, 442	_POSIX_MEMLOCK_RANGE.....	21, 28, 33, 439
94	_POSIX2_C_DEV.....	442	_POSIX_MEMORY_PROTECTION.....	21, 24
95	_POSIX2_EXPR_NEST_MAX.....	286, 290	28, 33, 439
96	_POSIX2_FORT_DEV.....	442	_POSIX_MESSAGE_PASSING.....	21, 28, 33, 439
97	_POSIX2_FORT_RUN.....	442	_POSIX_MONOTONIC_CLOCK.....	22, 33, 439
98	_POSIX2_LINE_MAX	286, 290, 292	_POSIX_MQ_OPEN_MAX	282, 287

Index

99	<code>_POSIX_MQ_PRIO_MAX</code>	282, 287	
100	<code>_POSIX_MULTIPLE_PROCESS</code>	20, 26, 444	
101	<code>_POSIX_NAME_MAX</code>	285, 287	
102	<code>_POSIX_NETWORKING</code>	20, 26, 444	
103	<code>_POSIX_NGROUPS_MAX</code>	288	
104	<code>_POSIX_NO_TRUNC</code>	21, 439, 457	
105	<code>_POSIX_OPEN_MAX</code>	288	
106	<code>_POSIX_PATH_MAX</code>	285, 288	
107	<code>_POSIX_PIPE</code>	20, 26, 444	
108	<code>_POSIX_PIPE_BUF</code>	285, 288	
109	<code>_POSIX_PRIORITIZED_IO</code>	22, 28, 33, 439	
110	<code>_POSIX_PRIORITY_SCHEDULING</code>	22, 28, 33, 439	
111	<code>_POSIX_PRIO_IO</code>	445	
112	<code>_POSIX_RAW_SOCKETS</code>	22	
113	<code>_POSIX_READER_WRITER_LOCKS</code>	439	
114	<code>_POSIX_REALTIME_SIGNALS</code>	22, 28, 33, 439	
115	<code>_POSIX_REGEX</code>	439	
116	<code>_POSIX_RE_DUP_MAX</code>	288	
117	<code>_POSIX_RTSIG_MAX</code>	283, 288	
118	<code>_POSIX_SAVED_IDS</code>	21, 440, 457	
119	<code>_POSIX_SEMAPHORES</code>	22, 28, 33, 440	
120	<code>_POSIX_SEM_NSEMS_MAX</code>	283, 288	
121	<code>_POSIX_SEM_VALUE_MAX</code>	283, 288	
122	<code>_POSIX_SHARED_MEMORY_OBJECTS</code>	22	
123	28, 33, 440	
124	<code>_POSIX_SHELL</code>	33, 440	
125	<code>_POSIX_SIGNALS</code>	20, 26, 444	
126	<code>_POSIX_SIGQUEUE_MAX</code>	283, 288	
127	<code>_POSIX_SINGLE_PROCESS</code>	20, 26, 444	
128	<code>_POSIX_SPAWN</code>	22, 33, 440	
129	<code>_POSIX_SPIN_LOCKS</code>	22, 30, 33, 440	
130	<code>_POSIX_SPARADIC_SERVER</code>	22, 33, 440	
131	<code>_POSIX_SSIZE_MAX</code>	288, 291	
132	<code>_POSIX_SS_REPL_MAX</code>	283, 288	
133	<code>_POSIX_STREAM_MAX</code>	283, 288	
134	<code>_POSIX_SYMLINK_MAX</code>	285, 289	
135	<code>_POSIX_SYMLOOP_MAX</code>	283, 289	
136	<code>_POSIX_SYNCHRONIZED_IO</code>	22, 28, 34, 440	
137	<code>_POSIX_SYNC_IO</code>	445	
138	<code>_POSIX_SYSTEM_DATABASE</code>	20, 26, 444	
139	<code>_POSIX_SYSTEM_DATABASE_R</code>	21, 24, 26, 444	
140	<code>_POSIX_THREADS</code>	22, 24, 30, 34, 441	
141	<code>_POSIX_THREAD_ATTR_STACKADDR</code>	22, 24	
142	34, 440	
143	<code>_POSIX_THREAD_ATTR_STACKSIZE</code>	22, 24	
144	34, 440	
145	<code>_POSIX_THREAD_CPUTIME</code>	22, 30, 34, 440	
146	<code>_POSIX_THREAD_DESTRUCTOR</code>		
147	<code>ITERATIONS</code>	283, 289	
148	<code>_POSIX_THREAD_KEYS_MAX</code>	283, 289	
149	<code>_POSIX_THREAD_PRIORITY_SCHEDULING</code>		
	22, 29-30, 34, 441	
	<code>_POSIX_THREAD_PRIO_INHERIT</code>	22, 29	
	34, 440	
	<code>_POSIX_THREAD_PRIO_PROTECT</code>	22, 29	
	34, 441	
	<code>_POSIX_THREAD_PROCESS_SHARED</code>	22	
	24, 34, 441	
	<code>_POSIX_THREAD_SAFE_FUNCTIONS</code>	22	
	24, 30, 34, 441	
	<code>_POSIX_THREAD_SPORADIC_SERVER</code>	22	
	30, 34, 441	
	<code>_POSIX_THREAD_THREADS_MAX</code>	283, 289	
	<code>_POSIX_TIMEOUTS</code>	22, 34, 441	
	<code>_POSIX_TIMERS</code>	22, 28, 30, 34, 441	
	<code>_POSIX_TIMER_MAX</code>	283, 289	
	<code>_POSIX_TTY_NAME_MAX</code>	284, 289	
	<code>_POSIX_TYPED_MEMORY_OBJECTS</code>	22, 35, 441	
	<code>_POSIX_TZNAME_MAX</code>	284, 289	
	<code>_POSIX_USER_GROUPS</code>	20, 26, 444	
	<code>_POSIX_USER_GROUPS_R</code>	21, 24, 26, 444	
	<code>_POSIX_VDISABLE</code>	21, 441, 457	
	<code>_POSIX_VERSION</code>	437	
	<code>_SC constants</code>		
	defined in <unistd.h>.....	448	
	<code>_SC_COLL_WIGHTS_MAX</code>	456	
	<code>_SC_EXPR_NEST_MAX</code>	456	
	<code>_SC_LINE_MAX</code>	456	
	<code>_SC_RE_DUP_MAX</code>	456	
	<code>_SC_STREAM_MAX</code>	456	
	<code>_SC_TZNAME_MAX</code>	456	
	<code>_XBS5_ILP32_OFF32</code>	443	
	<code>_XBS5_ILP32_OFFBIG</code>	443	
	<code>_XBS5_LP64_OFF64</code>	443	
	<code>_XBS5_LPBIG_OFFBIG</code>	443	
	<code>_XOPEN_CRYPT</code>	22, 24, 27, 443	
	<code>_XOPEN_ENH_I18N</code>	443	
	<code>_XOPEN_IOV_MAX</code>	282, 290	
	<code>_XOPEN_LEGACY</code>	22, 24, 31-32, 443	
	<code>_XOPEN_REALTIME</code>	23-24, 27-28, 443	
	<code>_XOPEN_REALTIME_THREADS</code>	23-24, 29, 443	
	<code>_XOPEN_SHM</code>	443	
	<code>_XOPEN_STREAMS</code>	31, 444	
	<code>_XOPEN_UNIX</code>	23, 437	
	<code>_XOPEN_VERSION</code>	437	
	<code>_XOPEN_XCU_VERSION</code>	437	
	<code>_XOPEN_XPG2</code>	437	
	<code>_XOPEN_XPG3</code>	437	
	<code>_XOPEN_XPG4</code>	438	
	<code>ABDAY</code>	278	
	<code>ABMON</code>	278	
	<code>abortive release</code>	41	

150	absolute path name.....	41, 123	asynchronously-generated signal.....	45
151	access mode.....	41	ATEXIT_MAX.....	282
152	additional file access control mechanism.....	41	attribute selection.....	420
153	address space.....	41	authentication.....	46
154	ADV.....	10	authorization.....	46
155	advisory information.....	41	background job.....	46
156	affirmative response.....	42	background process.....	46
157	AF_INET.....	391	background process group.....	46
158	AF_INET6.....	391	backquote.....	47
159	AF_UNIX.....	391	backslash.....	47
160	AF_UNSPEC.....	391	backspace character.....	47
161	AIO.....	10	bandinfo.....	369
162	AIO_ALLDONE.....	232	BAR.....	10
163	AIO_CANCELED.....	232	barrier.....	47
164	AIO_LISTIO_MAX.....	281	base character.....	47
165	AIO_MAX.....	282	basename.....	47
166	AIO_NOTCANCELED.....	232	basic regular expression.....	48, 198
167	AIO_PRIO_DELTA_MAX.....	282	batch access list.....	48
168	AI_CANONNAME.....	311	batch administrator.....	48
169	AI_NUMERICHOST.....	311	batch client.....	48
170	AI_PASSIVE.....	311	batch destination.....	48
171	alert.....	42	batch destination identifier.....	48
172	alert character.....	42	batch directive.....	49
173	alias name.....	42	batch job.....	49
174	alignment.....	42	batch job attribute.....	49
175	alternate file access control mechanism.....	43	batch job identifier.....	49
176	alternate signal stack.....	43	batch job name.....	49
177	ALT_DIGITS.....	278	batch job owner.....	50
178	AM_STR.....	278	batch job priority.....	50
179	anchoring.....	202	batch job state.....	50
180	ancillary data.....	43	batch name service.....	50
181	angle brackets.....	43	batch name space.....	50
182	ANYMARK.....	372	batch node.....	50
183	API.....	44	batch operator.....	51
184	application.....	43	batch queue.....	51
185	application address.....	43	batch queue attribute.....	51
186	application conformance.....	38	batch queue position.....	51
187	application program interface.....	44	batch queue priority.....	51
188	appropriate privileges.....	44	batch rerunability.....	52
189	AREGTYPE.....	415	batch restart.....	52
190	argument.....	44	batch server.....	52
191	ARG_MAX.....	282	batch server name.....	52
192	arm (a timer).....	44	batch service.....	52
193	assignment.....	44	batch service request.....	52
194	asterisk.....	45	batch submission.....	53
195	async-cancel-safe function.....	45	batch system.....	53
196	async-signal-safe function.....	45	batch target user.....	53
197	asynchronous events.....	45	batch user.....	53
198	asynchronous I/O completion.....	46	baud rate selection.....	419
199	asynchronous I/O operation.....	46	BC_BASE_MAX.....	285
200	asynchronous input and output.....	45	BC_DIM_MAX.....	285

Index

201	BC_SCALE_MAX	285	CHILD_MAX.....	282
202	BC_STRING_MAX.....	286	CHRTYPE	415
203	BE	10	circumflex.....	57
204	bind.....	53	CLD_CONTINUED	342
205	blank character.....	53	CLD_DUMPED	342
206	blank line	53	CLD_EXITED	342
207	blkcnt_t	405	CLD_KILLED	342
208	blksize_t.....	405	CLD_STOPPED.....	342
209	BLKTYPE.....	415	CLD_TRAPPED	342
210	block special file.....	54	CLOCAL.....	420
211	block-mode terminal.....	54	clock.....	57
212	blocked process (or thread)	54	clock jump.....	57
213	blocking	54	clock tick.....	58
214	BOOT_TIME.....	460	clockid_t	405
215	braces.....	54	CLOCKS_PER_SEC	405, 427-428
216	brackets.....	54	CLOCK_MONOTONIC	428
217	BRE (ERE) matching a single character	196	CLOCK_PROCESS_CPUTIME_ID.....	427
218	BRE (ERE) matching multiple characters.....	196	CLOCK_REALTIME.....	286, 427
219	break value.....	55	clock_t	405
220	BRKINT	418	CLOCK_THREAD_CPUTIME_ID.....	427
221	broadcast	55	CMMSG_DATA	390
222	BSD	244	CMMSG_FIRSTHDR.....	390
223	BSDLY	419	CMMSG_NXTHDR	390
224	BSn.....	419	coded character set.....	58
225	BUFSIZ.....	358	codeset	58
226	built-in.....	55	CODESET	278
227	built-in utility	55	collating element.....	58
228	BUS_ADRALN.....	342	collating element order.....	58
229	BUS_ADRERR.....	342	collation	59
230	BUS_OBJERR.....	342	collation sequence	59
231	byte	55	COLL_WEIGHTS_MAX	286
232	byte input/output functions	56	column position	59
233	can.....	8	COLUMNS.....	192
234	canonical mode input processing	215	command.....	59
235	carriage-return character.....	56	command language interpreter.....	60
236	CD.....	10	composite graphic symbol.....	60
237	character	56	condition variable.....	60
238	character array.....	56	conformance	19, 38
239	character class	56	POSIX	19
240	character encoding	136	POSIX system interfaces	20
241	state-dependent	140	XSI.....	19
242	character set.....	57	XSI system interfaces.....	23
243	character special file.....	57	conformance document.....	19
244	character string.....	57	conforming application.....	19
245	CHARCLASS_NAME_MAX	286, 292	conforming implementation options	25
246	charmap		connection	60
247	description	137	connection mode.....	60
248	CHAR_BIT	290	connectionless mode.....	60
249	CHAR_MAX.....	291	control character	61
250	CHAR_MIN.....	291	control modes.....	420
251	child process	57	control operator	61

252	controlling process	61	utimbuf	459
253	controlling terminal	61, 214	data type	
254	CONTTYPE	415	ACTION	333
255	conversion descriptor	61	cc_t	417
256	core file	61	DIR	244
257	CPT	11	div_t	362
258	CPU	404	ENTRY	333
259	CPU time	62	FILE	359
260	clock	62	fpos_t	359
261	timer	62	glob_t	268
262	CRDLY	418	ldiv_t	362
263	CREAD	420	mbstate_t	462
264	CRn	418	msglen_t	379
265	CRNCYSTR	278	msgqnum_t	379
266	CS	11	nl_catd	319
267	CSIZE	420	nl_item	319
268	Csn	420	ptrdiff_t	351
269	CSTOPB	420	regex_t	329
270	current working directory	62, 117	regmatch_t	329
271	cursor position	62	regoff_t	329
272	CX	11	shmatt_t	387
273	C_ constants in <cpio.h>	240	sigset_t	338
274	C_IRGRP	240	sig_atomic_t	338
275	C_IROTH	240	size_t	351
276	C_IRUSR	240	speed_t	417
277	C_ISBLK	240	tflag_t	417
278	C_ISCHR	240	VISIT	333
279	C_ISCTG	240	wchar_t	351
280	C_ISDIR	240	wctrans_t	466
281	C_ISFIFO	240	wctype_t	462
282	C_ISGID	240	wint_t	462
283	C_ISLNK	240	data types	
284	C_ISREG	240	defined in <sys/types.h>	405
285	C_ISSOCK	240	DATEMSK	192
286	C_ISUID	240	DAY_	278
287	C_ISVTX	240	DBL_ constants	
288	C_IWGRP	240	defined in <float.h>	261
289	C_IWOTH	240	DBL_DIG	261, 290
290	C_IWUSR	240	DBL_EPSILON	262
291	C_IXGRP	240	DBL_MANT_DIG	261
292	C_IXOTH	240	DBL_MAX	262, 290
293	C_IXUSR	240	DBL_MAX_10_EXP	262
294	data segment	63	DBL_MAX_EXP	261
295	data structure		DBL_MIN	262
296	dirent	244	DBL_MIN_10_EXP	261
297	entry	333	DBL_MIN_EXP	261
298	group	270	DBM	308
299	lconv	296	DBM_INSERT	308
300	msqid_ds	379	DBM_REPLACE	308
301	stat	394	DEAD_PROCESS	460
302	tms	404	deferred batch service	63

Index

303	DELAYTIMER_MAX	282	EDESTADDRREQ	247
304	device	63	EDOM	247
305	output	211	EDQUOT	247
306	device ID	63	EEXIST	247
307	dev_t	405	EFAULT	247
308	DIR	244	EFBIG	247
309	directory	63	effective group ID	65
310	directory entry (or link)	63	effective user ID	65
311	directory stream	63	EHOSTUNREACH	247
312	dirent structure	244	EIDRM	247
313	DIRTYPE	415	eight-bit transparency	65
314	disarm (a timer)	64	EILSEQ	247
315	display	64	EINPROGRESS	248
316	documentation	19	EINTR	248
317	dollar sign	64	EINVAL	248
318	dot	64	EIO	248
319	dot-dot	64	EISCONN	248
320	double-quote	64	EISDIR	248
321	downshifting	65	ELOOP	248
322	driver	65	EMFILE	248
323	D_FMT	278	EMLINK	248
324	D_T_FMT	278	EMPTY	460
325	E2BIG	247	empty directory	65
326	EACCES	247	empty line	66
327	EADDRINUSE	247	empty string (or null string)	66
328	EADDRNOTAVAIL	247	empty wide-character string	66
329	EAFNOSUPPORT	247	EMSGSIZE	248
330	EAGAIN	247	EMULTIHOP	248
331	EAI_AGAIN	312	ENAMETOOLONG	248
332	EAI_BADFLAGS	312	encoding	
333	EAI_FAIL	312	character	136
334	EAI_FAMILY	312	encoding rule	66
335	EAI_MEMORY	312	encryption	27
336	EAI_NONAME	312	ENETDOWN	248
337	EAI_SERVICE	312	ENETUNREACH	248
338	EAI_SOCKTYPE	312	ENFILE	248
339	EAI_SYSTEM	312	ENOBUFS	248
340	EALREADY	247	ENODATA	248
341	EBADF	247	ENODEV	248
342	EBADMSG	247	ENOENT	248
343	EBUSY	247	ENOEXEC	248
344	ECANCELED	247	ENOLCK	248
345	ECHILD	247	ENOLINK	248
346	ECHO	420	ENOMEM	248
347	ECHOE	420	ENOMSG	248
348	ECHOK	420	ENOPROTOPT	248
349	ECHONL	420	ENOSPC	248
350	ECONNABORTED	247	ENOSR	248
351	ECONNREFUSED	247	ENOSTR	248
352	ECONNRESET	247	ENOSYS	248
353	EDEADLK	247	ENOTCONN	248

354	ENOTDIR.....	248	FD.....	11
355	ENOTEMPTY.....	248	FD_CLOEXEC.....	251
356	ENOTSOCK.....	248	FD_CLR.....	401
357	ENOTSUP.....	248	FD_ISSET.....	401
358	ENOTTY.....	249	FD_SET.....	401
359	entire regular expression.....	66, 196	fd_set.....	401
360	environment variables		FD_SETSIZE.....	401
361	internationalization.....	189	FD_ZERO.....	401
362	ENXIO.....	249	feature test macro.....	68
363	EOF.....	358	FFDLY.....	419
364	EOPNOTSUPP.....	249	FFn.....	419
365	EOVERFLOW.....	249	field.....	68
366	EPERM.....	249	FIFO.....	69
367	EPIPE.....	249	FIFO special file.....	69
368	epoch.....	66	FIFO_TYPE.....	415
369	EPROTO.....	249	file.....	69
370	EPROTONOSUPPORT.....	249	FILE.....	358, 462
371	EPROTOTYPE.....	249	file access permissions.....	121
372	equivalence class.....	66	file characteristics	
373	era.....	67	data structure.....	396
374	ERA.....	278	header.....	396
375	ERANGE.....	249	file description.....	69
376	ERA_D_FMT.....	278	file descriptor.....	69
377	ERA_D_T_FMT.....	278	file group class.....	69
378	ERA_T_FMT.....	278	file mode.....	70
379	EROFS.....	249	file mode bits.....	70
380	ESPIPE.....	249	file name.....	70
381	ESRCH.....	249	file name portability.....	70
382	ESTALE.....	249	file offset.....	70
383	ETIME.....	249	file other class.....	71
384	ETIMEDOUT.....	249	file owner class.....	71
385	ETXTBSY.....	249	file permission bits.....	71
386	event management.....	67	file serial number.....	71
387	EWouldBlock.....	249	file system.....	71
388	EXDEV.....	249	file times update.....	122
389	executable file.....	67	file type.....	71
390	execute.....	67	FILENAME_MAX.....	358
391	execution time.....	62, 67	FILESIZEBITS.....	284
392	measurement.....	123	filter.....	72
393	monitoring.....	67	FIPS.....	21
394	EXIT_FAILURE.....	362	first open (of a file).....	72
395	EXIT_SUCCESS.....	362	flow control.....	72
396	expand.....	68	FLT_constants	
397	EXPR_NEST_MAX.....	286	defined in <float.h>.....	261
398	extended regular expression.....	68, 203	FLT_DIG.....	261, 290
399	extended security controls.....	68	FLT_EPSILON.....	262
400	extension		FLT_MANT_DIG.....	261
401	CX.....	11	FLT_MAX.....	262, 290
402	OH.....	12	FLT_MAX_10_EXP.....	261
403	XSI.....	16	FLT_MAX_EXP.....	261
404	F-LOCK.....	451	FLT_MIN.....	262

Index

405	FLT_MIN_10_EXP.....	261	F_GETLK.....	251
406	FLT_MIN_EXP.....	261	F_GETOWN.....	251
407	FLT_RADIX.....	259, 261	F_OK.....	445
408	FLT_ROUNDS.....	259	F_RDLCK.....	251
409	FLUSHR.....	370	F_SETFD.....	251
410	FLUSHRW.....	370	F_SETFL.....	251
411	FLUSHW.....	370	F_SETLK.....	251
412	FMNAMESZ.....	370	F_SETLKW.....	251
413	FNM_constants		F_SETOWN.....	251
414	in <fnmatch.h>.....	265	F_TEST.....	451
415	FNM_NOESCAPE.....	265	F_TLOCK.....	451
416	FNM_NOMATCH.....	265	F_ULOCK.....	451
417	FNM_NOSYS.....	265	F_UNLCK.....	251
418	FNM_PATHNAME.....	265	F_WRLCK.....	251
419	FNM_PERIOD.....	265	GETALL.....	385
420	FOPEN_MAX.....	283, 358	GETNCNT.....	385
421	foreground job.....	72	GETPID.....	385
422	foreground process.....	72	GETVAL.....	385
423	foreground process group.....	72	GETZCNT.....	385
424	foreground process group ID.....	72	gid_t.....	405
425	form-feed character.....	73	GLOB_constants	
426	format of entries.....	231	defined in <glob.h>.....	268
427	FPE_FLTDIV.....	342	GLOB_ABORTED.....	268
428	FPE_FLTINV.....	342	GLOB_APPEND.....	268
429	FPE_FLTOVF.....	342	GLOB_DOOFFS.....	268
430	FPE_FLTRES.....	342	GLOB_ERR.....	268
431	FPE_FLTSUB.....	342	GLOB_MARK.....	268
432	FPE_FLTUND.....	342	GLOB_NOCHECK.....	268
433	FPE_INTDIV.....	342	GLOB_NOESCAPE.....	268
434	FPE_INTOVF.....	342	GLOB_NOMATCH.....	268
435	FR.....	11	GLOB_NOSORT.....	268
436	fsblkcnt_t.....	405	GLOB_NOSPACE.....	268
437	FSC.....	11	GLOB_NOSYS.....	268
438	fsfilcnt_t.....	405	grammar	
439	FTW.....	266	locale.....	176
440	FTW_constants		regular expression.....	206
441	in <ftw.h>.....	266	graphic character.....	73
442	FTW_CHDIR.....	266	group database.....	73
443	FTW_D.....	266	group ID.....	73
444	FTW_DEPTH.....	266	group name.....	73
445	FTW_DNR.....	266	hard limit.....	74
446	FTW_DP.....	266	hard link.....	74
447	FTW_F.....	266	headers.....	231
448	FTW_MOUNT.....	266	HOME.....	192
449	FTW_NS.....	266	home directory.....	74
450	FTW_PHYS.....	266	host byte order.....	74
451	FTW_SL.....	266	HUGE_VAL.....	299
452	FTW_SLN.....	266	HUPCL.....	420
453	F_DUPFD.....	251	ICANON.....	420
454	F_GETFD.....	251	ICRNL.....	418
455	F_GETFL.....	251	idtype_t.....	411

456	id_t.....	405	IPC_EXCL.....	374
457	IEXTEN.....	420	IPC_NOWAIT.....	374
458	IGNBRK.....	418	IPC_PRIVATE.....	374
459	IGNCR.....	418	IPC_RMID.....	374
460	IGNPAR.....	418	IPC_SET.....	374
461	ILL_BADSTK.....	342	IPC_STAT.....	374
462	ILL_COPROC.....	342	IPPROTO_ICMP.....	315
463	ILL_ILLADR.....	342	IPPROTO_IP.....	315
464	ILL_ILLOPC.....	342	IPPROTO_TCP.....	315
465	ILL_ILLOPN.....	342	IPPROTO_UDP.....	315
466	ILL_ILLTRP.....	342	IPV6_JOIN_GROUP.....	315
467	ILL_PRVOPC.....	342	IPV6_LEAVE_GROUP.....	315
468	ILL_PRIVREG.....	342	IPV6_MULTICAST_HOPS.....	316
469	implementation-defined.....	8	IPV6_MULTICAST_IF.....	316
470	IN6_IS_ADDR_LINKLOCAL.....	316	IPV6_MULTICAST_LOOP.....	316
471	IN6_IS_ADDR_LOOPBACK.....	316	IPV6_UNICAST_HOPS.....	316
472	IN6_IS_ADDR_MC_GLOBAL.....	316	ISIG.....	420
473	IN6_IS_ADDR_MC_LINKLOCAL.....	316	ISTRIP.....	418
474	IN6_IS_ADDR_MC_NODELOCAL.....	316	itimerval.....	401
475	IN6_IS_ADDR_MC_ORGLOCAL.....	316	ITIMER_PROF.....	401
476	IN6_IS_ADDR_MC_SITELOCAL.....	316	ITIMER_REAL.....	401
477	IN6_IS_ADDR_MULTICAST.....	316	ITIMER_VIRTUAL.....	401
478	IN6_IS_ADDR_SITELOCAL.....	316	IXANY.....	418
479	IN6_IS_ADDR_UNSPECIFIED.....	316	IXOFF.....	418
480	IN6_IS_ADDR_V4COMPAT.....	316	IXON.....	418
481	IN6_IS_ADDR_V4MAPPED.....	316	I_ATMARK.....	372
482	INADDR_ANY.....	315	I_CANPUT.....	372
483	INADDR_BROADCAST.....	315	I_CKBAND.....	372
484	incomplete line.....	74	I_FDINSERT.....	371
485	INET6_ADDRSTRLEN.....	315	I_FIND.....	371
486	INET_ADDRSTRLEN.....	315	I_FLUSH.....	370
487	Inf.....	74	I_FLUSHBAND.....	370
488	INIT_PROCESS.....	460	I_GETBAND.....	372
489	INLCR.....	418	I_GETCLTIME.....	372
490	ino_t.....	405	I_GETSIG.....	371
491	INPCK.....	418	I_GRDOPT.....	371
492	interactive shell.....	75	I_GWROPT.....	371
493	internationalization.....	75	I_LINK.....	372
494	interprocess communication.....	75	I_LIST.....	372
495	INT_MAX.....	291	I_LOOK.....	370
496	INT_MIN.....	291	I_NREAD.....	371
497	invalid.....	196	I_PEEK.....	371
498	invariant values.....	292	I_PLINK.....	372
499	invoke.....	75	I_POP.....	370
500	iovec.....	408	I_PUNLINK.....	372
501	IOV_MAX.....	282	I_PUSH.....	370
502	IP6.....	11	I_RECVFD.....	372
503	IPC.....	374	I_SENDFD.....	371
504	IPC_constants.....		I_SETCLTIME.....	372
505	defined in <sys/ipc.h>.....	374	I_SETSIG.....	370
506	IPC_CREAT.....	374	I_SRDOPT.....	371

Index

507	I_STR.....	371	local customs	77
508	I_SWROPT	371	local IPC.....	77
509	I_UNLINK.....	372	local modes	420
510	job.....	75	locale	77, 143
511	job control.....	76	grammar.....	176
512	job control job ID	76	POSIX.....	144
513	key_t.....	405	locale definition	145
514	LANG.....	189	localization.....	77
515	last close (of a file)	76	login.....	77
516	LASTMARK.....	372	login name.....	78
517	LC_ALL.....	189 , 296	LOGIN_NAME_MAX.....	282
518	LC_COLLATE.....	189 , 286, 296	LOGIN_PROCESS.....	460
519	description	155	LOGNAME.....	192
520	LC_CTYPE.....	189 , 278, 296, 466	LOG_ALERT	414
521	description	147	LOG_AUTH.....	413
522	LC_MESSAGES.....	189 , 278, 296, 319	LOG_CONS.....	413
523	description	174	LOG_CRIT	414
524	LC_MONETARY.....	189 , 278, 296	LOG_CRON	413
525	description	163	LOG_DAEMON	413
526	LC_NUMERIC.....	189 , 278, 296	LOG_DEBUG	414
527	description	166	LOG_EMERG	414
528	LC_TIME.....	190 , 278, 296	LOG_ERR.....	414
529	description	168	LOG_INFO.....	414
530	LDBL_constants		LOG_KERN	413
531	defined in <float.h>.....	261	LOG_LOCAL.....	413
532	LDBL_DIG.....	261	LOG_LPR	413
533	LDBL_EPSILON	262	LOG_MAIL.....	413
534	LDBL_MANT_DIG	261	LOG_MASK.....	413
535	LDBL_MAX	262	LOG_NDELAY.....	413
536	LDBL_MAX_10_EXP	262	LOG_NEWS.....	413
537	LDBL_MAX_EXP.....	261	LOG_NOTICE.....	414
538	LDBL_MIN.....	262	LOG_NOWAIT	413
539	LDBL_MIN_10_EXP	261	LOG_ODELAY	413
540	LDBL_MIN_EXP.....	261	LOG_PID.....	413
541	legacy	8, 31	LOG_USER	413
542	limit		LOG_UUCP.....	413
543	numerical	290	LOG_WARNING	414
544	line	76	LONG_BIT	290-291
545	line control	421	LONG_MAX.....	291
546	LINES.....	192	LONG_MIN.....	292
547	LINE_MAX.....	286	lower multiplexing.....	103
548	linger.....	76	L_ctermid	358
549	link	77	L_tmpnam.....	358
550	link count.....	77	MAGIC	240
551	LINK_MAX.....	284	MAN.....	11
552	LIO_NOP.....	232	map.....	78
553	LIO_NOWAIT.....	232	MAP_FIXED	376
554	LIO_READ	232	MAP_PRIVATE	376
555	LIO_WAIT.....	232	MAP_SHARED	376
556	LIO_WRITE	232	margin codes	
557	LNKTYPE.....	415	notation.....	17

558	marked message	78	mode.....	79
559	matched.....	78, 196	mode_t.....	405
560	MAXARGS.....	349	MON.....	12
561	MAXFLOAT.....	299	monotonic clock.....	80
562	maximum values.....	286	MON_.....	278
563	MAX_CANON.....	284	MORECTL.....	372
564	MAX_INPUT.....	284	MOREDATA.....	372
565	may.....	8	mount point.....	80
566	MB_CUR_MAX.....	362	MPR.....	12
567	MB_LEN_MAX.....	290-291	MQ_OPEN_MAX.....	282
568	MCL_CURRENT.....	376	MQ_PRIO_MAX.....	282
569	MCL_FUTURE.....	376	MSG.....	12
570	mcontext_t.....	435	MSGVERB.....	192
571	memory mapped files.....	78	MSG_ANY.....	372
572	memory object.....	78	MSG_BAND.....	372
573	memory-resident.....	79	MSG_CTRUNC.....	391
574	message.....	79	MSG_DONTRROUTE.....	391
575	message catalog.....	79	MSG_EOR.....	391
576	message catalog descriptor.....	79	MSG_HIPRI.....	372
577	message queue.....	79	MSG_NOERROR.....	379
578	MF.....	11	MSG_OOB.....	391
579	minimum values.....	286	MSG_PEEK.....	391
580	MINSIGSTKSZ.....	340	MSG_TRUNC.....	391
581	ML.....	12	MSG_WAITALL.....	391
582	MLR.....	12	MS_ASYNC.....	376
583	MM_macros.....	263	MS_INVALIDATE.....	376
584	MM_APPL.....	263	MS_SYNC.....	376
585	MM_CONSOLE.....	263	multi-character collating element.....	80
586	MM_ERROR.....	263	mutex.....	80
587	MM_FIRM.....	263	MUXID_ALL.....	372
588	MM_HALT.....	263	M.....	298
589	MM_HARD.....	263	M_E.....	298
590	MM_INFO.....	263	M_LN.....	298
591	MM_NOCON.....	264	M_LOG10E.....	298
592	MM_NOMSG.....	263	M_LOG2E.....	298
593	MM_NOSEV.....	263	M_PI.....	298
594	MM_NOTOK.....	263	M_SQRT1_2.....	299
595	MM_NRECOV.....	263	M_SQRT2.....	299
596	MM_NULLACT.....	263	name.....	80
597	MM_NULLLBL.....	263	named STREAM.....	80
598	MM_NULLMC.....	263	NAME_MAX.....	244, 285
599	MM_NULLSEV.....	263	NaN (Not a Number).....	81
600	MM_NULLTAG.....	263	native language.....	81
601	MM_NULLTXT.....	263	NCCS.....	417
602	MM_OK.....	263	NDEBUG.....	235
603	MM_OPSYS.....	263	negative response.....	81
604	MM_PRINT.....	263	network.....	81
605	MM_RECOVER.....	263	network address.....	81
606	MM_SOFT.....	263	network byte order.....	81
607	MM_UTIL.....	263	newline character.....	82
608	MM_WARNING.....	263	NEW_TIME.....	460

Index

609	NGROUPS_MAX	286	operand	84
610	nice value.....	82	operator	84
611	NI_DGRAM.....	312	OPOST	418
612	NI_NAMEREQD	312	option	84
613	NI_NOFQDN.....	311	ADV.....	10
614	NI_NUMERICHOST	311	AIO.....	10
615	NI_NUMERICSERV	312	BAR.....	10
616	NLDLY.....	418	BE.....	10
617	nlink_t.....	405	CD.....	10
618	NLn.....	418	CPT.....	11
619	NLSPATH.....	190	CS.....	11
620	NL_ARGMAX.....	292	FD.....	11
621	NL_CAT_LOCALE.....	319	FR.....	11
622	NL_LANGMAX.....	292	FSC.....	11
623	NL_MSGMAX.....	292	IP6.....	11
624	NL_NMAX.....	292	MF.....	11
625	NL_SETD.....	319	ML.....	12
626	NL_SETMAX.....	292	MLR.....	12
627	NL_TEXTMAX.....	292	MON.....	12
628	NOEXPR.....	278	MPR.....	12
629	NOFLSH.....	420	MSG.....	12
630	non-blocking.....	82	PIO.....	13
631	non-canonical mode input processing.....	216	PS.....	13
632	non-spacing characters.....	82	RTS.....	13
633	NOSTR.....	278	SD.....	13
634	NUL.....	82	SEM.....	13
635	NULL.....	351, 358, 362, 366, 427, 445	SHM.....	13
636	null byte.....	83	SIO.....	13
637	null pointer.....	83	SPI.....	14
638	null string.....	83	SPN.....	14
639	null wide-character code.....	83	SS.....	14
640	number sign.....	83	TCT.....	14
641	numerical limits.....	290	TEF.....	15
642	NZERO.....	292	THR.....	14
643	OB.....	12	TMO.....	14
644	object file.....	83	TMR.....	14
645	OCRNL.....	418	TPI.....	14
646	octet.....	83	TPP.....	15
647	OF.....	12	TPS.....	15
648	offset maximum.....	84	TRC.....	15
649	off_t.....	405	TRI.....	15
650	OFILL.....	418	TRL.....	15
651	OH.....	12	TSA.....	15
652	OLD_TIME.....	460	TSF.....	16
653	ONLCR.....	418	TSH.....	16
654	ONLRET.....	418	TSP.....	16
655	ONOCR.....	418	TSS.....	16
656	opaque address.....	84	TYM.....	16
657	open file.....	84	UP.....	16
658	open file description.....	84	XSR.....	17
659	OPEN_MAX.....	282, 335	option-argument.....	85

660	options		
661	shell and utilities.....	35	
662	system interfaces	32	
663	orientation.....	85	
664	orphaned process group	85	
665	output devices.....	211	
666	O_ constants		
667	defined in <fcntl.h>.....	251-252	
668	O_ACCMODE.....	252	
669	O_APPEND	251	
670	O_CREAT	251	
671	O_DSYNC	251	
672	O_EXCL.....	251	
673	O_NOCTTY	251	
674	O_NONBLOCK	251	
675	O_RDONLY	252	
676	O_RDWR.....	252	
677	O_RSYNC	252	
678	O_SYNC	252	
679	O_TRUNC.....	251	
680	O_WRONLY	252	
681	page	85	
682	page size	85	
683	PAGESIZE.....	282	
684	PAGE_SIZE.....	282	
685	parameter	85	
686	PARENB.....	420	
687	parent directory	86	
688	parent process	86	
689	parent process ID.....	86	
690	PARMRK	418	
691	PARODD	420	
692	PATH.....	192	
693	path name.....	86	
694	path name component.....	86	
695	path name resolution.....	123	
696	path name variable values.....	284	
697	path prefix	86	
698	PATH_MAX.....	285, 293	
699	pattern.....	87	
700	period.....	87	
701	permissions	87	
702	persistence.....	87	
703	pid_t	405	
704	PIO	13	
705	pipe	87	
706	PIPE_BUF	285	
707	PM_STR.....	278	
708	POLLERR	320	
709	pollfd.....	320	
710	POLLHUP	320	
	POLLIN	320	
	polling	88	
	POLLNVAL	320	
	POLLOUT	320	
	POLLPRI.....	320	
	POLLRDBAND.....	320	
	POLLRDNORM.....	320	
	POLLWRBAND	320	
	POLLWRNORM.....	320	
	POLL_ERR	342	
	POLL_HUP	342	
	POLL_IN	342	
	POLL_MSG	342	
	POLL_OUT	342	
	POLL_PRI.....	342	
	portable character set	88, 133	
	portable file name character set	88, 122	
	positional parameter	88	
	POSIX		
	conformance	19	
	POSIX locale	144	
	POSIX shell and utilities.....	23	
	POSIX system interfaces		
	conformance	20	
	POSIX2_CHAR_TERM	23, 35	
	POSIX2_C_DEV	23, 35	
	POSIX2_FORT_DEV	23, 35	
	POSIX2_FORT_RUN	23, 35	
	POSIX2_LOCALEDEF.....	23, 36	
	POSIX2_PBS	23, 36	
	POSIX2_PBS_ACCOUNTING	23, 36	
	POSIX2_PBS_CHECKPOINT	36	
	POSIX2_PBS_LOCATE.....	23, 36	
	POSIX2_PBS_MESSAGE.....	23, 36	
	POSIX2_PBS_TRACK.....	23, 36	
	POSIX2_SW_DEV	23, 36	
	POSIX2_UPE	23, 36	
	POSIX_ALLOC_SIZE_MIN	285	
	POSIX_FADV_DONTNEED	252	
	POSIX_FADV_NOREUSE.....	252	
	POSIX_FADV_NORMAL.....	252	
	POSIX_FADV_RANDOM	252	
	POSIX_FADV_SEQUENTIAL	252	
	POSIX_FADV_WILLNEED.....	252	
	POSIX_MADV_DONTNEED	377	
	POSIX_MADV_NORMAL.....	376	
	POSIX_MADV_RANDOM.....	377	
	POSIX_MADV_SEQUENTIAL.....	377	
	POSIX_MADV_WILLNEED	377	
	POSIX_REC_INCR_XFER_SIZE.....	285	
	POSIX_REC_MAX_XFER_SIZE.....	285	

Index

711	POSIX_REC_MIN_XFER_SIZE.....	285	PTHREAD_CREATE_DETACHED.....	322
712	POSIX_REC_XFER_ALIGN.....	285	PTHREAD_CREATE_JOINABLE.....	322
713	POSIX_TYPED_MEM_ALLOCATE.....	377	PTHREAD_DESTRUCTOR_ITERATIONS.....	283
714	POSIX_TYPED_MEM_ALLOCATE_CONTIG.....	377	PTHREAD_EXPLICIT_SCHED.....	322
715	POSIX_TYPED_MEM_MAP_ALLOCATABLE.....	377	PTHREAD_INHERIT_SCHED.....	322
716	preallocation.....	88	PTHREAD_KEYS_MAX.....	283
717	preempted process (or thread).....	89	PTHREAD_MUTEX_DEFAULT.....	322
718	printable character.....	89	PTHREAD_MUTEX_ERRORCHECK.....	322
719	printable file.....	89	PTHREAD_MUTEX_INITIALIZER.....	322
720	priority.....	89	PTHREAD_MUTEX_NORMAL.....	322
721	priority band.....	89	PTHREAD_MUTEX_RECURSIVE.....	322
722	priority inversion.....	90	PTHREAD_ONCE_INIT.....	322
723	priority scheduling.....	90	PTHREAD_PRIO_INHERIT.....	322
724	priority-based scheduling.....	90	PTHREAD_PRIO_NONE.....	322
725	PRIO_constants		PTHREAD_PRIO_PROTECT.....	322
726	defined in <sys/resource.h>.....	381	PTHREAD_PROCESS_PRIVATE.....	322
727	PRIO_PGRP.....	381	PTHREAD_PROCESS_SHARED.....	322
728	PRIO_PROCESS.....	381	PTHREAD_RWLOCK_INITIALIZER.....	322
729	PRIO_USER.....	381	PTHREAD_SCOPE_PROCESS.....	322
730	privilege.....	90	PTHREAD_SCOPE_SYSTEM.....	322
731	process.....	90	PTHREAD_STACK_MIN.....	283
732	process group.....	90	PTHREAD_THREADS_MAX.....	283
733	process group ID.....	91	PWD.....	193
734	process group leader.....	91	P_ALL.....	411
735	process group lifetime.....	91	P_GID.....	411
736	process groups		P_PID.....	411
737	termios.....	213	P_tmpdir.....	358
738	process ID.....	91	radix character.....	93
739	process lifetime.....	91	RADIXCHAR.....	278
740	process memory locking.....	92	RAND_MAX.....	362
741	process termination.....	92	read-only file system.....	93
742	process virtual time.....	92	read-write lock.....	93
743	process-to-process communication.....	92	real group ID.....	93
744	profiling option groups.....	25	real time.....	93
745	program.....	92	real user ID.....	94
746	protocol.....	92	realtime.....	27
747	PROT_EXEC.....	376	REALTIME.....	232, 306, 331, 335
748	PROT_NONE.....	376	realtime signal extension.....	94
749	PROT_READ.....	376	realtime threads.....	29
750	PROT_READ constants		record.....	94
751	in <sys/mman.h>.....	376	redirection.....	94
752	PROT_WRITE.....	376	redirection operator.....	94
753	PS.....	13	reentrant function.....	94
754	pseudo-terminal.....	93	referenced shared memory object.....	95
755	PTHREAD_BARRIER_SERIAL_THREAD.....	322	refresh.....	95
756	PTHREAD_CANCELED.....	322	region.....	95
757	PTHREAD_CANCEL_ASYNCHRONOUS.....	322	REGTYPE.....	415
758	PTHREAD_CANCEL_DEFERRED.....	322	regular expression.....	95
759	PTHREAD_CANCEL_DISABLE.....	322	basic.....	198
760	PTHREAD_CANCEL_ENABLE.....	322	extended.....	203
761	PTHREAD_COND_INITIALIZER.....	322	grammar.....	206

762	regular file	95	RTLD_NOW	246
763	REG_constants		RTS	13
764	defined in <regex.h>	329	RTSIG_MAX	283, 339
765	REG_BADBR	329	runnable process (or thread)	96
766	REG_BADPAT	329	running process (or thread)	96
767	REG_BADRPT	330	runtime values	
768	REG_EBRACE	329	increasable	285
769	REG_EBRACK	329	invariant	281
770	REG_ECOLLATE	329	rusage	381
771	REG_ECTYPE	329	RUSAGE_CHILDREN	381
772	REG_EESCAPE	329	RUSAGE_SELF	381
773	REG_ENOSYS	330	R_OK	445
774	REG_EPAREN	329	saved resource limits	96
775	REG_ERANGE	330	saved set-group-ID	97
776	REG_ESPACE	330	saved set-user-ID	97
777	REG_ESUBREG	329	SA_constants	
778	REG_EXTENDED	329	declared in <signal.h>	340
779	REG_ICASE	329	SA_NOCLDSTOP	340
780	REG_NEWLINE	329	SA_NOCLDWAIT	340
781	REG_NOMATCH	329	SA_NODEFER	340
782	REG_NOSUB	329	SA_ONSTACK	340
783	REG_NOTBOL	329	SA_RESETHAND	340
784	REG_NOTEOL	329	SA_RESTART	340
785	relative path name	95, 123	SA_SIGINFO	340
786	relocatable file	95	SCHAR_MAX	291
787	relocation	96	SCHAR_MIN	291-292
788	requested batch service	96	scheduling	97
789	requirements	19	scheduling allocation domain	97
790	RE_DUP_MAX	283, 286	scheduling contention scope	97
791	rlimit	381	scheduling policy	97
792	RLIMIT_AS	382	SCHED_FIFO	331
793	RLIMIT_CORE	381	SCHED_OTHER	331
794	RLIMIT_CPU	381	SCHED_RR	331
795	RLIMIT_DATA	381	SCHED_SPORADIC	331
796	RLIMIT_FSIZE	381	SCM_RIGHTS	390
797	RLIMIT_NOFILE	382	screen	98
798	RLIMIT_STACK	382	scroll	98
799	RLIM_INFINITY	381	SD	13
800	RLIM_SAVED_CUR	381	SEEK_CUR	251, 358, 448
801	RLIM_SAVED_MAX	381	SEEK_END	251, 358, 448
802	RMSGD	371	SEEK_SET	251, 358, 448
803	RMSGN	371	SEGV_ACCERR	342
804	RNORM	371	SEGV_MAPERR	342
805	root directory	96	SEM	13
806	RPROTDAT	371	semaphore	98
807	RPROTDIS	371	SEM_NSEMS_MAX	283
808	RPROTNORM	371	SEM_UNDO	385
809	RS_HIPRI	371	SEM_VALUE_MAX	283
810	RTLD_GLOBAL	246	session	98
811	RTLD_LAZY	246	session leader	98
812	RTLD_LOCAL	246	session lifetime	99

Index

813	SETALL.....	385	SIGUSR1.....	339
814	SETVAL.....	385	SIGUSR2.....	339
815	shall.....	8	SIGVTALRM.....	339
816	shared memory object.....	99	SIGXCPU.....	339
817	shared memory objects.....	78	SIGXFSZ.....	339
818	shell.....	99	SIG_BLOCK.....	340
819	SHELL.....	193	SIG_DFL.....	338
820	shell script.....	99	SIG_ERR.....	338
821	shell, the.....	99	SIG_HOLD.....	338
822	SHM.....	13	SIG_IGN.....	338
823	SHM_RDONLY.....	387	SIG_SETMASK.....	340
824	SHM_RND.....	387	SIG_UNBLOCK.....	340
825	should.....	8	single-quote.....	100
826	SHRT_MAX.....	291	SIO.....	13
827	SHRT_MIN.....	292	size_t.....	366, 405
828	SHUT_RD.....	391	SI_ASYNCIO.....	342
829	SHUT_RDWR.....	392	SI_MESGQ.....	342
830	SHUT_WR.....	391	SI_QUEUE.....	342
831	SIGABRT.....	339	SI_TIMER.....	342
832	SIGALRM.....	339	SI_USER.....	342
833	SIGBUS.....	339, 342	slash.....	100
834	SIGCHLD.....	339, 342	SNDZERO.....	371
835	SIGCONT.....	339	socket.....	100
836	SIGEV_NONE.....	338	socket address.....	100
837	SIGEV_SIGNAL.....	338	SOCK_DGRAM.....	390
838	SIGEV_THREAD.....	338	SOCK_SEQPACKET.....	390
839	SIGFPE.....	339, 342	SOCK_STREAM.....	390
840	SIGHUP.....	339	soft limit.....	100
841	SIGILL.....	339, 342	SOL_SOCKET.....	390
842	siginfo_t.....	341	SOMAXCONN.....	391
843	SIGINT.....	339	source code.....	100
844	SIGKILL.....	339	SO_ACCEPTCONN.....	390
845	signal.....	99	SO_BROADCAST.....	390
846	signal stack.....	100	SO_DEBUG.....	391
847	SIGPIPE.....	339	SO_DONTROUTE.....	391
848	SIGPOLL.....	339, 342, 370	SO_ERROR.....	391
849	SIGPROF.....	339	SO_KEEPALIVE.....	391
850	SIGQUEUE_MAX.....	283	SO_LINGER.....	391
851	SIGQUIT.....	339	SO_OOINLINE.....	391
852	SIGRTMAX.....	338	SO_RCVBUF.....	391
853	SIGRTMIN.....	338	SO_RCVLOWAT.....	391
854	SIGSEGV.....	339, 342	SO_RCVTIMEO.....	391
855	SIGSTKSZ.....	340	SO_REUSEADDR.....	391
856	SIGSTOP.....	339	SO_SNDBUF.....	391
857	SIGSYS.....	339	SO_SNDLOWAT.....	391
858	SIGTERM.....	339	SO_SNDTIMEO.....	391
859	SIGTRAP.....	339, 342	SO_TYPE.....	391
860	SIGTSTP.....	339	space character.....	101
861	SIGTTIN.....	339	spawn.....	101
862	SIGTTOU.....	339	special built-in.....	101
863	SIGURG.....	339	special parameter.....	101

864	SPI.....	14	synchronized I/O file integrity completion.....	105
865	spin lock.....	101	synchronized I/O operation.....	105
866	SPN.....	14	synchronized input and output.....	104
867	sporadic server.....	101	synchronous I/O operation.....	105
868	SS.....	14	synchronously-generated signal.....	105
869	SSIZE_MAX.....	291, 406	system.....	105
870	ssize_t.....	405	system console.....	106
871	SS_DISABLE.....	340	system crash.....	105
872	SS_ONSTACK.....	340	system databases.....	106
873	SS_REPL_MAX.....	283	system documentation.....	106
874	stack_t.....	340	system process.....	106
875	standard error.....	101	system reboot.....	107
876	standard input.....	102	system-wide.....	107
877	standard output.....	102	S_ constants	
878	standard utilities.....	102	defined in <sys/stat.h>.....	394-395
879	stat data structure.....	394	S_ macros	
880	stderr.....	358	defined in <sys/stat.h>.....	395
881	STDERR_FILENO.....	452	S_BANDURG.....	371
882	stdin.....	358	S_ERROR.....	371
883	STDIN_FILENO.....	452	S_HANGUP.....	371
884	stdout.....	358	S_HIPRI.....	370
885	STDOUT_FILENO.....	452	S_IFBLK.....	394
886	strbuf.....	369	S_IFCHR.....	394
887	STREAM.....	102	S_IFDIR.....	395
888	stream.....	102	S_IFIFO.....	395
889	STREAM.....	248	S_IFLNK.....	395
890	STREAM end.....	103	S_IFMT.....	394
891	STREAM head.....	103	S_IFREG.....	395
892	STREAMS.....	31, 369	S_IFSOCK.....	395
893	STREAMS multiplexor.....	103	S_INPUT.....	370
894	STREAM_MAX.....	283	S_IRGRP.....	395
895	strfdinsert.....	369	S_IROTH.....	395
896	string.....	103	S_IRUSR.....	395
897	strioctl.....	369	S_IRWXG.....	395
898	strpeek.....	369	S_IRWXO.....	395
899	strrecvfd.....	369	S_IRWXU.....	395
900	str_list.....	369	S_ISBLK.....	395
901	str_mlist.....	370	S_ISCHR.....	395
902	ST_NOSUID.....	399	S_ISDIR.....	395
903	ST_RDONLY.....	399	S_ISFIFO.....	396
904	subshell.....	103	S_ISGID.....	395-396
905	successfully transferred.....	103	S_ISLNK.....	396
906	supplementary group ID.....	104	S_ISREG.....	396
907	suseconds_t.....	405	S_ISSOCK.....	396
908	suspended job.....	104	S_ISUID.....	395-396
909	symbolic link.....	104	S_ISVTX.....	395
910	SYMLINK_MAX.....	285, 293	S_IWGRP.....	395
911	SYMLOOP_MAX.....	283	S_IWOTH.....	395
912	SYMTYPE.....	415	S_IWUSR.....	395
913	synchronized I/O completion.....	104	S_IXGRP.....	395
914	synchronized I/O data integrity completion ..	104	S_IXOTH.....	395

Index

915	S_IXUSR	395	thread list.....	108
916	S_MSG.....	371	thread-safe	109
917	S_OUTPUT	370	thread-specific data key	109
918	S_RDBAND	370	tilde.....	109
919	S_RDNORM	370	timeb.....	403
920	S_TYPEISMQ.....	396	timeouts.....	109
921	S_TYPEISSEM	396	timer	109
922	S_TYPEISSHM	396	timer overrun	109
923	S_TYPEISTMO	396	TIMER_ABSTIME.....	427
924	S_WRBAND	370	TIMER_MAX.....	283
925	S_WRNORM	370	timer_t.....	405
926	tab character	107	timeval.....	401
927	TABDLY	418	time_t	405
928	TABn.....	418	TMAGIC.....	415
929	TCIFLUSH	421	TMAGLEN.....	415
930	TCIOFF	421	TMO	14
931	TCIOFLUSH.....	421	TMPDIR.....	193
932	TCION	421	TMP_MAX.....	293, 358
933	TCOFLUSH.....	421	TMR.....	14
934	TCOOFF	421	TOEXEC	415
935	TCOON	421	token.....	110
936	TCP_NODELAY	318	TOREAD.....	415
937	TCSADRAIN	420	TOSTOP.....	420
938	TCSAFLUSH	420	TOWRITE.....	415
939	TCSANOW	420	TPI.....	14
940	TCT	14	TPP.....	15
941	TEF.....	15	TPS.....	15
942	TERM	193	TRAP_BRKPT.....	342
943	terminal		TRAP_TRACE.....	342
944	controlling.....	214	TRC.....	15
945	terminal (or terminal device)	107	TRI	15
946	terminal types.....	211	TRL	15
947	termios	213	TSA	15
948	canonical mode input processing.....	215	TSF	16
949	control modes.....	222	TSGID.....	415
950	controlling terminal	214	TSH.....	16
951	input modes.....	219	TSP.....	16
952	local modes.....	223	TSS	16
953	non-canonical mode input processing.....	216	TSUID.....	415
954	output modes	220	TSVTX.....	415
955	process groups.....	213	TTY_NAME_MAX.....	284
956	special control characters	224	TUEXEC.....	415
957	text column	108	TUREAD.....	415
958	text file	108	TUWRITE.....	415
959	TGEXEC.....	415	TVERSION.....	415
960	TGREAD.....	415	TVERSLEN.....	415
961	TGWRITE.....	415	TYM.....	16
962	THOUSEP	278	typed memory name space	113
963	THR	14	typed memory object.....	114
964	thread.....	108	typed memory pool.....	114
965	thread ID.....	108	typed memory port.....	114

966	TZ.....	193	WEXITSTATUS.....	362, 411
967	TZNAME_MAX.....	284	white space.....	116
968	T_FMT.....	278	wide characters.....	137
969	T_FMT_AMPM.....	278	wide-character code (C language).....	116
970	t_uscalar_t.....	369	wide-character input/output functions.....	116
971	UCHAR_MAX.....	291	wide-character string.....	116
972	ucontext_t.....	435	WIFCONTINUED.....	411
973	uid_t.....	405	WIFEXITED.....	362, 411
974	UINT_MAX.....	291	WIFSIGNALED.....	362, 411
975	ULONG_MAX.....	291	WIFSTOPPED.....	362, 411
976	UL_GETFSIZE.....	436	WNOHANG.....	362, 411
977	UL_SETFSIZE.....	436	WNOWAIT.....	411
978	UN.....	16	word.....	117
979	unbind.....	114	WORD_BIT.....	290-291
980	undefined.....	8	working directory.....	117
981	unspecified.....	9	worldwide portability interface.....	117
982	UP.....	16	WRDE_APPEND.....	468
983	upper multiplexing.....	103	WRDE_BADCHAR.....	468
984	upshifting.....	114	WRDE_BADVAL.....	468
985	user database.....	114	WRDE_CMDSUB.....	468
986	user ID.....	115	WRDE_DOOFFS.....	468
987	user name.....	115	WRDE_NOCMD.....	468
988	USER_PROCESS.....	460	WRDE_NOSPACE.....	468
989	USHRT_MAX.....	291	WRDE_NOSYS.....	468
990	utility.....	115	WRDE_REUSE.....	468
991	Utility Syntax Guidelines.....	229	WRDE_SHOWERR.....	468
992	utmpx.....	460	WRDE_SYNTAX.....	468
993	variable.....	115	WRDE_UNDEF.....	468
994	VEOF.....	417	write.....	117
995	VEOL.....	417	WSTOPPED.....	411
996	VERASE.....	417	WSTOPSIG.....	362, 411
997	vertical-tab character.....	116	WTERMSIG.....	362, 411
998	VFS.....	399	WUNTRACED.....	362, 411
999	VINTR.....	417	W_OK.....	445
1000	VKILL.....	417	XSI.....	16, 117
1001	VQUIT.....	417	conformance.....	19
1002	VSTART.....	417	XSI conformance.....	23
1003	VSTOP.....	417	XSI options groups.....	27
1004	VSUSP.....	417	XSI STREAMS.....	31
1005	VTDLY.....	419	XSI system interfaces	
1006	VTn.....	419	conformance.....	23
1007	warning		XSI-conformant.....	117
1008	MAN.....	11	XSR.....	17
1009	OB.....	12	X_OK.....	445
1010	OF.....	12	YESEXPR.....	278
1011	UN.....	16	YESSTR.....	278
1012	WCHAR_MAX.....	464	zombie process.....	118
1013	WCHAR_MIN.....	464		
1014	WCONTINUED.....	411		
1015	WEOF.....	462, 464, 466		
1016	WEXITED.....	411		

Introduction

1.1 Scope

The scope of IEEE Std. 1003.1-200x is described in the Base Definitions volume of IEEE Std. 1003.1-200x.

1.2 Conformance

Conformance requirements for IEEE Std. 1003.1-200x are defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 2, Conformance.

1.3 Normative References

Normative references for IEEE Std. 1003.1-200x are defined in the Base Definitions volume of IEEE Std. 1003.1-200x.

1.4 Changes from Issue 4

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

The change history is subject to revision. The intent is to document changes from Issue 4 thru Issue 6, with the latest change history also documenting changes from the ISO POSIX-1: 1996 standard.

The following sections describe changes made to this volume of IEEE Std. 1003.1-200x since Issue 4. The CHANGE HISTORY section for each entry details the technical changes that have been made to that entry since Issue 4. Changes made between Issue 2 and Issue 4 are not included.

1.4.1 Changes from Issue 4 to Issue 4, Version 2

The following list summarizes the major changes that were made in this volume of IEEE Std. 1003.1-200x from Issue 4 to Issue 4, Version 2:

- The X/Open UNIX extension was added. This specifies the common core APIs of 4.3 Berkeley Software Distribution (BSD 4.3), the OSF AES, and SVID Issue 3.
- STREAMS were added as part of the X/Open UNIX extension.
- Existing Issue 4 functions were clarified as a result of industry feedback.

28 1.4.2 Changes from Issue 4, Version 2 to Issue 5

29 The following list summarizes the major changes that were made in this volume of
30 IEEE Std. 1003.1-200x from Issue 4, Version 2 to Issue 5:

- 31 • Functions previously defined in the ISO POSIX-2 standard C-language Binding, Shared
32 Memory, Enhanced Internationalization, and X/Open UNIX Extension Feature Groups were
33 moved to the BASE.
- 34 • Threads were added to the BASE for alignment with the POSIX Threads Extension.
- 35 • The Realtime Threads Feature Group was added.
- 36 • The Realtime Feature Group was added for alignment with the POSIX Realtime Extension.
- 37 • Multibyte Support Extensions (MSE) were added to the BASE for alignment with
38 ISO/IEC 9899:1990/Amendment 1:1995 (E).
- 39 • Large File Summit (LFS) Extensions were added to the BASE for support of 64-bit or larger
40 files and file systems.
- 41 • X/Open-specific threads extensions were added to the BASE.
- 42 • X/Open-specific dynamic linking functions were added to the BASE.
- 43 • A new category Legacy was added.
- 44 • The categories TO BE WITHDRAWN and WITHDRAWN were removed.

45 1.4.3 Changes from Issue 5 to Issue 6 (IEEE Std. 1003.1-200x)

46 The following list summarizes the major changes that were made in this volume of
47 IEEE Std. 1003.1-200x from Issue 5 to Issue 6:

- 48 • This volume of IEEE Std. 1003.1-200x is extensively revised so it can be both an IEEE POSIX
49 Standard and an Open Group Technical Standard.
- 50 • The POSIX System Interfaces requirements incorporate support of FIPS 151-2.
- 51 • The POSIX System Interfaces requirements are updated to align with some features of the
52 Single UNIX Specification.
- 53 • A RATIONALE section is added to each reference page.

54 **1.5 New Features**55 **1.5.1 New Features in Issue 4, Version 2**

56 The functions, headers, and external variables first introduced in Issue 4, Version 2 are listed in
57 the table below.

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

New Functions, Headers, and External Variables in Issue 4, Version 2				
FD_CLR()	endutxent()	gettimeofday()	ptsname()	sigaltstack()
FD_ISSET()	expm1()	getutxent()	putmsg()	sighold()
FD_SET()	fattach()	getutxid()	putpmsg()	sigignore()
FD_ZERO()	fchdir()	getutxline()	pututxline()	siginterrupt()
_longjmp()	fchmod()	getwd()	random()	sigpause()
_setjmp()	fchown()	grantpt()	re_comp()	sigrelse()
a64l()	fcvt()	ilogb()	re_exec()	sigset()
acosh()	fdetach()	index()	readlink()	sigstack()
asinh()	ffs()	initsate()	readv()	srandom()
atanh()	fntmsg()	insque()	realpath()	statvfs()
basename()	fstatvfs()	ioctl()	regcmp()	strcasecmp()
bcmp()	ftime()	isastream()	regex()	strdup()
bcopy()	ftok()	killpg()	remainder()	strncasecmp()
brk()	ftruncate()	l64a()	remque()	swapcontext()
bsd_signal()	gcvt()	lchown()	rindex()	symlink()
bzero()	getcontext()	lockf()	rint()	sync()
cbrt()	getdate()	log1p()	sbrk()	syslog()
closelog()	getdtablesize()	logb()	scalb()	tcgetsid()
dbm_clearerr()	getgrent()	lstat()	select()	truncate()
dbm_close()	gethostid()	makecontext()	setcontext()	ttyslot()
dbm_delete()	getitimer()	mknod()	setgrent()	ualarm()
dbm_error()	getmsg()	mkstemp()	setitimer()	unlockpt()
dbm_fetch()	getpagesize()	mktemp()	setlogmask()	usleep()
dbm_firstkey()	getpgid()	mmap()	setpgrp()	utimes()
dbm_nextkey()	getpmsg()	mprotect()	setpriority()	valloc()
dbm_open()	getpriority()	msync()	setpwent()	vfork()
dbm_store()	getpwent()	munmap()	setregid()	wait3()
dirname()	getrlimit()	nextafter()	setreuid()	waitid()
ecvt()	getrusage()	nftw()	setrlimit()	writev()
endgrent()	getsid()	openlog()	setstate()	
endpwent()	getsubopt()	poll()	setutxent()	
<fmtmsg.h>	<re_comp.h>	<sys/resource.h>	<sys/uio.h>	<utmpx.h>
<libgen.h>	<strings.h>	<sys/statvfs.h>	<sys/un.h>	
<ndbm.h>	<stropts.h>	<sys/time.h>	<syslog.h>	
<poll.h>	<sys/mman.h>	<sys/timeb.h>	<ucontext.h>	
getdate_err	__loc1			

96 **1.5.2 New Features in Issue 5**

97 The functions and headers first introduced in Issue 5 are listed in the table below.

	New Functions and Headers in Issue 5		
100	<i>aio_cancel()</i>	<i>pthread_attr_getstacksize()</i>	<i>pthread_self()</i>
101	<i>aio_error()</i>	<i>pthread_attr_init()</i>	<i>pthread_setcancelstate()</i>
102	<i>aio_fsync()</i>	<i>pthread_attr_setdetachstate()</i>	<i>pthread_setcanceltype()</i>
103	<i>aio_read()</i>	<i>pthread_attr_setguardsize()</i>	<i>pthread_setconcurrency()</i>
104	<i>aio_return()</i>	<i>pthread_attr_setinheritsched()</i>	<i>pthread_setschedparam()</i>
105	<i>aio_suspend()</i>	<i>pthread_attr_setschedparam()</i>	<i>pthread_setspecific()</i>
106	<i>aio_write()</i>	<i>pthread_attr_setschedpolicy()</i>	<i>pthread_sigmask()</i>
107	<i>asctime_r()</i>	<i>pthread_attr_setscope()</i>	<i>pthread_testcancel()</i>
108	<i>btowc()</i>	<i>pthread_attr_setstackaddr()</i>	<i>putc_unlocked()</i>
109	<i>clock_getres()</i>	<i>pthread_attr_setstacksize()</i>	<i>putchar_unlocked()</i>
110	<i>clock_gettime()</i>	<i>pthread_cancel()</i>	<i>pwrite()</i>
111	<i>clock_settime()</i>	<i>pthread_cleanup_pop()</i>	<i>rand_r()</i>
112	<i>ctime_r()</i>	<i>pthread_cleanup_push()</i>	<i>readdir_r()</i>
113	<i>dlclose()</i>	<i>pthread_cond_broadcast()</i>	<i>sched_get_priority_max()</i>
114	<i>dLError()</i>	<i>pthread_cond_destroy()</i>	<i>sched_get_priority_min()</i>
115	<i>dlopen()</i>	<i>pthread_cond_init()</i>	<i>sched_getparam()</i>
116	<i>dlsym()</i>	<i>pthread_cond_signal()</i>	<i>sched_getscheduler()</i>
117	<i>fdatasync()</i>	<i>pthread_cond_timedwait()</i>	<i>sched_rr_get_interval()</i>
118	<i>flockfile()</i>	<i>pthread_cond_wait()</i>	<i>sched_setparam()</i>
119	<i>fseeko()</i>	<i>pthread_condattr_destroy()</i>	<i>sched_setscheduler()</i>
120	<i>ftello()</i>	<i>pthread_condattr_getpshared()</i>	<i>sched_yield()</i>
121	<i>ftrylockfile()</i>	<i>pthread_condattr_init()</i>	<i>sem_close()</i>
122	<i>funlockfile()</i>	<i>pthread_condattr_setpshared()</i>	<i>sem_destroy()</i>
123	<i>fwide()</i>	<i>pthread_create()</i>	<i>sem_getvalue()</i>
124	<i>fwprintf()</i>	<i>pthread_detach()</i>	<i>sem_init()</i>
125	<i>fwscanf()</i>	<i>pthread_equal()</i>	<i>sem_open()</i>
126	<i>getc_unlocked()</i>	<i>pthread_exit()</i>	<i>sem_post()</i>
127	<i>getchar_unlocked()</i>	<i>pthread_getconcurrency()</i>	<i>sem_trywait()</i>
128	<i>getgrgid_r()</i>	<i>pthread_getschedparam()</i>	<i>sem_unlink()</i>
129	<i>getgrnam_r()</i>	<i>pthread_getspecific()</i>	<i>sem_wait()</i>
130	<i>getlogin_r()</i>	<i>pthread_join()</i>	<i>shm_open()</i>
131	<i>getpwnam_r()</i>	<i>pthread_key_create()</i>	<i>shm_unlink()</i>
132	<i>getpwuid_r()</i>	<i>pthread_key_delete()</i>	<i>sigqueue()</i>
133	<i>gmtime_r()</i>	<i>pthread_kill()</i>	<i>sigtimedwait()</i>
134	<i>lio_listio()</i>	<i>pthread_mutex_destroy()</i>	<i>sigwait()</i>
135	<i>localtime_r()</i>	<i>pthread_mutex_getprioceiling()</i>	<i>sigwaitinfo()</i>
136	<i>mbrlen()</i>	<i>pthread_mutex_init()</i>	<i>snprintf()</i>
137	<i>mbrtowc()</i>	<i>pthread_mutex_lock()</i>	<i>strtok_r()</i>
138	<i>mbsinit()</i>	<i>pthread_mutex_setprioceiling()</i>	<i>swprintf()</i>
139	<i>mbsrtowcs()</i>	<i>pthread_mutex_trylock()</i>	<i>swscanf()</i>
140	<i>mlock()</i>	<i>pthread_mutex_unlock()</i>	<i>timer_create()</i>
141	<i>mlockall()</i>	<i>pthread_mutexattr_destroy()</i>	<i>timer_delete()</i>
142	<i>mq_close()</i>	<i>pthread_mutexattr_getprioceiling()</i>	<i>timer_getoverrun()</i>
143	<i>mq_getattr()</i>	<i>pthread_mutexattr_getprotocol()</i>	<i>timer_gettime()</i>
144	<i>mq_notify()</i>	<i>pthread_mutexattr_getpshared()</i>	<i>timer_settime()</i>
145	<i>mq_open()</i>	<i>pthread_mutexattr_gettype()</i>	<i>towctrans()</i>

146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167

New Functions and Headers in Issue 5

<i>mq_receive()</i>	<i>pthread_mutexattr_init()</i>	<i>ttyname_r()</i>
<i>mq_send()</i>	<i>pthread_mutexattr_setprioceiling()</i>	<i>vfwprintf()</i>
<i>mq_setattr()</i>	<i>pthread_mutexattr_setprotocol()</i>	<i>vsnprintf()</i>
<i>mq_unlink()</i>	<i>pthread_mutexattr_setpshared()</i>	<i>vswprintf()</i>
<i>munlock()</i>	<i>pthread_mutexattr_settype()</i>	<i>vwprintf()</i>
<i>munlockall()</i>	<i>pthread_once()</i>	<i>wcrtomb()</i>
<i>nanosleep()</i>	<i>pthread_rwlock_destroy()</i>	<i>wcsrtombs()</i>
<i>pread()</i>	<i>pthread_rwlock_init()</i>	<i>wcsstr()</i>
<i>pthread_atfork()</i>	<i>pthread_rwlock_rdlock()</i>	<i>wctob()</i>
<i>pthread_attr_destroy()</i>	<i>pthread_rwlock_tryrdlock()</i>	<i>wctrans()</i>
<i>pthread_attr_getdetachstate()</i>	<i>pthread_rwlock_trywrlock()</i>	<i>wmemchr()</i>
<i>pthread_attr_getguardsize()</i>	<i>pthread_rwlock_unlock()</i>	<i>wmemcmp()</i>
<i>pthread_attr_getinheritsched()</i>	<i>pthread_rwlock_wrlock()</i>	<i>wmemcpy()</i>
<i>pthread_attr_getschedparam()</i>	<i>pthread_rwlockattr_destroy()</i>	<i>wmemmove()</i>
<i>pthread_attr_getschedpolicy()</i>	<i>pthread_rwlockattr_getpshared()</i>	<i>wmemset()</i>
<i>pthread_attr_getscope()</i>	<i>pthread_rwlockattr_init()</i>	<i>wprintf()</i>
<i>pthread_attr_getstackaddr()</i>	<i>pthread_rwlockattr_setpshared()</i>	<i>wscanf()</i>
<aio.h>	<iso646.h>	<sched.h>
<dlfcn.h>	<mqueue.h>	<semaphore.h>
<inttypes.h>	<pthread.h>	<wctype.h>

168 **1.5.3 New Features in Issue 6**169 ***Notes to Reviewers***170 *This section with side shading will not appear in the final copy. - Ed.*171 A table listing new functions, headers, etc. since the ISO POSIX-1:1996 standard will be added
172 here in a future draft.

173 1.6 Terminology

174 This section appears in the Base Definitions volume of IEEE Std. 1003.1-200x, but is repeated
175 here for convenience:

176 For the purposes of IEEE Std. 1003.1-200x, the following terminology definitions apply:

177 **can**

178 Describes a permissible optional feature or behavior available to the user or application. The
179 feature or behavior is mandatory for an implementation that conforms to
180 IEEE Std. 1003.1-200x. An application can rely on the existence of the feature or behavior.

181 **implementation-defined**

182 Describes a value or behavior that is not defined by IEEE Std. 1003.1-200x but is selected by
183 an implementor. The value or behavior may vary among implementations that conform to
184 IEEE Std. 1003.1-200x. An application should not rely on the existence of the value or
185 behavior. An application that relies on such a value or behavior cannot be assured to be
186 portable across conforming implementations.

187 The implementor shall document such a value or behavior so that it can be used correctly
188 by an application.

189 **legacy**

190 Describes a feature or behavior that is being retained for compatibility with older
191 applications, but which has limitations which make it inappropriate for developing portable
192 applications. New applications should use alternative means of obtaining equivalent
193 functionality.

194 **may**

195 Describes a feature or behavior that is optional for an implementation that conforms to
196 IEEE Std. 1003.1-200x. An application should not rely on the existence of the feature or
197 behavior. An application that relies on such a feature or behavior cannot be assured to be
198 portable across conforming implementations.

199 To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

200 **shall**

201 For an implementation that conforms to IEEE Std. 1003.1-200x, describes a feature or
202 behavior that is mandatory. An application can rely on the existence of the feature or
203 behavior.

204 For an application or user, describes a behavior that is mandatory.

205 **should**

206 For an implementation that conforms to IEEE Std. 1003.1-200x, describes a feature or
207 behavior that is recommended but not mandatory. An application should not rely on the
208 existence of the feature or behavior. An application that relies on such a feature or behavior
209 cannot be assured to be portable across conforming implementations.

210 For an application, describes a feature or behavior that is recommended programming
211 practice for optimum portability.

212 **undefined**

213 Describes the nature of a value or behavior not defined by IEEE Std. 1003.1-200x which
214 results from use of an invalid program construct or invalid data input.

215 The value or behavior may vary among implementations that conform to
216 IEEE Std. 1003.1-200x. An application should not rely on the existence or validity of the
217 value or behavior. An application that relies on any particular value or behavior cannot be

218 assured to be portable across conforming implementations.

219 **unspecified**

220 Describes the nature of a value or behavior not specified by IEEE Std. 1003.1-200x which
221 results from use of a valid program construct or valid data input.

222 The value or behavior may vary among implementations that conform to
223 IEEE Std. 1003.1-200x. An application should not rely on the existence or validity of the
224 value or behavior. An application that relies on any particular value or behavior cannot be
225 assured to be portable across conforming implementations.

226 **1.7 Definitions**

227 Concepts and definitions are defined in the Base Definitions volume of IEEE Std. 1003.1-200x.

228 1.8 Relationship to Other Formal Standards

229 Great care has been taken to ensure that this volume of IEEE Std. 1003.1-200x is fully aligned
230 with the following standards:

231 ISO C (1999)

232 ISO/IEC 9899: 1999, Programming Languages — C.

233 Parts of the ISO/IEC 9899: 1999 standard (hereinafter referred to as the ISO C standard) are
234 referenced to describe requirements also mandated by this volume of IEEE Std. 1003.1-200x.
235 Some functions and headers included within this volume of IEEE Std. 1003.1-200x have a version
236 in the ISO C standard; in this case *CX* markings are added as appropriate to show where the
237 ISO C standard has been extended. Any conflict between this volume of IEEE Std. 1003.1-200x
238 and the ISO C standard is unintentional.

239 This volume of IEEE Std. 1003.1-200x also allows, but does not require, mathematics functions to
240 support IEEE Std. 754-1985 and IEEE Std. 854-1987.

241 1.9 Portability

242 Some of the utilities in the Shell and Utilities volume of IEEE Std. 1003.1-200x and functions in
 243 the System Interfaces volume of IEEE Std. 1003.1-200x describe functionality that might not be
 244 fully portable to systems meeting the requirements for POSIX conformance (see the Base
 245 Definitions volume of IEEE Std. 1003.1-200x, Chapter 2, Conformance).

246 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in
 247 the margin identifies the nature of the option, extension, or warning (see Section 1.9.1). For
 248 maximum portability, an application should avoid such functionality.

249 1.9.1 Codes

250 Margin codes and their meanings are listed in the Base Definitions volume of
 251 IEEE Std. 1003.1-200x, but are repeated here for convenience:

252 ADV **Advisory Information**

253 The functionality described is optional. The functionality described is also an extension to the
 254 ISO C standard.

255 Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section.
 256 Where additional semantics apply to a function, the material is identified by use of the ADV
 257 margin legend.

258 AIO **Asynchronous Input and Output**

259 The functionality described is optional. The functionality described is also an extension to the
 260 ISO C standard.

261 Where applicable, functions are marked with the AIO margin legend in the SYNOPSIS section.
 262 Where additional semantics apply to a function, the material is identified by use of the AIO
 263 margin legend.

264 BAR **Barriers**

265 The functionality described is optional. The functionality described is also an extension to the
 266 ISO C standard.

267 Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.
 268 Where additional semantics apply to a function, the material is identified by use of the BAR
 269 margin legend.

270 BE **Batch Environment Services and Utilities**

271 The functionality described is optional.

272 Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.
 273 Where additional semantics apply to a utility, the material is identified by use of the BE margin
 274 legend.

275 CD **C-Language Development Utilities**

276 The functionality described is optional.

277 Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.
 278 Where additional semantics apply to a utility, the material is identified by use of the CD margin
 279 legend.

280 CPT **Process CPU-Time Clocks**

281 The functionality described is optional. The functionality described is also an extension to the
 282 ISO C standard.

283 Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.
 284 Where additional semantics apply to a function, the material is identified by use of the CPT

285 margin legend.

286 CS **Clock Selection**
287 The functionality described is optional. The functionality described is also an extension to the
288 ISO C standard.

289 Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section.
290 Where additional semantics apply to a function, the material is identified by use of the CS
291 margin legend.

292 CX **Extension to the ISO C standard**
293 The functionality described is an extension to the ISO C standard. Application writers may
294 make use of an extension as it is supported on all IEEE Std. 1003.1-200x-conforming systems.

295 FD **FORTTRAN Development Utilities**
296 The functionality described is optional.

297 Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.
298 Where additional semantics apply to a utility, the material is identified by use of the FD margin
299 legend.

300 FR **FORTTRAN Runtime Utilities**
301 The functionality described is optional.

302 Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.
303 Where additional semantics apply to a utility, the material is identified by use of the FR margin
304 legend.

305 FSC **File Synchronization**
306 The functionality described is optional. The functionality described is also an extension to the
307 ISO C standard.

308 Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.
309 Where additional semantics apply to a function, the material is identified by use of the FSC
310 margin legend.

311 IP6 **IPV6**
312 The functionality described is optional. The functionality described is also an extension to the
313 ISO C standard.

314 Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.
315 Where additional semantics apply to a function, the material is identified by use of the IP6
316 margin legend.

317 MAN **Mandatory in the Next Draft**
318 This is an interim draft code used to aid reviewers during the development of
319 IEEE Std. 1003.1-200x. It denotes a feature that was previously an option or extension that is
320 being brought into the mandatory base functionality. This margin code will be removed from the
321 final draft.

322 MF **Memory Mapped Files**
323 The functionality described is optional. The functionality described is also an extension to the
324 ISO C standard.

325 Where applicable, functions are marked with the MF margin legend in the SYNOPSIS section.
326 Where additional semantics apply to a function, the material is identified by use of the MF
327 margin legend.

328 ML **Process Memory Locking**
329 The functionality described is optional. The functionality described is also an extension to the

330 ISO C standard.

331 Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.
 332 Where additional semantics apply to a function, the material is identified by use of the ML
 333 margin legend.

334 MLR **Range Memory Locking**
 335 The functionality described is optional. The functionality described is also an extension to the
 336 ISO C standard.

337 Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
 338 Where additional semantics apply to a function, the material is identified by use of the MLR
 339 margin legend.

340 MON **Monotonic Clock**
 341 The functionality described is optional. The functionality described is also an extension to the
 342 ISO C standard.

343 Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.
 344 Where additional semantics apply to a function, the material is identified by use of the MON
 345 margin legend.

346 MPR **Memory Protection**
 347 The functionality described is optional. The functionality described is also an extension to the
 348 ISO C standard.

349 Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section.
 350 Where additional semantics apply to a function, the material is identified by use of the MPR
 351 margin legend.

352 MSG **Message Passing**
 353 The functionality described is optional. The functionality described is also an extension to the
 354 ISO C standard.

355 Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.
 356 Where additional semantics apply to a function, the material is identified by use of the MSG
 357 margin legend.

358 OB **Obsolescent**
 359 The functionality described may be withdrawn in a future version of this volume of
 360 IEEE Std. 1003.1-200x. Strictly Conforming POSIX Applications and Strictly Conforming XSI
 361 Applications shall not use obsolescent features.

362 OF **Output Format Incompletely Specified**
 363 The functionality described is an XSI extension. The format of the output produced by the utility
 364 is not fully specified. It is therefore not possible to post-process this output in a consistent
 365 fashion. Typical problems include unknown length of strings and unspecified field delimiters.

366 OH **Optional Header**
 367 In the SYNOPSIS section of some interfaces in the System Interfaces volume of
 368 IEEE Std. 1003.1-200x an included header is marked as in the following example:

369 OH `#include <sys/types.h>`
 370 `#include <grp.h>`
 371 `struct group *getgrnam(const char *name);`

372 This indicates that the marked header is not required on XSI-conformant systems.

373	PIO	Prioritized Input and Output
374		The functionality described is optional. The functionality described is also an extension to the
375		ISO C standard.
376		Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.
377		Where additional semantics apply to a function, the material is identified by use of the PIO
378		margin legend.
379	PS	Process Scheduling
380		The functionality described is optional. The functionality described is also an extension to the
381		ISO C standard.
382		Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.
383		Where additional semantics apply to a function, the material is identified by use of the PS
384		margin legend.
385	RTS	Realtime Signals Extension
386		The functionality described is optional. The functionality described is also an extension to the
387		ISO C standard.
388		Where applicable, functions are marked with the RTS margin legend in the SYNOPSIS section.
389		Where additional semantics apply to a function, the material is identified by use of the RTS
390		margin legend.
391	SD	Software Development Utilities
392		The functionality described is optional.
393		Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.
394		Where additional semantics apply to a utility, the material is identified by use of the SD margin
395		legend.
396	SEM	Semaphores
397		The functionality described is optional. The functionality described is also an extension to the
398		ISO C standard.
399		Where applicable, functions are marked with the SEM margin legend in the SYNOPSIS section.
400		Where additional semantics apply to a function, the material is identified by use of the SEM
401		margin legend.
402	SHM	Shared Memory Objects
403		The functionality described is optional. The functionality described is also an extension to the
404		ISO C standard.
405		Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.
406		Where additional semantics apply to a function, the material is identified by use of the SHM
407		margin legend.
408	SIO	Synchronized Input and Output
409		The functionality described is optional. The functionality described is also an extension to the
410		ISO C standard.
411		Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.
412		Where additional semantics apply to a function, the material is identified by use of the SIO
413		margin legend.
414	SPI	Spin Locks
415		The functionality described is optional. The functionality described is also an extension to the
416		ISO C standard.

417 Where applicable, functions are marked with the SPI margin legend in the SYNOPSIS section.
418 Where additional semantics apply to a function, the material is identified by use of the SPI
419 margin legend.

420 SPN **Spawn**
421 The functionality described is optional. The functionality described is also an extension to the
422 ISO C standard.

423 Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.
424 Where additional semantics apply to a function, the material is identified by use of the SPN
425 margin legend.

426 SS **Process Sporadic Server**
427 The functionality described is optional. The functionality described is also an extension to the
428 ISO C standard.

429 Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.
430 Where additional semantics apply to a function, the material is identified by use of the SS
431 margin legend.

432 TCT **Thread CPU-Time Clocks**
433 The functionality described is optional. The functionality described is also an extension to the
434 ISO C standard.

435 Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.
436 Where additional semantics apply to a function, the material is identified by use of the TCT
437 margin legend.

438 THR **Threads**
439 The functionality described is optional. The functionality described is also an extension to the
440 ISO C standard.

441 Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section.
442 Where additional semantics apply to a function, the material is identified by use of the THR
443 margin legend.

444 TMO **Timeouts**
445 The functionality described is optional. The functionality described is also an extension to the
446 ISO C standard.

447 Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section.
448 Where additional semantics apply to a function, the material is identified by use of the TMO
449 margin legend.

450 TMR **Timers**
451 The functionality described is optional. The functionality described is also an extension to the
452 ISO C standard.

453 Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section.
454 Where additional semantics apply to a function, the material is identified by use of the TMR
455 margin legend.

456 TPI **Threads Priority Inheritance**
457 The functionality described is optional. The functionality described is also an extension to the
458 ISO C standard.

459 Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
460 Where additional semantics apply to a function, the material is identified by use of the TPI
461 margin legend.

462 TPP **Thread Priority Protection**
463 The functionality described is optional. The functionality described is also an extension to the
464 ISO C standard.

465 Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
466 Where additional semantics apply to a function, the material is identified by use of the TPP
467 margin legend.

468 TPS **Thread Execution Scheduling**
469 The functionality described is optional. The functionality described is also an extension to the
470 ISO C standard.

471 Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
472 Where additional semantics apply to a function, the material is identified by use of the TPS
473 margin legend.

474 TRC **Trace**
475 The functionality described is optional. The functionality described is also an extension to the
476 ISO C standard.

477 Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
478 Where additional semantics apply to a function, the material is identified by use of the TRC
479 margin legend.

480 TEF **Trace Event Filter**
481 The functionality described is optional. The functionality described is also an extension to the
482 ISO C standard.

483 Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.
484 Where additional semantics apply to a function, the material is identified by use of the TEF
485 margin legend.

486 TRL **Trace Log**
487 The functionality described is optional. The functionality described is also an extension to the
488 ISO C standard.

489 Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
490 Where additional semantics apply to a function, the material is identified by use of the TRL
491 margin legend.

492 TRI **Trace Inherit**
493 The functionality described is optional. The functionality described is also an extension to the
494 ISO C standard.

495 Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
496 Where additional semantics apply to a function, the material is identified by use of the TRI
497 margin legend.

498 TSA **Thread Stack Address Attribute**
499 The functionality described is optional. The functionality described is also an extension to the
500 ISO C standard.

501 Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
502 Where additional semantics apply to a function, the material is identified by use of the TSA
503 margin legend.

504 TSF **Thread-Safe Functions**
505 The functionality described is optional. The functionality described is also an extension to the
506 ISO C standard.

507 Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section.
508 Where additional semantics apply to a function, the material is identified by use of the TSF
509 margin legend.

510 TSH **Thread Process-Shared Synchronization**
511 The functionality described is optional. The functionality described is also an extension to the
512 ISO C standard.

513 Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.
514 Where additional semantics apply to a function, the material is identified by use of the TSH
515 margin legend.

516 TSP **Thread Sporadic Server**
517 The functionality described is optional. The functionality described is also an extension to the
518 ISO C standard.

519 Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.
520 Where additional semantics apply to a function, the material is identified by use of the TSP
521 margin legend.

522 TSS **Thread Stack Address Size**
523 The functionality described is optional. The functionality described is also an extension to the
524 ISO C standard.

525 Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.
526 Where additional semantics apply to a function, the material is identified by use of the TSS
527 margin legend.

528 TYM **Typed Memory Objects**
529 The functionality described is optional. The functionality described is also an extension to the
530 ISO C standard.

531 Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.
532 Where additional semantics apply to a function, the material is identified by use of the TYM
533 margin legend.

534 UN **Possibly Unsupportable Feature**
535 The functionality described is an XSI extension. It need not be possible to implement the
536 required functionality (as defined) on all conformant systems and the functionality need not be
537 present. This may, for example, be the case where the conformant system is hosted and the
538 underlying system provides the service in an alternative way.

539 UP **User Portability Utilities**
540 The functionality described is optional.

541 Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.
542 Where additional semantics apply to a utility, the material is identified by use of the UP margin
543 legend.

544 XSI **Extension**
545 The functionality described is an XSI extension. Functionality marked XSI is also an extension to
546 the ISO C standard. Application writers may confidently make use of an extension on all
547 systems supporting the X/Open System Interfaces Extension.

548 If an entire SYNOPSIS section is shaded and marked with one XSI, all the functionality described
549 in that reference page is an extension. See the Base Definitions volume of IEEE Std. 1003.1-200x,
550 Section 3.441, XSI.

551 XSR **XSI STREAMS**
552 The functionality described is optional. The functionality described is also an extension to the
553 ISO C standard.

554 Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section.
555 Where additional semantics apply to a function, the material is identified by use of the XSR
556 margin legend.

557 1.10 Format of Entries

558 The entries in Chapter 3 are based on a common format as follows. The only sections relating to
559 conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS sections.

560 NAME

561 This section gives the name or names of the entry and briefly states its purpose.

562 SYNOPSIS

563 This section summarizes the use of the entry being described. If it is necessary to
564 include a header to use this function, the names of such headers are shown, for
565 example:

```
566 #include <stdio.h>
```

567 DESCRIPTION

568 This section describes the functionality of the function or header.

569 RETURN VALUE

570 This section indicates the possible return values, if any.

571 If the implementation can detect errors, “successful completion” means that no error
572 has been detected during execution of the function. If the implementation does detect
573 an error, the error is indicated.

574 For functions where no errors are defined, “successful completion” means that if the
575 implementation checks for errors, no error has been detected. If the implementation can
576 detect errors, and an error is detected, the indicated return value is returned and *errno*
577 may be set.

578 ERRORS

579 This section gives the symbolic names of the values returned in *errno* if an error occurs.
580 “No errors are defined” means that values and usage of *errno*, if any, depend on the
581 implementation.

582 EXAMPLES

583 This section is non-normative.

584 This section gives examples of usage, where appropriate. In the event of conflict
585 between an example and a normative part of this volume of IEEE Std. 1003.1-200x, the
586 normative material is to be taken as correct.

587 APPLICATION USAGE

588 This section is non-normative.

589 This section gives warnings and advice to application writers about the entry. In the
590 event of conflict between warnings and advice and a normative part of this volume of
591 IEEE Std. 1003.1-200x, the normative material is to be taken as correct.

592 RATIONALE

593 This section is non-normative.

594 This section contains historical information concerning the contents of this volume of
595 IEEE Std. 1003.1-200x and why features were included or discarded by the standard
596 developers.

597 FUTURE DIRECTIONS

598 This section is non-normative.

599 This section provides comments which should be used as a guide to current thinking;
600 there is not necessarily a commitment to adopt these future directions.

601 **SEE ALSO**

602 This section is non-normative.

603 This section gives references to related information.

604 **CHANGE HISTORY**

605 This section is non-normative.

606 This section shows the derivation of the entry and any significant changes that have
607 been made to it.

608

609 This chapter covers information that is relevant to all the functions specified in Chapter 3 and
610 the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 13, Headers.

611 **2.1 Use and Implementation of Functions**

612 Each of the following statements shall apply unless explicitly stated otherwise in the detailed
613 descriptions that follow:

- 614 1. If an argument to a function has an invalid value (such as a value outside the domain of
615 the function, or a pointer outside the address space of the program, or a null pointer), the
616 behavior is undefined.
- 617 2. Any function declared in a header may also be implemented as a macro defined in the
618 header, so a library function should not be declared explicitly if its header is included. Any
619 macro definition of a function can be suppressed locally by enclosing the name of the
620 function in parentheses, because the name is then not followed by the left parenthesis that
621 indicates expansion of a macro function name. For the same syntactic reason, it is
622 permitted to take the address of a library function even if it is also defined as a macro. The
623 use of the C-language **#undef** construct to remove any such macro definition shall also
624 ensure that an actual function is referred to.
- 625 3. Any invocation of a library function that is implemented as a macro shall expand to code
626 that evaluates each of its arguments exactly once, fully protected by parentheses where
627 necessary, so it is generally safe to use arbitrary expressions as arguments. Likewise, those
628 function-like macros described in the following sections may be invoked in an expression
629 anywhere a function with a compatible return type could be called.
- 630 4. Provided that a library function can be declared without reference to any type defined in a
631 header, it is also permissible to declare the function, either explicitly or implicitly, and use
632 it without including its associated header.
- 633 5. If a function that accepts a variable number of arguments is not declared (explicitly or by
634 including its associated header), the behavior is undefined.

635 2.2 The Compilation Environment

636 2.2.1 POSIX.1 Symbols

637 Certain symbols in this volume of IEEE Std. 1003.1-200x are defined in headers (see the Base
638 Definitions volume of IEEE Std. 1003.1-200x, Chapter 13, Headers). Some of those headers could
639 also define other symbols than those defined by this volume of IEEE Std. 1003.1-200x, potentially
640 conflicting with symbols used by the application. Also, this volume of IEEE Std. 1003.1-200x
641 defines symbols that are not permitted by other standards to appear in those headers without
642 some control on the visibility of those symbols.

643 Symbols called feature test macros are used to control the visibility of symbols that might be
644 included in a header. Implementations, future versions of this volume of IEEE Std. 1003.1-200x,
645 and other standards may define additional feature test macros.

646 In the compilation of an application that **#defines** a feature test macro specified by
647 IEEE Std. 1003.1-200x, no header defined by IEEE Std. 1003.1-200x shall be included prior to the
648 definition of the feature test macro. This restriction also applies to any implementation-
649 provided header in which these feature test macros are used. If the definition of the macro does
650 not precede the **#include**, the result is undefined.

651 Feature test macros shall begin with the underscore character ('_').

652 2.2.1.1 The `_POSIX_C_SOURCE` Feature Test Macro

653 A POSIX-conforming application should ensure that the feature test macro `_POSIX_C_SOURCE`
654 is defined before inclusion of any header.

655 When an application includes a header described by this volume of IEEE Std. 1003.1-200x, and
656 when this feature test macro is defined to have at least the value `200xMML`:

- 657 1. All symbols required by this volume of IEEE Std. 1003.1-200x to appear when the header is
658 included shall be made visible.
- 659 2. Symbols that are explicitly permitted, but not required, by this volume of
660 IEEE Std. 1003.1-200x to appear in that header (including those in reserved name spaces)
661 may be made visible.
- 662 3. Additional symbols not required or explicitly permitted by this volume of
663 IEEE Std. 1003.1-200x to be in that header shall not be made visible, except when enabled
664 by another feature test macro or by having defined `_POSIX_C_SOURCE` with a value
665 larger than `200xxxL`.

666 Identifiers in this volume of IEEE Std. 1003.1-200x may only be undefined using the **#undef**
667 directive as described in Section 2.1 (on page 511) or Section 2.2.2 (on page 513). These **#undef**
668 directives shall follow all **#include** directives of any header in this volume of
669 IEEE Std. 1003.1-200x.

670 2.2.1.2 The `_XOPEN_SOURCE` Feature Test Macro

671 XSI An XSI-conforming application should ensure that the feature test macro `_XOPEN_SOURCE` is
672 defined with the value `600` before inclusion of any header. This is needed to enable the
673 functionality described in Section 2.2.1.1 and in addition to enable the X/Open System Interfaces
674 Extension.

675 Since this volume of IEEE Std. 1003.1-200x is aligned with the ISO C standard, and since all
676 functionality enabled by `_POSIX_C_SOURCE` set greater than zero and less than or equal to
677 `200xxxL` should be enabled by `_XOPEN_SOURCE` set equal to `600`, there should be no need to

678 define either `_POSIX_SOURCE` or `_POSIX_C_SOURCE` if `_XOPEN_SOURCE` is so defined.
679 Therefore, if `_XOPEN_SOURCE` is set equal to 600 and `_POSIX_SOURCE` is defined, or
680 `_POSIX_C_SOURCE` is set greater than zero and less than or equal to 200xxxL, the behavior is
681 the same as if only `_XOPEN_SOURCE` is defined and set equal to 600. However, should
682 `_POSIX_C_SOURCE` be set to a value greater than 200xxxL, the behavior is undefined.

683 2.2.2 The Name Space

684 All identifiers in this volume of IEEE Std. 1003.1-200x, except *environ*, are defined in at least one
685 of the headers, as shown in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 13,
686 XSI Headers. When `_XOPEN_SOURCE` or `_POSIX_C_SOURCE` is defined, each header defines or
687 declares some identifiers, potentially conflicting with identifiers used by the application. The set
688 of identifiers visible to the application consists of precisely those identifiers from the header
689 pages of the included headers, as well as additional identifiers reserved for the implementation.
690 In addition, some headers may make visible identifiers from other headers as indicated on the
691 relevant header pages.

692 Implementations may also add members to a structure or union without controlling the
693 visibility of those members with a feature test macro, as long as a user-defined macro with the
694 same name cannot interfere with the correct interpretation of the program. The identifiers
695 reserved for use by the implementation are described below:

- 696 1. Each identifier with external linkage described in the header section is reserved for use as
697 an identifier with external linkage if the header is included.
- 698 2. Each macro name described in the header section is reserved for any use if the header is
699 included.
- 700 3. Each identifier with file scope described in the header section is reserved for use as an
701 identifier with file scope in the same name space if the header is included.

702 The prefixes `posix_`, `POSIX_`, and `_POSIX_` are reserved for use by IEEE Std. 1003.1-200x and
703 other POSIX standards. Implementations may add symbols to the headers shown in the
704 following table, provided the identifiers for those symbols begin with the corresponding
705 reserved prefixes in the following table, and do not use the reserved prefixes `posix_`, `POSIX_`, or
706 `_POSIX_`.

	Header	Prefix	Suffix	Complete Name
710 AIO	<aio.h>	aio_, lio_, AIO_, LIO_		
711	<arpa/inet.h>	in_, inet_		
712	<ctype.h>	to[a-z], is[a-z]		
713	<dirent.h>	d_		
714	<errno.h>	E[0-9], E[A-Z]		
715	<fcntl.h>	l_		
716	<glob.h>	gl_		
717	<grp.h>	gr_		
718	<inttypes.h>	int[0-9a-z_]_t, uint[0-9a-z_]_t		
719	<limits.h>		_MAX, _MIN	
720	<locale.h>	LC_[A-Z]		
721 MSG	<mqueue.h>	mq_, MQ_		
722 XSI	<ndbm.h>	dbm_		
723	<netdb.h>	h_, n_, p_, s_		
724	<net/if.h>	if_		
725	<netinet/in.h>	in_, ip_, s_, sin_		
726 XSI	<poll.h>	pd_, ph_, ps_		
727	<pthread.h>	pthread_, PTHREAD_		
728	<pwd.h>	pw_		
729	<regex.h>	re_, rm_		
730 PS	<sched.h>	sched_, SCHED_		
731 SEM	<semaphore.h>	sem_, SEM_		
732	<signal.h>	sa_, uc_, SIG[A-Z], SIG_[A-Z]		
733 XSI		ss_, sv_		
734 RTS		si_, SI_, sigev_, SIGEV_, sival_		
735 XSI	<stropts.h>	bi_, ic_, l_, sl_, str_		
736	<stdint.h>	int[0-9a-z_]_t, uint[0-9a-z_]_t		
737	<stdlib.h>	str[a-z]		
738	<string.h>	str[a-z], mem[a-z], wcs[a-z]		
739 XSI	<sys/ipc.h>	ipc_		key, pad, seq
740 MF	<sys/mman.h>	shm_, MAP_, MCL_, MS_, PROT_		
741 XSI	<sys/msg.h>	msg		msg
742 XSI	<sys/resource.h>	rlim_, ru_		
743 XSI	<sys/sem.h>	sem		sem
744 XSI	<sys/shm.h>	shm		
745	<sys/socket.h>	_ss, sa_, if_, ifc_, ifru_, infu_, ifra_, msg_, cmsg_, l_		
746				
747	<sys/stat.h>	st_		
748 XSI	<sys/statvfs.h>	f_		
749	<sys/time.h>	fds_, it_, tv_, FD_		
750	<sys/times.h>	tms_		
751 XSI	<sys/uio.h>	iov_		
752	<sys/un.h>	sun_		
753	<sys/utsname.h>	uts_		
754 XSI	<sys/wait.h>	si_, W[A-Z], P_		
755	<termios.h>	c_		

756
 757
 758
 759
 760 TMR
 761 TMR
 762 XSI
 763 XSI
 764
 765 XSI
 766
 767
 768
 769
 770

Header	Prefix	Suffix	Complete Name
<time.h>	tm_		
	clock_, timer_, it_, tv_, CLOCK_, TIMER_		
<ucontext.h>	uc_, ss_		
<ulimit.h>	UL_		
<utime.h>	utim_		
<utmpx.h>	ut_	_LVL, _TIME, _PROCESS	
<wchar.h>	wcs[a-z]		
<wctype.h>	is[a-z], to[a-z]		
<wordexp.h>	we_		
ANY header	POSIX_, _POSIX_, posix_	_t	

771 **Note:** The notation [A-Z] indicates any uppercase letter in the portable character set. The
 772 notation [a-z] indicates any lowercase letter in the portable character set. Commas
 773 and spaces in the lists of prefixes and complete names in the above table are not part
 774 of any prefix or complete name.

775 If any header in the following table is included, macros with the prefixes shown may be defined. |
 776 After the last inclusion of a given header, an application may use identifiers with the
 777 corresponding prefixes for its own purpose, provided their use is preceded by an **#undef** of the
 778 corresponding macro.

779
780
781 XSI
782
783 XSI
784
785 XSI
786
787
788 XSI
789
790
791
792 XSI
793 XSI
794
795
796 XSI
797
798
799 XSI
800 XSI
801 XSI
802 XSI
803 XSI
804 XSI
805 XSI
806 XSI
807 XSI
808
809 XSI
810 XSI
811 XSI
812 XSI
813
814

Header	Prefix
<dfcn.h>	RTLD_
<fcntl.h>	F_, O_, S_
<fmtmsg.h>	MM_
<fnmatch.h>	FNM_
<ftw.h>	FTW
<glob.h>	GLOB_
<inttypes.h>	PRI[a-z], SCN[a-z]
<ndbm.h>	DBM_
<net/if.h>	IF_
<netinet/in.h>	IMPLINK_, IN_, INADDR_, IP_, IPPORT_, IPPROTO_, SOCK_
<netinet/tcp.h>	TCP_
<nl_types.h>	NL_
<poll.h>	POLL
<regex.h>	REG_
<signal.h>	SA_, SIG_[0-9a-z_], BUS_, CLD_, FPE_, ILL_, POLL_, SEGV_, SI_, SS_, SV_, TRAP_
stdint.h	INT[0-9A-Z_]_MIN, INT[0-9A-Z_]_MAX, INT[0-9A-Z_]_C UINT[0-9A-Z_]_MIN, UINT[0-9A-Z_]_MAX, UINT[0-9A-Z_]_C
<stropts.h>	FLUSH[A-Z], I_, M_, MUXID_R[A-Z], S_, SND[A-Z], STR
<syslog.h>	LOG_
<sys/ipc.h>	IPC_
<sys/mman.h>	PROT_, MAP_, MS_
<sys/msg.h>	MSG[A-Z]
<sys/resource.h>	PRIO_, RLIM_, RLIMIT_, RUSAGE_
<sys/sem.h>	SEM_
<sys/shm.h>	SHM[A-Z], SHM_[A-Z]
<sys/socket.h>	AF_, CMSG_, MSG_, PF_, SCM_, SHUT_, SO
<sys/stat.h>	S_
<sys/statvfs.h>	ST_
<sys/time.h>	FD_, ITIMER_
<sys/uio.h>	IOV_
<sys/wait.h>	BUS_, CLD_, FPE_, ILL_, POLL_, SEGV_, SI_, TRAP_
<termios.h>	V, I, O, TC, B[0-9] (See below.)
<wordexp.h>	WRDE_

815 **Note:** The notation [0–9] indicates any digit. The notation [A–Z] indicates any uppercase
816 letter in the portable character set. The notation [0–9a–z_] indicates any digit, any
817 lowercase letter in the portable character set, or underscore.

818 The following reserved names are used as exact matches for <termios.h>:

819 XSI	CBAUD	EXTB	VDSUSP
820	DEFECHO	FLUSHO	VLNEXT
821	ECHOCTL	LOBLK	VREPRINT
822	ECHOKE	PENDIN	VSTATUS
823	ECHOPRT	SWTCH	VWERASE
824	EXTA	VDISCARD	

- 825 The following identifiers are reserved regardless of the inclusion of headers:
- 826 1. All identifiers that begin with an underscore and either an uppercase letter or another
827 underscore are always reserved for any use by the implementation.
- 828 2. All identifiers that begin with an underscore are always reserved for use as identifiers with
829 file scope in both the ordinary identifier and tag name spaces.
- 830 3. All identifiers in the table below are reserved for use as identifiers with external linkage.
831 Some of these identifiers do not appear in this volume of IEEE Std. 1003.1-200x, but are
832 reserved for future use by the ISO C standard.

833	_Exit	chrtf	exit	fsetpos	mbrtowc	sinhl
834	abort	chrtl	exp	ftell	mbsinit	sinl
835	abs	cqos	exp2	fwide	mbsrtowcs	sprintf
836	acos	cqosf	exp2f	fwprintf	mbstowcs	sqrt
837	acosf	cqosh	exp2l	fwwrite	mbtowc	sqrtf
838	acosh	cqoshf	expf	fwscanf	mjem[a-z]*	sqrtl
839	acoshf	cqoshl	expl	getc	mkttime	srand
840	acoshl	cqosl	expm1	getchar	modf	sscanf
841	acosl	ceil	expm1f	getenv	modff	str[a-z]*
842	acosl	ceilf	expm1l	gets	modfl	strtof
843	asctime	ceilf	fabs	getwc	nan	strtoimax
844	asin	ceil	fabsf	getwchar	nanf	strtold
845	asinf	ceil	fabsl	gmtime	nanl	strtoll
846	asinh	cexp	fclose	hypotf	nearbyint	strtoull
847	asinhf	cexpf	fdim	hypotl	nearbyintf	strtoumax
848	asinhl	cexpl	fdimf	ilogb	nearbyintl	swprintf
849	asinl	cimag	fdiml	ilogbf	nextafterf	swscanf
850	asinl	cimagf	fclearexcept	ilogbl	nextafterl	system
851	atan	cimagl	fgetenv	imaxabs	nexttoward	tan
852	atan2	clearerr	fgetexceptflag	imaxdiv	nexttowardf	tanf
853	atan2f	clock	fgetround	is[a-z]*	nexttowardl	tanh
854	atan2l	clog	fholdexcept	isblank	perror	tanhf
855	atanf	clogf	fEOF	iswblank	pow	tanhl
856	atanf	clogl	fraiseexcept	labs	powf	tanl
857	atanh	conj	ferror	ldexp	powl	tgamma
858	atanh	conjf	fsetenv	ldexpf	printf	tgammaf
859	atanhf	conjl	fsetexceptflag	ldexpl	putc	tgammal
860	atanhl	copysign	fsetround	ldiv	putchar	time
861	atanl	copysignf	fetestexcept	ldiv	puts	tmpfile
862	atanl	copysignl	fupdateenv	lgammaf	putwc	tmpnam
863	atexit	cqs	fflush	lgammal	putwchar	to[a-z]*
864	atof	cqsf	fgetc	llabs	qsort	trunc
865	atoi	cqsh	fgetpos	llrint	raise	truncf
866	atol	cqshf	fgets	llrintf	rand	truncl
867	atoll	cqshl	fgetwc	llrintl	realloc	ungetc
868	bsearch	cqsl	fgetws	llround	remainderf	ungetwc
869	cabs	cpow	floor	llroundf	remainderl	va_end
870	cabsf	cpowf	floorf	llroundl	remove	vfprintf
871	cabsl	cpowl	floorl	lqcaleconv	remquo	vscanf
872	cacos	cproj	fma	lqcaltime	remquof	vfprintf

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

The following table should be made complete by including everything not in the previous table.

- The following identifiers are also reserved for use as identifiers with external linkage:

Table 2-1 XSI Identifiers

a64l	fcntl	getpriority	mknod	regex	sigelse
basename	fchdir	getpwent	mkstemp	remainder	sigset
bcmp	fchmod	getrlimit	mktemp	remque	sigstack
bcopy	fchown	getrusage	mmap	rindex	srandom
brk	fcntl	getsid	mprotect	sbrk	statvfs
bsd_signal	fcntl	getsubopt	rand48	scalb	strcascmp
bzero	ffs	gettimeofday	msync	select	strdup
cbrt	fmtmsg	getutxent	mmap	setcontext	strcascmp
closelog	statvfs	getutxid	nextafter	setgrent	swapcontext
dbm_clearerr	ftime	getutxline	nftw	setitimer	symlink
dbm_close	ftok	getwd	nicp	_setjmp	sync
dbm_delete	truncate	grantpt	openlog	setlogmask	syslog
dbm_error	gcvt	index	poll	setpgp	tcgetsid
dbm_fetch	getcontext	initsstate	ptname	setpriority	truncate
dbm_firstkey	getdate	insque	putmsg	setpwent	ttyslot
dbm_nextkey	getdtablesize	ioctl	putpmsg	setpuid	ualarm
dbm_open	getgrent	isastream	pututxline	setrlimit	unlockpt
dbm_store	getgrgid	killpg	random	setstate	usleep
dirname	gethostid	l64a	readlink	setutxent	utimes
ecvt	getitimer	lchown	readv	sigaltstack	valloc
endgrent	getmsg	lockf	realpath	sighold	vfork
endpwent	getpagesize	_longjmp	re_comp	sigignore	wait3
endservent	getpgid	lstat	re_exec	siginterrupt	waitid
endutxent	getpmsg	makecontext	regcmp	sigpause	wrtv

Table 2-2 Sockets Identifiers

accept	if_freenameindex	recvfrom	shutdown
bind	if_indextoname	recvmsg	socket
connect	if_nameindex	send	socketpair
getpeername	if_nameindex	sendmsg	
getsockname	listen	sendto	
getsockopt	recv	setsockopt	

938

Table 2-3 IP Address Resolution Identifiers

939	endhostent	getipnodebyaddr	getservbyname	inet_netof
940	endnetent	getipnodebyname	getservbyport	inet_network
941	endprotoent	getnameinfo	getservent	inet_ntoa
942	endservent	getnetbyaddr	h_errno	ntohl
943	getaddrinfo	getnetbyname	htonl	ntohs
944	gethostbyaddr	getnetent	htons	sethostent
945	gethostbyname	getprotobyname	inet_addr	setnetent
946	gethostent	getprotobynumber	inet_lnaof	setprotoent
947	gethostname	getprotoent	inet_makeaddr	setservent

948 All the identifiers defined in this volume of IEEE Std. 1003.1-200x that have external linkage are
 949 always reserved for use as identifiers with external linkage.

950 No other identifiers are reserved.

951 Applications shall not declare or define identifiers with the same name as an identifier reserved
 952 in the same context. Since macro names are replaced whenever found, independent of scope and
 953 name space, macro names matching any of the reserved identifier names shall not be defined by
 954 an application if any associated header is included.

955 Except that the effect of each inclusion of `<assert.h>` depends on the definition of `NDEBUG`,
 956 headers may be included in any order, and each may be included more than once in a given
 957 scope, with no difference in effect from that of being included only once.

958 If used, the application shall ensure that a header is included outside of any external declaration
 959 or definition, and it shall be first included before the first reference to any type or macro it
 960 defines, or to any function or object it declares. However, if an identifier is declared or defined in
 961 more than one header, the second and subsequent associated headers may be included after the
 962 initial reference to the identifier. Prior to the inclusion of a header, the application shall not
 963 define any macros with names lexically identical to symbols defined by that header.

964 2.3 Error Numbers

965 Most functions can provide an error number. The means by which each function provides its
966 error numbers is specified in its description.

967 Some functions provide the error number in a variable accessed through the symbol *errno*. The
968 symbol *errno*, defined by including the `<errno.h>` header, is a macro that expands to a
969 modifiable **lvalue** of type **int**.

970 The value of *errno* should only be examined when it is indicated to be valid by a function's return
971 value. No function in this volume of IEEE Std. 1003.1-200x shall set *errno* to zero. For each thread
972 of a process, the value of *errno* shall not be affected by function calls or assignments to *errno* by
973 other threads.

974 Some functions return an error number directly as the function value. These functions return a
975 value of zero to indicate success.

976 If more than one error occurs in processing a function call, any one of the possible errors may be
977 returned, as the order of detection is undefined.

978 Implementations may support additional errors not included in this list, may generate errors
979 included in this list under circumstances other than those described here, or may contain
980 extensions or limitations that shall prevent some errors from occurring. The ERRORS section on
981 each page specifies whether an error shall be returned, or whether it may be returned.
982 Implementations shall not generate a different error number from the ones described here for
983 error conditions described in this volume of IEEE Std. 1003.1-200x, but may generate additional
984 errors unless explicitly disallowed for a particular function.

985 Each implementation shall document, in the conformance document, situations in which each of
986 the optional conditions defined in IEEE Std. 1003.1-200x are detected. The conformance
987 document may also contain statements that one or more of the optional error conditions are not
988 detected.

989 For functions under the Threads option for which [EINTR] is not listed as a possible error
990 condition in this volume of IEEE Std. 1003.1-200x, an implementation shall not return an error
991 code of [EINTR].

992 The following symbolic names identify the possible error numbers, in the context of the
993 functions specifically defined in this volume of IEEE Std. 1003.1-200x; these general descriptions
994 are more precisely defined in the ERRORS sections of the functions that return them. Only these
995 symbolic names should be used in programs, since the actual value of the error number is
996 unspecified. All values listed in this section shall be unique integer constant expressions with
997 type **int** suitable for use in **#if** preprocessing directives, except as noted below. The values for all
998 these names shall be found in the `<errno.h>` header defined in the Base Definitions volume of
999 IEEE Std. 1003.1-200x. The actual values are unspecified by this volume of IEEE Std. 1003.1-200x.

1000 [E2BIG]

1001 Argument list too long. The sum of the number of bytes used by the new process image's
1002 argument list and environment list is greater than the system-imposed limit of {ARG_MAX}
1003 bytes.

1004 or:

1005 Lack of space in an output buffer.

1006 or:

1007 Argument is greater than the system-imposed maximum.

1008	[EACCES]	
1009		Permission denied. An attempt was made to access a file in a way forbidden by its file
1010		access permissions.
1011	[EADDRINUSE]	
1012		Address in use. The specified address is in use.
1013	[EADDRNOTAVAIL]	
1014		Address not available. The specified address is not available from the local system.
1015	[EAFNOSUPPORT]	
1016		Address family not supported. The implementation does not support the specified address
1017		family, or the specified address is not a valid address for the address family of the specified
1018		socket.
1019	[EAGAIN]	
1020		Resource temporarily unavailable. This is a temporary condition and later calls to the same
1021		routine may complete normally.
1022	[EALREADY]	
1023		Connection already in progress. A connection request is already in progress for the specified
1024		socket.
1025	[EBADF]	
1026		Bad file descriptor. A file descriptor argument is out of range, refers to no open file, or a
1027		read (write) request is made to a file that is only open for writing (reading).
1028	[EBADMSG]	
1029	XSR	Bad message. During a <i>read()</i> , <i>getmsg()</i> , or <i>ioctl()</i> I_RECVFD request to a STREAMS device,
1030		a message arrived at the head of the STREAM that is inappropriate for the function
1031		receiving the message.
1032		<i>read()</i> Message waiting to be read on a STREAM is not a data message.
1033		<i>getmsg()</i> A file descriptor was received instead of a control message.
1034		<i>ioctl()</i> Control or data information was received instead of a file descriptor when
1035		I_RECVFD was specified.
1036		or:
1037		Bad Message. The implementation has detected a corrupted message.
1038	[EBUSY]	
1039		Resource busy. An attempt was made to make use of a system resource that is not currently
1040		available, as it is being used by another process in a manner that would have conflicted with
1041		the request being made by this process.
1042	[ECANCELED]	
1043		Operation canceled. The associated asynchronous operation was canceled before
1044		completion.
1045	[ECHILD]	
1046		No child process. A <i>wait()</i> or <i>waitpid()</i> function was executed by a process that had no
1047		existing or unwaited-for child process.
1048	[ECONNABORTED]	
1049		Connection aborted. The connection has been aborted.
1050	[ECONNREFUSED]	
1051		Connection refused. An attempt to connect to a socket was refused because there was no

1052	process listening or because the queue of connection requests was full and the underlying
1053	protocol does not support retransmissions.
1054	[ECONNRESET]
1055	Connection reset. The connection was forcibly closed by the peer.
1056	[EDEADLK]
1057	Resource deadlock would occur. An attempt was made to lock a system resource that
1058	would have resulted in a deadlock situation.
1059	[EDESTADDRREQ]
1060	Destination address required. No bind address was established.
1061	[EDOM]
1062	Domain error. An input argument is outside the defined domain of the mathematical
1063	function (defined in the ISO C standard).
1064	[EDQUOT]
1065	Reserved.
1066	[EEXIST]
1067	File exists. An existing file was mentioned in an inappropriate context; for example, as a
1068	new link name in the <i>link()</i> function.
1069	[EFAULT]
1070	Bad address. The system detected an invalid address in attempting to use an argument of a
1071	call. The reliable detection of this error cannot be guaranteed, and when not detected may
1072	result in the generation of a signal, indicating an address violation, which is sent to the
1073	process.
1074	[EFBIG]
1075	File too large. The size of a file would exceed the maximum file size of an implementation or
1076	offset maximum established in the corresponding file description.
1077	[EHOSTUNREACH]
1078	Host is unreachable. The destination host cannot be reached (probably because the host is
1079	down or a remote router cannot reach it).
1080	[EIDRM]
1081	Identifier removed. Returned during XSI interprocess communication if an identifier has
1082	been removed from the system.
1083	[EILSEQ]
1084	Illegal byte sequence. A wide-character code has been detected that does not correspond to
1085	a valid character, or a byte sequence does not form a valid wide-character code (defined in
1086	the ISO C standard).
1087	[EINPROGRESS]
1088	Operation in progress. This code is used to indicate that an asynchronous operation has not
1089	yet completed.
1090	or:
1091	O_NONBLOCK is set for the socket file descriptor and the connection cannot be
1092	immediately established.
1093	[EINTR]
1094	Interrupted function call. An asynchronous signal was caught by the process during the
1095	execution of an interruptible function. If the signal handler performs a normal return, the
1096	interrupted function call may return this condition (see the Base Definitions volume of

1097	IEEE Std. 1003.1-200x, <signal.h>).
1098	[EINVAL]
1099	Invalid argument. Some invalid argument was supplied; for example, specifying an
1100	undefined signal in a <i>signal()</i> function or a <i>kill()</i> function.
1101	[EIO]
1102	Input/output error. Some physical input or output error has occurred. This error may be
1103	reported on a subsequent operation on the same file descriptor. Any other error-causing
1104	operation on the same file descriptor may cause the [EIO] error indication to be lost.
1105	[EISCONN]
1106	Socket is connected. The specified socket is already connected.
1107	[EISDIR]
1108	Is a directory. An attempt was made to open a directory with write mode specified.
1109	[ELOOP]
1110	Symbolic link loop. A loop exists in symbolic links encountered during path name
1111	resolution. This error may also be returned if more than {SYMLOOP_MAX} symbolic links
1112	are encountered during path name resolution.
1113	[EMFILE]
1114	Too many open files. An attempt was made to open more than the maximum number of
1115	{OPEN_MAX} file descriptors allowed in this process.
1116	[EMLINK]
1117	Too many links. An attempt was made to have the link count of a single file exceed
1118	{LINK_MAX}.
1119	[EMSGSIZE]
1120	Message too large. A message sent on a transport provider was larger than an internal
1121	message buffer or some other network limit.
1122	or:
1123	Inappropriate message buffer length.
1124	[EMULTIHOP]
1125	Reserved.
1126	[ENAMETOOLONG]
1127	File name too long. The length of a path name exceeds {PATH_MAX}, or a path name
1128	component is longer than {NAME_MAX}. This error may also occur when path name
1129	substitution, as a result of encountering a symbolic link during path name resolution,
1130	results in a path name string the size of which exceeds {PATH_MAX}.
1131	[ENETDOWN]
1132	Network is down. The local network interface used to reach the destination is down.
1133	[ENETRESET]
1134	The connection was aborted by the network.
1135	[ENETUNREACH]
1136	Network unreachable. No route to the network is present.
1137	[ENFILE]
1138	Too many files open in system. Too many files are currently open in the system. The system
1139	has reached its predefined limit for simultaneously open files and temporarily cannot accept
1140	requests to open another one.

1141	[ENOBUFS]	
1142	No buffer space available. Insufficient buffer resources were available in the system to	
1143	perform the socket operation.	
1144	XSR [ENODATA]	
1145	No message available. No message is available on the STREAM head read queue.	
1146	[ENODEV]	
1147	No such device. An attempt was made to apply an inappropriate function to a device; for	
1148	example, trying to read a write-only device such as a printer.	
1149	[ENOENT]	
1150	No such file or directory. A component of a specified path name does not exist, or the path	
1151	name is an empty string.	
1152	[ENOEXEC]	
1153	Executable file format error. A request is made to execute a file that, although it has the	
1154	appropriate permissions, is not in the format required by the implementation for executable	
1155	files.	
1156	[ENOLCK]	
1157	No locks available. A system-imposed limit on the number of simultaneous file and record	
1158	locks has been reached and no more are currently available.	
1159	[ENOLINK]	
1160	Reserved.	
1161	[ENOMEM]	
1162	Not enough space. The new process image requires more memory than is allowed by the	
1163	hardware or system-imposed memory management constraints.	
1164	[ENOMSG]	
1165	No message of the desired type. The message queue does not contain a message of the	
1166	required type during XSI interprocess communication.	
1167	[ENOPROTOPT]	
1168	Protocol not available. The protocol option specified to <i>setsockopt()</i> is not supported by the	
1169	implementation.	
1170	[ENOSPC]	
1171	No space left on a device. During the <i>write()</i> function on a regular file or when extending a	
1172	directory, there is no free space left on the device.	
1173	XSR [ENOSR]	
1174	No STREAM resources. Insufficient STREAMS memory resources are available to perform a	
1175	STREAMS-related function. This is a temporary condition; it may be recovered from if other	
1176	processes release resources.	
1177	XSR [ENOSTR]	
1178	Not a STREAM. A STREAM function was attempted on a file descriptor that was not	
1179	associated with a STREAMS device.	
1180	[ENOSYS]	
1181	Function not implemented. An attempt was made to use a function that is not available in	
1182	this implementation.	
1183	[ENOTCONN]	
1184	Socket not connected. The socket is not connected.	

1185	[ENOTDIR]
1186	Not a directory. A component of the specified path name exists, but it is not a directory,
1187	when a directory was expected.
1188	[ENOTEMPTY]
1189	Directory not empty. A directory other than an empty directory was supplied when an
1190	empty directory was expected.
1191	[ENOTSOCK]
1192	Not a socket. The file descriptor does not refer to a socket.
1193	[ENOTSUP]
1194	Not supported. The implementation does not support this feature of the Realtime Option
1195	Group.
1196	[ENOTTY]
1197	Inappropriate I/O control operation. A control function has been attempted for a file or
1198	special file for which the operation is inappropriate.
1199	[ENXIO]
1200	No such device or address. Input or output on a special file refers to a device that does not
1201	exist, or makes a request beyond the capabilities of the device. It may also occur when, for
1202	example, a tape drive is not on-line.
1203	[EOPNOTSUPP]
1204	Operation not supported on socket. The type of socket (address family or protocol) does not
1205	support the requested operation.
1206	[EOVERFLOW]
1207	Value too large to be stored in data type. The user ID or group ID of an IPC or file system
1208	object was too large to be stored into the appropriate member of the caller-provided
1209	structure. This error shall only occur on implementations that support a larger range of user
1210	ID or group ID values than the declared structure member can support. This usually occurs
1211	because the IPC or file system object resides on a remote machine with a larger value of the
1212	type uid_t , off_t , or gid_t than the local system.
1213	[EPERM]
1214	Operation not permitted. An attempt was made to perform an operation limited to
1215	processes with appropriate privileges or to the owner of a file or other resource.
1216	[EPIPE]
1217	Broken pipe. A write was attempted on a socket, pipe, or FIFO for which there is no process
1218	to read the data.
1219	[EPROTO]
1220	Protocol error. Some protocol error occurred. This error is device-specific, but is generally
1221	not related to a hardware failure.
1222	[EPROTONOSUPPORT]
1223	Protocol not supported. The protocol is not supported by the address family, or the protocol
1224	is not supported by the implementation.
1225	[EPROTOTYPE]
1226	Socket type not supported. The socket type is not supported by the protocol.
1227	[ERANGE]
1228	Result too large or too small. The result of the function is too large (overflow) or too small
1229	(underflow) to be represented in the available space (defined in the ISO C standard).

1230	[EROFS]	
1231		Read-only file system. An attempt was made to modify a file or directory on a file system
1232		that is read-only.
1233	[ESPIPE]	
1234		Invalid seek. An attempt was made to access the file offset associated with a pipe or FIFO.
1235	[ESRCH]	
1236		No such process. No process can be found corresponding to that specified by the given
1237		process ID.
1238	[ESTALE]	
1239		Reserved.
1240	XSR [ETIME]	
1241		STREAM <i>ioctl()</i> timeout. The timer set for a STREAMS <i>ioctl()</i> call has expired. The cause of
1242		this error is device-specific and could indicate either a hardware or software failure, or a
1243		timeout value that is too short for the specific operation. The status of the <i>ioctl()</i> operation
1244		is indeterminate.
1245	[ETIMEDOUT]	
1246		Connection timed out. The connection to a remote machine has timed out. If the connection
1247		timed out during execution of the function that reported this error (as opposed to timing
1248		out prior to the function being called), it is unspecified whether the function has completed
1249		some or all of the documented behavior associated with a successful completion of the
1250		function.
1251		or:
1252		Operation timed out. The time limit associated with the operation was exceeded before the
1253		operation completed.
1254	[ETXTBSY]	
1255		Text file busy. An attempt was made to execute a pure-procedure program that is currently
1256		open for writing, or an attempt has been made to open for writing a pure-procedure
1257		program that is being executed.
1258	[EWOULDBLOCK]	
1259		Operation would block. An operation on a socket marked as non-blocking has encountered
1260		a situation such as no data available that otherwise would have caused the function to
1261		suspend execution.
1262		A conforming implementation may assign the same values for [EWOULDBLOCK] and
1263		[EAGAIN].
1264	[EXDEV]	
1265		Improper link. A link to a file on another file system was attempted.
1266	2.3.1 Additional Error Numbers	
1267		Additional implementation-defined error numbers may be defined in <code><errno.h></code> .

1268 2.4 Signal Concepts

1269 2.4.1 Signal Generation and Delivery

1270 A signal is said to be *generated* for (or sent to) a process or thread when the event that causes the
1271 signal first occurs. Examples of such events include detection of hardware faults, timer
1272 RTS expiration, signals generated via the **sigevent** structure and terminal activity, as well as
1273 RTS invocations of *kill()* and *sigqueue()* functions. In some circumstances, the same event generates
1274 signals for multiple processes.

1275 At the time of generation, a determination is made whether the signal has been generated for the
1276 process or for a specific thread within the process. Signals which are generated by some action
1277 attributable to a particular thread, such as a hardware fault, are generated for the thread that
1278 caused the signal to be generated. Signals that are generated in association with a process ID or
1279 process group ID or an asynchronous event such as terminal activity are generated for the
1280 process.

1281 Each process has an action to be taken in response to each signal defined by the system (see
1282 Section 2.4.3 (on page 530)). A signal is said to be *delivered* to a process when the appropriate
1283 action for the process and signal is taken. A signal is said to be *accepted* by a process when the
1284 signal is selected and returned by one of the *sigwait()* functions.

1285 During the time between the generation of a signal and its delivery or acceptance, the signal is
1286 said to be *pending*. Ordinarily, this interval cannot be detected by an application. However, a
1287 signal can be *blocked* from delivery to a thread. If the action associated with a blocked signal is
1288 anything other than to ignore the signal, and if that signal is generated for the thread, the signal
1289 shall remain pending until it is unblocked, it is accepted when it is selected and returned by a
1290 call to the *sigwait()* function, or the action associated with it is set to ignore the signal. Signals
1291 generated for the process shall be delivered to exactly one of those threads within the process
1292 which is in a call to a *sigwait()* function selecting that signal or has not blocked delivery of the
1293 signal. If there are no threads in a call to a *sigwait()* function selecting that signal, and if all
1294 threads within the process block delivery of the signal, the signal shall remain pending on the
1295 process until a thread calls a *sigwait()* function selecting that signal, a thread unblocks delivery
1296 of the signal, or the action associated with the signal is set to ignore the signal. If the action
1297 associated with a blocked signal is to ignore the signal and if that signal is generated for the
1298 process, it is unspecified whether the signal is discarded immediately upon generation or
1299 remains pending.

1300 Each thread has a *signal mask* that defines the set of signals currently blocked from delivery to it.
1301 The signal mask for a thread is initialized from that of its parent or creating thread, or from the
1302 corresponding thread in the parent process if the thread was created as the result of a call to
1303 *fork()*. The *sigaction()*, *sigprocmask()*, and *sigsuspend()* functions control the manipulation of the
1304 signal mask.

1305 The determination of which action is taken in response to a signal is made at the time the signal
1306 is delivered, allowing for any changes since the time of generation. This determination is
1307 independent of the means by which the signal was originally generated. If a subsequent
1308 occurrence of a pending signal is generated, it is implementation-defined as to whether the
1309 RTS signal is delivered or accepted more than once in circumstances other than those in which
1310 queuing is required under the Realtime Signals Extension option. The order in which multiple,
1311 simultaneously pending signals outside the range SIGRTMIN to SIGRTMAX are delivered to or
1312 accepted by a process is unspecified.

1313 When any stop signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is generated for a process, any
1314 pending SIGCONT signals for that process shall be discarded. Conversely, when SIGCONT is
1315 generated for a process, all pending stop signals for that process shall be discarded. When

1316 SIGCONT is generated for a process that is stopped, the process shall be continued, even if the
 1317 SIGCONT signal is blocked or ignored. If SIGCONT is blocked and not ignored, it shall remain
 1318 pending until it is either unblocked or a stop signal is generated for the process.

1319 An implementation shall document any condition not specified by this volume of
 1320 IEEE Std. 1003.1-200x under which the implementation generates signals.

1321 **2.4.2 Realtime Signal Generation and Delivery**

1322 RTS This section describes extensions to support realtime signal generation and delivery. This
 1323 functionality is dependent on support of the Realtime Signals Extension option (and the rest of
 1324 this section is not further shaded for this option).

1325 Some signal-generating functions, such as high-resolution timer expiration, asynchronous I/O
 1326 completion, interprocess message arrival, and the *sigqueue()* function, support the specification
 1327 of an application-defined value, either explicitly as a parameter to the function or in a **sigevent**
 1328 structure parameter. The **sigevent** structure is defined in <**signal.h**> and shall contain at least
 1329 the following members:

1330

Member Type	Member Name	Description
int	<i>sigev_notify</i>	Notification type.
int	<i>sigev_signo</i>	Signal number.
union signal	<i>sigev_value</i>	Signal value.
void*(unsigned signal)	<i>sigev_notify_function</i>	Notification function.
(pthread_attr_t*)	<i>sigev_notify_attributes</i>	Notification attributes.

1337 The *sigev_notify* member specifies the notification mechanism to use when an asynchronous
 1338 event occurs. This volume of IEEE Std. 1003.1-200x defines the following values for the
 1339 *sigev_notify* member:

1340 SIGEV_NONE No asynchronous notification shall be delivered when the event of
 1341 interest occurs.

1342 SIGEV_SIGNAL The signal specified in *sigev_signo* shall be generated for the process when
 1343 the event of interest occurs. If the implementation supports the Realtime
 1344 Signals Extension option and if the SA_SIGINFO flag is set for that signal
 1345 number, then the signal shall be queued to the process and the value
 1346 specified in *sigev_value* shall be the *si_value* component of the generated
 1347 signal. If SA_SIGINFO is not set for that signal number, it is unspecified
 1348 whether the signal is queued and what value, if any, is sent.

1349 SIGEV_THREAD A notification function shall be called to perform notification.

1350 An implementation may define additional notification mechanisms.

1351 The *sigev_signo* member specifies the signal to be generated. The *sigev_value* member is the
 1352 application-defined value to be passed to the signal-catching function at the time of the signal
 1353 delivery or to be returned at signal acceptance as the *si_value* member of the **siginfo_t** structure.

1354 The **signal** union is defined in <**signal.h**> and contains at least the following members:

1355
1356
1357
1358

Member Type	Member Name	Description
int	<i>sival_int</i>	Integer signal value.
void*	<i>sival_ptr</i>	Pointer signal value.

1359
1360

The *sival_int* member is used when the application-defined value is of type **int**; the *sival_ptr* member is used when the application-defined value is a pointer.

1361
1362
1363
1364
1365
1366

When a signal is generated by the *sigqueue()* function or any signal-generating function that supports the specification of an application-defined value, the signal shall be marked pending and, if the SA_SIGINFO flag is set for that signal, the signal shall be queued to the process along with the application-specified signal value. Multiple occurrences of signals so generated are queued in FIFO order. It is unspecified whether signals so generated are queued when the SA_SIGINFO flag is not set for that signal.

1367
1368
1369
1370

Signals generated by the *kill()* function or other events that cause signals to occur, such as detection of hardware faults, *alarm()* timer expiration, or terminal activity, and for which the implementation does not support queuing, have no effect on signals already queued for the same signal number.

1371
1372
1373

When multiple unblocked signals, all in the range SIGRTMIN to SIGRTMAX, are pending, the behavior shall be as if the implementation delivers the pending unblocked signal with the lowest signal number within that range. No other ordering of signal delivery is specified.

1374
1375

If, when a pending signal is delivered, there are additional signals queued to that signal number, the signal remains pending. Otherwise, the pending indication is reset.

1376
1377
1378

Multi-threaded programs can use an alternate event notification mechanism. When a notification is processed, and the *sigev_notify* member of the **sigevent** structure has the value SIGEV_THREAD, the function *sigev_notify_function* is called with parameter *sigev_value*.

1379
1380
1381
1382
1383
1384

The function shall be executed in an environment as if it were the *start_routine* for a newly created thread with thread attributes specified by *sigev_notify_attributes*. If *sigev_notify_attributes* is NULL, the behavior shall be as if the thread were created with the *detachstate* attribute set to PTHREAD_CREATE_DETACHED. Supplying an attributes structure with a *detachstate* attribute of PTHREAD_CREATE_JOINABLE results in undefined behavior. The signal mask of this thread is implementation-defined.

1385 2.4.3 Signal Actions

1386
1387
1388

There are three types of action that can be associated with a signal: SIG_DFL, SIG_IGN, or a pointer to a function. Initially, all signals shall be set to SIG_DFL or SIG_IGN prior to entry of the *main()* routine (see the *exec* functions). The actions prescribed by these values are as follows:

1389

SIG_DFL Signal-specific default action.

1390
1391 RTS
1392
1393

The default actions for the signals defined in this volume of IEEE Std. 1003.1-200x are specified under **<signal.h>**. If the Realtime Signals Extension option is supported, the default actions for the realtime signals in the range SIGRTMIN to SIGRTMAX are to terminate the process abnormally.

1394
1395
1396
1397
1398
1399
1400

If the default action is to stop the process, the execution of that process is temporarily suspended. When a process stops, a SIGCHLD signal shall be generated for its parent process, unless the parent process has set the SA_NOCLDSTOP flag. While a process is stopped, any additional signals that are sent to the process shall not be delivered until the process is continued, except SIGKILL which always terminates the receiving process. A process that is a member of an orphaned process group shall not be allowed to stop in response to

1401 the SIGTSTP, SIGTTIN, or SIGTTOU signals. In cases where delivery of one of
 1402 these signals would stop such a process, the signal shall be discarded.

1403 Setting a signal action to SIG_DFL for a signal that is pending, and whose default
 1404 action is to ignore the signal (for example, SIGCHLD), shall cause the pending
 1405 signal to be discarded, whether or not it is blocked.

1406 The default action for SIGCONT is to resume execution at the point where the
 1407 RTS process was stopped, after first handling any pending unblocked signals. If the
 1408 Realttime Signals Extension option is supported, any queued values pending shall
 1409 be discarded and the resources used to queue them shall be released and returned
 1410 to the system for other use.

1411 SIG_IGN Ignore signal.

1412 Delivery of the signal shall have no effect on the process. The behavior of a process
 1413 RTS is undefined after it ignores a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal that
 1414 RTS was not generated by *kill()*, *sigqueue()*, or *raise()*.

1415 The system shall not allow the action for the signals SIGKILL or SIGSTOP to be set
 1416 to SIG_IGN.

1417 Setting a signal action to SIG_IGN for a signal that is pending shall cause the
 1418 pending signal to be discarded, whether or not it is blocked.

1419 If a process sets the action for the SIGCHLD signal to SIG_IGN, the behavior is
 1420 XSI unspecified, except as specified below.

1421 If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the
 1422 calling processes shall not be transformed into zombie processes when they
 1423 terminate. If the calling process subsequently waits for its children, and the process
 1424 has no unwaited-for children that were transformed into zombie processes, it shall
 1425 block until all of its children terminate, and *wait()*, *waitid()*, and *waitpid()* shall fail
 1426 and set *errno* to [ECHILD].

1427 RTS If the Realttime Signals Extension option is supported, any queued values pending
 1428 shall be discarded and the resources used to queue them shall be released and
 1429 made available to queue other signals.

1430 *pointer to a function*

1431 Catch signal.

1432 On delivery of the signal, the receiving process is to execute the signal-catching
 1433 function at the specified address. After returning from the signal-catching function,
 1434 the receiving process shall resume execution at the point at which it was
 1435 interrupted.

1436 If the SA_SIGINFO flag for the signal is cleared, the signal-catching function shall
 1437 be entered as a C-language function call as follows:

1438 `void func(int signo);`

1439 XSI|RTS If the SA_SIGINFO flag for the signal is set, the signal-catching function shall be
 1440 entered as a C-language function call as follows:

1441 `void func(int signo, siginfo_t *info, void *context);`

1442 where *func* is the specified signal-catching function, *signo* is the signal number of
 1443 the signal being delivered, and *info* is a pointer to a **siginfo_t** structure defined in
 1444 **<signal.h>** containing at least the following members:

1445
1446
1447
1448
1449

Member Type	Member Name	Description
int	<i>si_signo</i>	Signal number
int	<i>si_code</i>	Cause of the signal
union signal	<i>si_value</i>	Signal value

1450
1451
1452

The *si_signo* member contains the signal number. This is the same as the *signo* parameter. The *si_code* member contains a code identifying the cause of the signal. The following values are defined for *si_code*:

1453 **Notes to Reviewers**

1454
1455

This section with side shading will not appear in the final copy. - Ed.

The shading in this area needs some work.

1456 XSI|RTS
1457
1458
1459

SI_USER The signal was sent by the *kill()* function. The implementation may set *si_code* to **SI_USER** if the signal was sent by the *raise()* or *abort()* functions or any similar functions provided as implementation extensions.

1460 RTS

SI_QUEUE The signal was sent by the *sigqueue()* function.

1461 RTS
1462

SI_TIMER The signal was generated by the expiration of a timer set by *timer_settime()*.

1463 RTS
1464

SI_ASYNCIO The signal was generated by the completion of an asynchronous I/O request.

1465 RTS
1466

SI_MESGQ The signal was generated by the arrival of a message on an empty message queue.

1467
1468
1469

If the signal was not generated by one of the functions or events listed above, the *si_code* shall be set to an implementation-defined value that is not equal to any of the values defined above.

1470 RTS
1471
1472

If the Realtime Signals Extension is supported, and *si_code* is one of **SI_QUEUE**, **SI_TIMER**, **SI_ASYNCIO**, or **SI_MESGQ**, then *si_value* contains the application-specified signal value. Otherwise, the contents of *si_value* are undefined.

1473
1474 XSI
1475 RTS

The behavior of a process is undefined after it returns normally from a signal-catching function for a **SIGBUS**, **SIGFPE**, **SIGILL**, or **SIGSEGV** signal that was not generated by *kill()*, *sigqueue()*, or *raise()*.

1476

The system shall not allow a process to catch the signals **SIGKILL** and **SIGSTOP**.

1477
1478
1479

If a process establishes a signal-catching function for the **SIGCHLD** signal while it has a terminated child process for which it has not waited, it is unspecified whether a **SIGCHLD** signal is generated to indicate that child process.

1480
1481
1482
1483

When signal-catching functions are invoked asynchronously with process execution, the behavior of some of the functions defined by this volume of IEEE Std. 1003.1-200x is unspecified if they are called from a signal-catching function.

1484
1485
1486

The following table defines a set of functions that are either reentrant or not interruptible by signals and are async-signal-safe. Therefore applications may invoke them, without restriction, from signal-catching functions:

1487 **Notes to Reviewers**1488 *This section with side shading will not appear in the final copy. - Ed.*

1489 The contents of the following tables need to be reviewed.

1490 Base functions:

1491	<code>_Exit()</code>	<code>fpathconf()</code>	<code>pipe()</code>	<code>stat()</code>
1492	<code>_exit()</code>	<code>fstat()</code>	<code>raise()</code>	<code>symlink()</code>
1493	<code>access()</code>	<code>fsync()</code>	<code>read()</code>	<code>sysconf()</code>
1494	<code>alarm()</code>	<code>ftruncate()</code>	<code>readlink()</code>	<code>tcdrain()</code>
1495	<code>cfgetispeed()</code>	<code>getegid()</code>	<code>rename()</code>	<code>tcflow()</code>
1496	<code>cfgetospeed()</code>	<code>geteuid()</code>	<code>rmdir()</code>	<code>tcflush()</code>
1497	<code>cfsetispeed()</code>	<code>getgid()</code>	<code>setgid()</code>	<code>tcgetattr()</code>
1498	<code>cfsetospeed()</code>	<code>getgroups()</code>	<code>setpgid()</code>	<code>tcgetpgrp()</code>
1499	<code>chdir()</code>	<code>getpgrp()</code>	<code>setsid()</code>	<code>tcsendbreak()</code>
1500	<code>chmod()</code>	<code>getpid()</code>	<code>setuid()</code>	<code>tcsetattr()</code>
1501	<code>chown()</code>	<code>getppid()</code>	<code>sigaction()</code>	<code>tcsetpgrp()</code>
1502	<code>close()</code>	<code>getuid()</code>	<code>sigaddset()</code>	<code>time()</code>
1503	<code>creat()</code>	<code>kill()</code>	<code>sigdelset()</code>	<code>times()</code>
1504	<code>dup()</code>	<code>link()</code>	<code>sigemptyset()</code>	<code>umask()</code>
1505	<code>dup2()</code>	<code>lseek()</code>	<code>sigfillset()</code>	<code>uname()</code>
1506	<code>execle()</code>	<code>lstat()</code>	<code>sigismember()</code>	<code>unlink()</code>
1507	<code>execve()</code>	<code>mkdir()</code>	<code>signal()</code>	<code>utime()</code>
1508	<code>fchmod()</code>	<code>mkfifo()</code>	<code>sigpending()</code>	<code>wait()</code>
1509	<code>fchown()</code>	<code>open()</code>	<code>sigprocmask()</code>	<code>waitpid()</code>
1510	<code>fcntl()</code>	<code>pathconf()</code>	<code>sigsuspend()</code>	<code>write()</code>
1511	<code>fork()</code>	<code>pause()</code>	<code>sleep()</code>	

1512 Realtime functions:

1513	<code>aio_error()</code>	<code>clock_gettime()</code>	<code>sigpause()</code>	<code>timer_getoverrun()</code>
1514	<code>aio_return()</code>	<code>fdatasync()</code>	<code>sigqueue()</code>	<code>timer_gettime()</code>
1515	<code>aio_suspend()</code>	<code>sem_post()</code>	<code>sigset()</code>	<code>timer_settime()</code>

1516 Tracing functions:

1517 `posix_trace_event()` |

1518 All functions not in the above table are considered to be unsafe with respect to |
 1519 signals. In the presence of signals, all functions defined by this volume of |
 1520 IEEE Std. 1003.1-200x shall behave as defined when called from or interrupted by a |
 1521 signal-catching function, with a single exception: when a signal interrupts an |
 1522 unsafe function and the signal-catching function calls an unsafe function, the |
 1523 behavior is undefined.

1524 When a signal is delivered to a thread, if the action of that signal specifies termination, stop, or
 1525 continue, the entire process shall be terminated, stopped, or continued, respectively.

1526 **2.4.4 Signal Effects on Other Functions**

1527 Signals affect the behavior of certain functions defined by this volume of IEEE Std. 1003.1-200x if
1528 delivered to a process while it is executing such a function. If the action of the signal is to
1529 terminate the process, the process shall be terminated and the function shall not return. If the
1530 action of the signal is to stop the process, the process shall stop until continued or terminated.
1531 Generation of a SIGCONT signal for the process shall cause the process to be continued, and the
1532 original function shall continue at the point the process was stopped. If the action of the signal is
1533 to invoke a signal-catching function, the signal-catching function shall be invoked; in this case
1534 the original function is said to be *interrupted* by the signal.

1535 **Notes to Reviewers**

1536 *This section with side shading will not appear in the final copy. - Ed.*

1537 D3, XSH, ERN 20 points out a discrepancy between the following sentence and the paragraph
1538 above beginning "All functions not in the above ...". An interpretation will be filed.

1539 If the signal-catching function executes a **return** statement, the behavior of the interrupted
1540 function shall be as described individually for that function. Signals that are ignored shall not
1541 affect the behavior of any function; signals that are blocked shall not affect the behavior of any
1542 function until they are unblocked and then delivered, except as specified for the *sigpending()* and
1543 *sigwait()* functions.

1544 2.5 Standard I/O Streams

1545 A stream is associated with an external file (which may be a physical device) by *opening* a file,
1546 which may involve *creating* a new file. Creating an existing file causes its former contents to be
1547 discarded if necessary. If a file can support positioning requests, (such as a disk file, as opposed
1548 to a terminal), then a *file position indicator* associated with the stream is positioned at the start
1549 (byte number 0) of the file, unless the file is opened with append mode, in which case it is
1550 implementation-defined whether the file position indicator is initially positioned at the
1551 beginning or end of the file. The file position indicator is maintained by subsequent reads,
1552 writes, and positioning requests, to facilitate an orderly progression through the file. All input
1553 takes place as if bytes were read by successive calls to *fgetc()*; all output takes place as if bytes
1554 were written by successive calls to *fputc()*.

1555 When a stream is *unbuffered*, bytes are intended to appear from the source or at the destination
1556 as soon as possible; otherwise, bytes may be accumulated and transmitted as a block. When a
1557 stream is *fully buffered*, bytes are intended to be transmitted as a block when a buffer is filled.
1558 When a stream is *line buffered*, bytes are intended to be transmitted as a block when a newline
1559 byte is encountered. Furthermore, bytes are intended to be transmitted as a block when a buffer
1560 is filled, when input is requested on an unbuffered stream, or when input is requested on a line-
1561 buffered stream that requires the transmission of bytes. Support for these characteristics is
1562 implementation-defined, and may be affected via *setbuf()* and *setvbuf()*.

1563 A file may be disassociated from a controlling stream by *closing* the file. Output streams are
1564 flushed (any unwritten buffer contents are transmitted) before the stream is disassociated from
1565 the file. The value of a pointer to a FILE object is indeterminate after the associated file is closed
1566 (including the standard streams).

1567 A file may be subsequently reopened, by the same or another program execution, and its
1568 contents reclaimed or modified (if it can be repositioned at its start). If the *main()* function
1569 returns to its original caller, or if the *exit()* function is called, all open files are closed (hence all
1570 output streams are flushed) before program termination. Other paths to program termination,
1571 such as calling *abort()*, need not close all files properly.

1572 The address of the FILE object used to control a stream may be significant; a copy of a FILE
1573 object need not necessarily serve in place of the original.

1574 At program start-up, three streams are predefined and need not be opened explicitly: *standard*
1575 *input* (for reading conventional input), *standard output* (for writing conventional output), and
1576 *standard error* (for writing diagnostic output). When opened, the standard error stream is not
1577 fully buffered; the standard input and standard output streams are fully buffered if and only if
1578 the stream can be determined not to refer to an interactive device.

1579 2.5.1 Interaction of File Descriptors and Standard I/O Streams

1580 cx This section describes the interaction of file descriptors and standard I/O streams. This
1581 functionality is an extension to the ISO C standard (and the rest of this section is not further CX
1582 shaded).

1583 An open file description may be accessed through a file descriptor, which is created using
1584 functions such as *open()* or *pipe()*, or through a stream, which is created using functions such as
1585 *fopen()* or *popen()*. Either a file descriptor or a stream is called a *handle* on the open file
1586 description to which it refers; an open file description may have several handles.

1587 Handles can be created or destroyed by explicit user action, without affecting the underlying
1588 open file description. Some of the ways to create them include *fcntl()*, *dup()*, *fdopen()*, *fileno()*,
1589 and *fork()*. They can be destroyed by at least *fclose()*, *close()*, and the *exec* functions.

1590 A file descriptor that is never used in an operation that could affect the file offset (for example,
 1591 *read()*, *write()*, or *lseek()*) is not considered a handle for this discussion, but could give rise to one
 1592 (for example, as a consequence of *fdopen()*, *dup()*, or *fork()*). This exception does not include the
 1593 file descriptor underlying a stream, whether created with *fopen()* or *fdopen()*, so long as it is not
 1594 used directly by the application to affect the file offset. The *read()* and *write()* functions
 1595 implicitly affect the file offset; *lseek()* explicitly affects it.

1596 The result of function calls involving any one handle (the *active handle*) is defined elsewhere in
 1597 this volume of IEEE Std. 1003.1-200x, but if two or more handles are used, and any one of them is
 1598 a stream, the application shall ensure that their actions are coordinated as described below. If
 1599 this is not done, the result is undefined.

1600 A handle which is a stream is considered to be closed when either an *fclose()* or *freopen()* is
 1601 executed on it (the result of *freopen()* is a new stream, which cannot be a handle on the same
 1602 open file description as its previous value), or when the process owning that stream terminates
 1603 with *exit()* or *abort()*. A file descriptor is closed by *close()*, *_exit()*, or the *exec* functions when
 1604 `FD_CLOEXEC` is set on that file descriptor.

1605 For a handle to become the active handle, the application shall ensure that the actions below are
 1606 performed between the last use of the handle (the current active handle) and the first use of the
 1607 second handle (the future active handle). The second handle then becomes the active handle. All
 1608 activity by the application affecting the file offset on the first handle shall be suspended until it
 1609 again becomes the active file handle. (If a stream function has as an underlying function one that
 1610 affects the file offset, the stream function shall be considered to affect the file offset.)

1611 The handles need not be in the same process for these rules to apply.

1612 Note that after a *fork()*, two handles exist where one existed before. The application shall ensure
 1613 that, if both handles can ever be accessed, they are both in a state where the other could become
 1614 the active handle first. The application shall prepare for a *fork()* exactly as if it were a change of
 1615 active handle. (If the only action performed by one of the processes is one of the *exec* functions or
 1616 *_exit()* (not *exit()*), the handle is never accessed in that process.)

1617 For the first handle, the first applicable condition below applies. After the actions required
 1618 below are taken, if the handle is still open, the application can close it.

- 1619 • If it is a file descriptor, no action is required.
- 1620 • If the only further action to be performed on any handle to this open file descriptor is to close
 1621 it, no action need be taken.
- 1622 • If it is a stream which is unbuffered, no action need be taken.
- 1623 • If it is a stream which is line buffered, and the last byte written to the stream was a newline
 1624 (that is, as if a:
 1625

```
putc( '\n' )
```


 1626 was the most recent operation on that stream), no action need be taken.
- 1627 • If it is a stream which is open for writing or appending (but not also open for reading), the
 1628 application shall either perform an *flush()*, or the stream shall be closed.
- 1629 • If the stream is open for reading and it is at the end of the file (*feof()* is true), no action need
 1630 be taken.
- 1631 • If the stream is open with a mode that allows reading and the underlying open file
 1632 description refers to a device that is capable of seeking, the application shall either perform
 1633 an *flush()*, or the stream shall be closed.

1634 Otherwise, the result is undefined.

1635 For the second handle:

- 1636 • If any previous active handle has been used by a function that explicitly changed the file
1637 offset, except as required above for the first handle, the application shall perform an *lseek()* or
1638 *fseek()* (as appropriate to the type of handle) to an appropriate location.

1639 If the active handle ceases to be accessible before the requirements on the first handle, above,
1640 have been met, the state of the open file description becomes undefined. This might occur during
1641 functions such as a *fork()* or *_exit()*.

1642 The *exec* functions make inaccessible all streams that are open at the time they are called,
1643 independent of which streams or file descriptors may be available to the new process image.

1644 When these rules are followed, regardless of the sequence of handles used, implementations
1645 shall ensure that an application, even one consisting of several processes, shall yield correct
1646 results: no data shall be lost or duplicated when writing, and all data shall be written in order,
1647 except as requested by seeks. It is implementation-defined whether, and under what conditions,
1648 all input is seen exactly once.

1649 If the rules above are not followed, the result is unspecified.

1650 Each function that operates on a stream is said to have zero or more *underlying functions*. This
1651 means that the stream function shares certain traits with the underlying functions, but does not
1652 require that there be any relation between the implementations of the stream function and its
1653 underlying functions.

1654 2.5.2 Stream Orientation and Encoding Rules

1655 For conformance to the ISO/IEC 9899:1999 standard, the definition of a stream includes an
1656 *orientation*. After a stream is associated with an external file, but before any operations are
1657 performed on it, the stream is without orientation. Once a wide-character input/output function
1658 has been applied to a stream without orientation, the stream shall become *wide-oriented*.
1659 Similarly, once a byte input/output function has been applied to a stream without orientation,
1660 the stream shall become *byte-oriented*. Only a call to the *freopen()* function or the *fwide()* function
1661 can otherwise alter the orientation of a stream.

1662 A successful call to *freopen()* shall remove any orientation. The three predefined streams *standard*
1663 *input*, *standard output*, and *standard error* shall be unoriented at program start-up.

1664 Byte input/output functions cannot be applied to a wide-oriented stream, and wide-character
1665 input/output functions cannot be applied to a byte-oriented stream. The remaining stream
1666 operations shall not affect and shall not be affected by a stream's orientation, except for the
1667 following additional restrictions:

- 1668 • Binary wide-oriented streams have the file positioning restrictions ascribed to both text and
1669 binary streams.
- 1670 • For wide-oriented streams, after a successful call to a file-positioning function that leaves the
1671 file position indicator prior to the end-of-file, a wide-character output function can overwrite
1672 a partial character; any file contents beyond the byte(s) written are henceforth undefined.

1673 Each wide-oriented stream has an associated **mbstate_t** object that stores the current parse state
1674 of the stream. A successful call to *fgetpos()* shall store a representation of the value of this
1675 **mbstate_t** object as part of the value of the **fpos_t** object. A later successful call to *fsetpos()* using
1676 the same stored **fpos_t** value shall restore the value of the associated **mbstate_t** object as well as
1677 the position within the controlled stream.

1678 Implementations that support multiple encoding rules associate an encoding rule with the
1679 stream. The encoding rule shall be determined by the setting of the *LC_CTYPE* category in the
1680 current locale at the time when the stream becomes wide-oriented. If a wide-character
1681 input/output function is applied to a byte-oriented stream, the encoding rule used is undefined.
1682 As with the stream's orientation, the encoding rule associated with a stream cannot be changed
1683 once it has been set, except by a successful call to *freopen()* which clears the encoding rule and
1684 resets the orientation to unoriented.

1685 Although both text and binary wide-oriented streams are conceptually sequences of wide
1686 characters, the external file associated with a wide-oriented stream is a sequence of (possibly
1687 multibyte) characters generalized as follows:

- 1688 • Multibyte encodings within files may contain embedded null bytes (unlike multibyte
1689 encodings valid for use internal to the program).
- 1690 • A file need not begin nor end in the initial shift state.

1691 Moreover, the encodings used for characters may differ among files. Both the nature and choice
1692 of such encodings are implementation-defined.

1693 The wide-character input functions read characters from the stream and convert them to wide
1694 characters as if they were read by successive calls to the *fgetwc()* function. Each conversion shall
1695 occur as if by a call to the *mbrtowc()* function, with the conversion state described by the stream's
1696 own **mbstate_t** object, except the encoding rule associated with the stream is used instead of the
1697 encoding rule implied by the *LC_CTYPE* category of the current locale.

1698 The wide-character output functions convert wide characters to (possibly multibyte) characters
1699 and write them to the stream as if they were written by successive calls to the *fputwc()* function.
1700 Each conversion shall occur as if by a call to the *wcrtomb()* function, with the conversion state
1701 described by the stream's own **mbstate_t** object, except the encoding rule associated with the
1702 stream is used instead of the encoding rule implied by the *LC_CTYPE* category of the current
1703 locale.

1704 An *encoding error* shall occur if the character sequence presented to the underlying *mbrtowc()*
1705 function does not form a valid (generalized) character, or if the code value passed to the
1706 underlying *wcrtomb()* function does not correspond to a valid (generalized) character. The
1707 wide-character input/output functions and the byte input/output functions store the value of
1708 the macro *EILSEQ* in *errno* if and only if an encoding error occurs.

1709 **2.6 STREAMS**

1710 XSR STREAMS functionality is provided on implementations supporting the XSI STREAMS Option
 1711 Group. This functionality is dependent on support of the XSI STREAMS option (and the rest of
 1712 this section is not further shaded for this option).

1713 STREAMS provides a uniform mechanism for implementing networking services and other
 1714 character-based I/O. The STREAMS function provides direct access to protocol modules. A
 1715 STREAM is typically a full-duplex connection between a process and an open device or pseudo-
 1716 device. However, since pipes may be STREAMS-based, a STREAM can be a full-duplex
 1717 connection between two processes. The STREAM itself exists entirely within the implementation
 1718 and provides a general character I/O function for processes. It optionally includes one or more
 1719 intermediate processing modules that are interposed between the process end of the STREAM
 1720 (STREAM head) and a device driver at the end of the STREAM (STREAM end).

1721 STREAMS I/O is based on messages. Messages flow in both directions in a STREAM. A given
 1722 module need not understand and process every message in the STREAM, but every module in
 1723 the STREAM handles every message. Each module accepts messages from one of its neighbor
 1724 modules in the STREAM, and passes them to the other neighbor. For example, a line discipline
 1725 module may transform the data. Data flow through the intermediate modules is bidirectional,
 1726 with all modules handling, and optionally processing, all messages. There are three types of
 1727 messages:

- 1728 • *Data messages* containing actual data for input or output
- 1729 • *Control data* containing instructions for the STREAMS modules and underlying
 1730 implementation
- 1731 • Other messages, which include file descriptors

1732 The function between the STREAM and the rest of the implementation is provided by a set of
 1733 functions at the STREAM head. When a process calls *write()*, *putmsg()*, *putpmsg()*, or *ioctl()*,
 1734 messages are sent down the STREAM, and *read()*, *getmsg()*, or *getpmsg()* accepts data from the
 1735 STREAM and passes it to a process. Data intended for the device at the downstream end of the
 1736 STREAM is packaged into messages and sent downstream, while data and signals from the
 1737 device are composed into messages by the device driver and sent upstream to the STREAM
 1738 head.

1739 When a STREAMS-based device is opened, a STREAM is created that contains two modules: the
 1740 STREAM head module and the STREAM end (driver) module. If pipes are STREAMS-based in
 1741 an implementation, when a pipe is created, two STREAMS are created, each containing a
 1742 STREAM head module. Other modules are added to the STREAM using *ioctl()*. New modules
 1743 are “pushed” onto the STREAM one at a time in last-in, first-out (LIFO) style, as though the
 1744 STREAM was a push-down stack.

1745 **Priority**

1746 Message types are classified according to their queuing priority and may be *normal* (non-
 1747 priority), *priority*, or *high-priority* messages. A message belongs to a particular priority band that
 1748 determines its ordering when placed on a queue. Normal messages have a priority band of 0 and
 1749 are always placed at the end of the queue following all other messages in the queue. High-
 1750 priority messages are always placed at the head of a queue, but are discarded if there is already a
 1751 high-priority message in the queue. Their priority band is ignored; they are high-priority by
 1752 virtue of their type. Priority messages have a priority band greater than 0. Priority messages are
 1753 always placed after any messages of the same or higher priority. High-priority and priority
 1754 messages are used to send control and data information outside the normal flow of control. By
 1755 convention, high-priority messages are not affected by flow control. Normal and priority

1756 messages have separate flow controls.

1757 **Message Parts**

1758 A process may access STREAMS messages that contain a data part, control part, or both. The
1759 data part is that information which is transmitted over the communication medium and the
1760 control information is used by the local STREAMS modules. The other types of messages are
1761 used between modules and are not accessible to processes. Messages containing only a data part
1762 are accessible via *putmsg()*, *putpmsg()*, *getmsg()*, *getpmsg()*, *read()*, or *write()*. Messages
1763 containing a control part with or without a data part are accessible via calls to *putmsg()*,
1764 *putpmsg()*, *getmsg()*, or *getpmsg()*.

1765 **2.6.1 Accessing STREAMS**

1766 A process accesses STREAMS-based files using the standard functions *close()*, *ioctl()*, *getmsg()*,
1767 *getpmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *putpmsg()*, *read()*, or *write()*. Refer to the applicable
1768 function definitions for general properties and errors.

1769 Calls to *ioctl()* are used to perform control functions with the STREAMS-based device associated
1770 with the file descriptor *fildes*. The arguments *command* and *arg* are passed to the STREAMS file
1771 designated by *fildes* and are interpreted by the STREAM head. Certain combinations of these
1772 arguments may be passed to a module or driver in the STREAM.

1773 Since these STREAMS requests are a subset of *ioctl()*, they are subject to the errors described
1774 there.

1775 STREAMS modules and drivers can detect errors, sending an error message to the STREAM
1776 head, thus causing subsequent functions to fail and set *errno* to the value specified in the
1777 message. In addition, STREAMS modules and drivers can elect to fail a particular *ioctl()* request
1778 alone by sending a negative acknowledgement message to the STREAM head. This causes just
1779 the pending *ioctl()* request to fail and set *errno* to the value specified in the message.

1780 2.7 XSI Interprocess Communication

1781 XSI This section describes extensions to support interprocess communication. This functionality is
 1782 dependent on support of the XSI Extension (and the rest of this section is not further shaded for
 1783 this option).

1784 The following message passing, semaphore, and shared memory services form an XSI
 1785 interprocess communication facility. Certain aspects of their operation are common, and are
 1786 described below.

1787
 1788

IPC Functions		
<i>msgctl()</i>	<i>semctl()</i>	<i>shmctl()</i>
<i>msgget()</i>	<i>semget()</i>	<i>shmdt()</i>
<i>msgrcv()</i>	<i>semop()</i>	<i>shmget()</i>
<i>msgsnd()</i>	<i>shmat()</i>	

1789
 1790
 1791
 1792

1793 Another interprocess communication facility is provided by functions in the Realtime Option
 1794 Group; see Section 2.8 (on page 543).

1795 2.7.1 IPC General Description

1796 Each individual shared memory segment, message queue, and semaphore set is identified by a
 1797 unique positive integer, called respectively a shared memory identifier, *shmid*, a semaphore
 1798 identifier, *semid*, and a message queue identifier, *msqid*. The identifiers are returned by calls to
 1799 *shmget()*, *semget()*, and *msgget()*, respectively.

1800 Associated with each identifier is a data structure which contains data related to the operations
 1801 which may be or may have been performed; see the Base Definitions volume of
 1802 IEEE Std. 1003.1-200x, <sys/shm.h>, <sys/sem.h>, and <sys/msg.h> for their descriptions.

1803 Each of the data structures contains both ownership information and an **ipc_perm** structure (see
 1804 the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/ipc.h>) which are used in conjunction
 1805 to determine whether or not read/write (read/alter for semaphores) permissions should be
 1806 granted to processes using the IPC facilities. The *mode* member of the **ipc_perm** structure acts as
 1807 a bit field which determines the permissions.

1808 The values of the bits are given below in octal notation.

1809
 1810

Bit	Meaning
0400	Read by user.
0200	Write by user.
0040	Read by group.
0020	Write by group.
0004	Read by others.
0002	Write by others.

1811
 1812
 1813
 1814
 1815
 1816

1817 The name of the **ipc_perm** structure is *shm_perm*, *sem_perm*, or *msg_perm*, depending on which
 1818 service is being used. In each case, read and write/alter permissions are granted to a process if
 1819 one or more of the following are true ("xxx" is replaced by *shm*, *sem*, or *msg*, as appropriate):

- 1820 • The process has appropriate privileges.
- 1821 • The effective user ID of the process matches *xxx_perm.cuid* or *xxx_perm.uid* in the data
 1822 structure associated with the IPC identifier, and the appropriate bit of the *user* field in
 1823 *xxx_perm.mode* is set.

- 1824
- 1825
- 1826
- 1827
- The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* but the effective group ID of the process matches *xxx_perm.cgid* or *xxx_perm.gid* in the data structure associated with the IPC identifier, and the appropriate bit of the *group* field in *xxx_perm.mode* is set.
- 1828
- 1829
- 1830
- 1831
- The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* and the effective group ID of the process does not match *xxx_perm.cgid* or *xxx_perm.gid* in the data structure associated with the IPC identifier, but the appropriate bit of the *other* field in *xxx_perm.mode* is set.
- 1832
- Otherwise, the permission is denied.

1833 **2.8 Realtime**

1834 This section defines functions to support the source portability of applications with realtime
 1835 requirements. The presence of many of these functions is dependent on support for
 1836 implementation options described in the text.

1837 The specific functional areas included in this section and their scope include the following. Full
 1838 definitions of these terms can be found in the Base Definitions volume of IEEE Std. 1003.1-200x,
 1839 Chapter 3, Definitions.

- 1840 • Semaphores
- 1841 • Process Memory Locking
- 1842 • Memory Mapped Files and Shared Memory Objects
- 1843 • Priority Scheduling
- 1844 • Realtime Signal Extension
- 1845 • Timers
- 1846 • Interprocess Communication
- 1847 • Synchronized Input and Output
- 1848 • Asynchronous Input and Output

1849 All the realtime functions defined in this volume of IEEE Std. 1003.1-200x are portable, although
 1850 some of the numeric parameters used by an implementation may have hardware dependencies.

1851 **2.8.1 Realtime Signals**

1852 RTS Realtime signal generation and delivery is dependent on support for the Realtime Signals
 1853 Extension option.

1854 See Section 2.4.2 (on page 529).

1855 **2.8.2 Asynchronous I/O**

1856 AIO The functionality described in this section is dependent on support of the Asynchronous Input
 1857 and Output option (and the rest of this section is not further shaded for this option).

1858 An asynchronous I/O control block structure **aiocb** is used in many asynchronous I/O
 1859 functions. It is defined in the Base Definitions volume of IEEE Std. 1003.1-200x, <**aio.h**> and has
 1860 at least the following members:

Member Type	Member Name	Description
int	<i>aio_fildes</i>	File descriptor.
off_t	<i>aio_offset</i>	File offset.
volatile void*	<i>aio_buf</i>	Location of buffer.
size_t	<i>aio_nbytes</i>	Length of transfer.
int	<i>aio_reqprio</i>	Request priority offset.
struct sigevent	<i>aio_sigevent</i>	Signal number and value.
int	<i>aio_lio_opcode</i>	Operation to be performed.

1861 The *aio_fildes* element is the file descriptor on which the asynchronous operation is performed.

1870 If O_APPEND is not set for the file descriptor *aio_fildes* and if *aio_fildes* is associated with a
 1871 device that is capable of seeking, then the requested operation takes place at the absolute
 1872 position in the file as given by *aio_offset*, as if *lseek()* were called immediately prior to the

1873 operation with an *offset* argument equal to *aio_offset* and a *whence* argument equal to `SEEK_SET`.
 1874 If `O_APPEND` is set for the file descriptor, or if *aio_fildes* is associated with a device that is
 1875 incapable of seeking, write operations append to the file in the same order as the calls were
 1876 made, with the following exception: under implementation-defined circumstances, such as
 1877 operation on a multiprocessor or when requests of differing priorities are submitted at the same
 1878 time, the ordering restriction may be relaxed. After a successful call to enqueue an asynchronous
 1879 I/O operation, the value of the file offset for the file is unspecified. The *aio_nbytes* and *aio_buf*
 1880 elements are the same as the *nbyte* and *buf* arguments defined by `read()` and `write()`, respectively.

1881 If `_POSIX_PRIORITIZED_IO` and `_POSIX_PRIORITY_SCHEDULING` are defined, then
 1882 asynchronous I/O is queued in priority order, with the priority of each asynchronous operation
 1883 based on the current scheduling priority of the calling process. The *aio_reqprio* member can be
 1884 used to lower (but not raise) the asynchronous I/O operation priority and is within the range
 1885 zero through `{AIO_PRIO_DELTA_MAX}`, inclusive. Unless both `_POSIX_PRIORITIZED_IO` and
 1886 `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing asynchronous I/O
 1887 requests is unspecified. When both `_POSIX_PRIORITIZED_IO` and
 1888 `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing of requests submitted
 1889 by processes whose schedulers are not `SCHED_FIFO`, `SCHED_RR`, or `SCHED_SPORADIC` is
 1890 unspecified. The priority of an asynchronous request is computed as (process scheduling
 1891 priority) minus *aio_reqprio*. The priority assigned to each asynchronous I/O request is an
 1892 indication of the desired order of execution of the request relative to other asynchronous I/O
 1893 requests for this file. If `_POSIX_PRIORITIZED_IO` is defined, requests issued with the same
 1894 priority to a character special file are processed by the underlying device in FIFO order; the order
 1895 of processing of requests of the same priority issued to files that are not character special files is
 1896 unspecified. Numerically higher priority values indicate requests of higher priority. The value of
 1897 *aio_reqprio* has no effect on process scheduling priority. When prioritized asynchronous I/O
 1898 requests to the same file are blocked waiting for a resource required for that I/O operation, the
 1899 higher-priority I/O requests shall be granted the resource before lower-priority I/O requests are
 1900 granted the resource. The relative priority of asynchronous I/O and synchronous I/O is
 1901 implementation-defined. If `_POSIX_PRIORITIZED_IO` is defined, the implementation shall
 1902 define for which files I/O prioritization is supported.

1903 The *aio_sigevent* determines how the calling process shall be notified upon I/O completion, as
 1904 specified in Section 2.4.1 (on page 528). If *aio_sigevent.sigev_notify* is `SIGEV_NONE`, then no
 1905 signal shall be posted upon I/O completion, but the error status for the operation and the return
 1906 status for the operation shall be set appropriately.

1907 The *aio_lio_opcode* field is used only by the `lio_listio()` call. The `lio_listio()` call allows multiple
 1908 asynchronous I/O operations to be submitted at a single time. The function takes as an
 1909 argument an array of pointers to `aiocb` structures. Each `aiocb` structure indicates the operation to
 1910 be performed (read or write) via the *aio_lio_opcode* field.

1911 The address of the `aiocb` structure is used as a handle for retrieving the error status and return
 1912 status of the asynchronous operation while it is in progress.

1913 The `aiocb` structure and the data buffers associated with the asynchronous I/O operation are
 1914 being used by the system for asynchronous I/O while, and only while, the error status of the
 1915 asynchronous operation is equal to `EINPROGRESS`. Applications shall not modify the `aiocb`
 1916 structure while the structure is being used by the system for asynchronous I/O.

1917 The return status of the asynchronous operation is the number of bytes transferred by the I/O
 1918 operation. If the error status is set to indicate an error completion, then the return status is set to
 1919 the return value that the corresponding `read()`, `write()`, or `fsync()` call would have returned.
 1920 When the error status is not equal to `EINPROGRESS`, the return status shall reflect the return
 1921 status of the corresponding synchronous operation.

1922 2.8.3 Memory Management

1923 2.8.3.1 Memory Locking

1924 ML The functionality described in this section is dependent on support of the Process Memory
1925 Locking option (and the rest of this section is not further shaded for this option).

1926 Range memory locking operations are defined in terms of pages. Implementations may restrict
1927 the size and alignment of range lockings to be on page-size boundaries. The page size, in bytes,
1928 is the value of the configurable system variable {PAGESIZE}. If an implementation has no
1929 restrictions on size or alignment, it may specify a 1-byte page size.

1930 Memory locking guarantees the residence of portions of the address space. It is
1931 implementation-defined whether locking memory guarantees fixed translation between virtual
1932 addresses (as seen by the process) and physical addresses. Per-process memory locks are not
1933 inherited across a *fork()*, and all memory locks owned by a process are unlocked upon *exec* or
1934 process termination. Unmapping of an address range removes any memory locks established on
1935 that address range by this process.

1936 2.8.3.2 Memory Mapped Files

1937 MF The functionality described in this section is dependent on support of the Memory Mapped Files
1938 option (and the rest of this section is not further shaded for this option).

1939 Range memory mapping operations are defined in terms of pages. Implementations may
1940 restrict the size and alignment of range mappings to be on page-size boundaries. The page size,
1941 in bytes, is the value of the configurable system variable {PAGESIZE}. If an implementation has
1942 no restrictions on size or alignment, it may specify a 1-byte page size.

1943 Memory mapped files provide a mechanism that allows a process to access files by directly
1944 incorporating file data into its address space. Once a file is mapped into a process address space,
1945 the data can be manipulated as memory. If more than one process maps a file, its contents are
1946 shared among them. If the mappings allow shared write access, then data written into the
1947 memory object through the address space of one process appears in the address spaces of all
1948 processes that similarly map the same portion of the memory object.

1949 SHM Shared memory objects are named regions of storage that may be independent of the file system
1950 and can be mapped into the address space of one or more processes to allow them to share the
1951 associated memory.

1952 SHM An *unlink()* of a file or *shm_unlink()* of a shared memory object, while causing the removal of the
1953 name, does not unmap any mappings established for the object. Once the name has been
1954 removed, the contents of the memory object are preserved as long as it is referenced. The
1955 memory object remains referenced as long as a process has the memory object open or has some
1956 area of the memory object mapped.

1957 2.8.3.3 Memory Protection

1958 MPR MF The functionality described in this section is dependent on support of the Memory Protection
1959 and Memory Mapped Files option (and the rest of this section is not further shaded for these
1960 options).

1961 When an object is mapped, various application accesses to the mapped region may result in
1962 signals. In this context, SIGBUS is used to indicate an error using the mapped object, and
1963 SIGSEGV is used to indicate a protection violation or misuse of an address:

- 1964 • A mapping may be restricted to disallow some types of access.

- 1965 • Write attempts to memory that was mapped without write access, or any access to memory
1966 mapped PROT_NONE, shall result in a SIGSEGV signal.
- 1967 • References to unmapped addresses shall result in a SIGSEGV signal.
- 1968 • Reference to whole pages within the mapping, but beyond the current length of the object,
1969 shall result in a SIGBUS signal.
- 1970 • The size of the object is unaffected by access beyond the end of the object (even if a SIGBUS is
1971 not generated).

1972 2.8.3.4 *Typed Memory Objects*

1973 TYM The functionality described in this section is dependent on support of the Typed Memory
1974 Objects option (and the rest of this section is not further shaded for this option).

1975 Implementations may support the Typed Memory Objects option without supporting the
1976 Memory Mapped Files option or the Shared Memory Objects option. Typed memory objects are
1977 implementation-configurable named storage pools accessible from one or more processors in a
1978 system, each via one or more ports, such as backplane buses, LANs, I/O channels, and so on.
1979 Each valid combination of a storage pool and a port is identified through a name that is defined
1980 at system configuration time, in an implementation-defined manner; the name may be
1981 independent of the file system. Using this name, a typed memory object can be opened and
1982 mapped into process address space. For a given storage pool and port, it is necessary to support
1983 both dynamic allocation from the pool as well as mapping at an application-supplied offset
1984 within the pool; when dynamic allocation has been performed, subsequent deallocation must be
1985 supported. Lastly, accessing typed memory objects from different ports requires a method for
1986 obtaining the offset and length of contiguous storage of a region of typed memory (dynamically
1987 allocated or not); this allows typed memory to be shared among processes and/or processors
1988 while being accessed from the desired port.

1989 2.8.4 **Process Scheduling**

1990 **Scheduling Policies**

1991 PS The functionality described in this section is dependent on support of the Process Scheduling
1992 option (and the rest of this section is not further shaded for this option).

1993 The scheduling semantics described in this volume of IEEE Std. 1003.1-200x are defined in terms
1994 of a conceptual model that contains a set of thread lists. No implementation structures are
1995 necessarily implied by the use of this conceptual model. It is assumed that no time elapses
1996 during operations described using this model, and therefore no simultaneous operations are
1997 possible. This model discusses only processor scheduling for runnable threads, but it should be
1998 noted that greatly enhanced predictability of realtime applications result if the sequencing of
1999 other resources takes processor scheduling policy into account.

2000 There is, conceptually, one thread list for each priority. Any runnable thread may be on any
2001 thread list. Multiple scheduling policies shall be provided. Each non-empty thread list is
2002 ordered, contains a head as one end of its order, and a tail as the other. The purpose of a
2003 scheduling policy is to define the allowable operations on this set of lists (for example, moving
2004 threads between and within lists).

2005 Each process shall be controlled by an associated scheduling policy and priority. These
2006 parameters may be specified by explicit application execution of the *sched_setscheduler()* or
2007 *sched_setparam()* functions.

2008 Each thread shall be controlled by an associated scheduling policy and priority. These
 2009 parameters may be specified by explicit application execution of the *pthread_setschedparam()*
 2010 function.

2011 Associated with each policy is a priority range. Each policy definition shall specify the minimum
 2012 priority range for that policy. The priority ranges for each policy may but need not overlap the
 2013 priority ranges of other policies.

2014 A conforming implementation shall select the thread that is defined as being at the head of the
 2015 highest priority non-empty thread list to become a running thread, regardless of its associated
 2016 policy. This thread is then removed from its thread list.

2017 Four scheduling policies are specifically required. Other implementation-defined scheduling
 2018 policies may be defined. The following symbols are defined in the Base Definitions volume of
 2019 IEEE Std. 1003.1-200x, <**sched.h**>:

2020 **SCHED_FIFO** First in, first out (FIFO) scheduling policy.

2021 **SCHED_RR** Round robin scheduling policy.

2022 ss **SCHED_SPORADIC** Sporadic server scheduling policy.

2023 **SCHED_OTHER** Another scheduling policy.

2024 The values of these symbols shall be distinct.

2025 **SCHED_FIFO**

2026 Conforming implementations shall include a scheduling policy called the FIFO scheduling
 2027 policy.

2028 Threads scheduled under this policy are chosen from a thread list that is ordered by the time its
 2029 threads have been on the list without being executed; generally, the head of the list is the thread
 2030 that has been on the list the longest time, and the tail is the thread that has been on the list the
 2031 shortest time.

2032 Under the **SCHED_FIFO** policy, the modification of the definitional thread lists is as follows:

- 2033 1. When a running thread becomes a preempted thread, it becomes the head of the thread list
 2034 for its priority.
- 2035 2. When a blocked thread becomes a runnable thread, it becomes the tail of the thread list for
 2036 its priority.
- 2037 3. When a running thread calls the *sched_setscheduler()* function, the process specified in the
 2038 function call is modified to the specified policy and the priority specified by the *param*
 2039 argument.
- 2040 4. When a running thread calls the *sched_setparam()* function, the priority of the process
 2041 specified in the function call is modified to the priority specified by the *param* argument.
- 2042 5. When a running thread calls the *pthread_setschedparam()* function, the thread specified in
 2043 the function call is modified to the specified policy and the priority specified by the *param*
 2044 argument.
- 2045 6. If a thread whose policy or priority has been modified is a running thread or is runnable, it
 2046 then becomes the tail of the thread list for its new priority.
- 2047 7. When a running thread issues the *sched_yield()* function, the thread becomes the tail of the
 2048 thread list for its priority.

2049 8. At no other time is the position of a thread with this scheduling policy within the thread
2050 lists affected.

2051 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max()*
2052 and *sched_get_priority_min()* functions when SCHED_FIFO is provided as the parameter.
2053 Conforming implementations shall provide a priority range of at least 32 priorities for this
2054 policy.

2055 SCHED_RR

2056 Conforming implementations shall include a scheduling policy called the *round robin* scheduling
2057 policy. This policy is identical to the SCHED_FIFO policy with the additional condition that
2058 when the implementation detects that a running thread has been executing as a running thread
2059 for a time period of the length returned by the *sched_rr_get_interval()* function or longer, the
2060 thread shall become the tail of its thread list and the head of that thread list shall be removed
2061 and made a running thread.

2062 The effect of this policy is to ensure that if there are multiple SCHED_RR threads at the same
2063 priority, one of them does not monopolize the processor. An application should not rely only on
2064 the use of SCHED_RR to ensure application progress among multiple threads if the application
2065 includes threads using the SCHED_FIFO policy at the same or higher priority levels or
2066 SCHED_RR threads at a higher priority level.

2067 A thread under this policy that is preempted and subsequently resumes execution as a running
2068 thread completes the unexpired portion of its round robin interval time period.

2069 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max()*
2070 and *sched_get_priority_min()* functions when SCHED_RR is provided as the parameter.
2071 Conforming implementations shall provide a priority range of at least 32 priorities for this
2072 policy.

2073 SCHED_SPORADIC

2074 SS|TSP The functionality described in this section is dependent on support of the Process Sporadic
2075 Server or Thread Sporadic Server options (and the rest of this section is not further shaded for
2076 these options).

2077 If `_POSIX_SPORADIC_SERVER` or `_POSIX_THREAD_SPORADIC_SERVER` is defined, the
2078 implementation shall include a scheduling policy identified by the value `SCHED_SPORADIC`.

2079 The sporadic server policy is based primarily on two parameters: the *replenishment period* and the
2080 *available execution capacity*. The replenishment period is given by the *sched_ss_repl_period*
2081 member of the **sched_param** structure. The available execution capacity is initialized to the
2082 value given by the *sched_ss_init_budget* member of the same parameter. The sporadic server
2083 policy is identical to the SCHED_FIFO policy with some additional conditions that cause the
2084 thread's assigned priority to be switched between the values specified by the *sched_priority* and
2085 *sched_ss_low_priority* members of the **sched_param** structure.

2086 The priority assigned to a thread using the sporadic server scheduling policy is determined in
2087 the following manner: if the available execution capacity is greater than zero and the number of
2088 pending replenishment operations is strictly less than *sched_ss_max_repl*, the thread is assigned
2089 the priority specified by *sched_priority*; otherwise, the assigned priority shall be
2090 *sched_ss_low_priority*. If the value of *sched_priority* is less than or equal to the value of
2091 *sched_ss_low_priority*, the results are undefined. When active, the thread shall belong to the
2092 thread list corresponding to its assigned priority level, according to the mentioned priority
2093 assignment. The modification of the available execution capacity and, consequently of the
2094 assigned priority, is done as follows:

- 2095 1. When the thread at the head of the *sched_priority* list becomes a running thread, its
 2096 execution time shall be limited to at most its available execution capacity, plus the
 2097 resolution of the execution time clock used for this scheduling policy. This resolution shall
 2098 be implementation-defined.
- 2099 2. Each time the thread is inserted at the tail of the list associated with *sched_priority*—
 2100 because as a blocked thread it became runnable with priority *sched_priority* or because a
 2101 replenishment operation was performed—the time at which this operation is done is
 2102 posted as the *activation_time*.
- 2103 3. When the running thread with assigned priority equal to *sched_priority* becomes a
 2104 preempted thread, it becomes the head of the thread list for its priority, and the execution
 2105 time consumed is subtracted from the available execution capacity. If the available
 2106 execution capacity would become negative by this operation, it shall be set to zero.
- 2107 4. When the running thread with assigned priority equal to *sched_priority* becomes a blocked
 2108 thread, the execution time consumed is subtracted from the available execution capacity,
 2109 and a replenishment operation is scheduled, as described in 6 and 7. If the available
 2110 execution capacity would become negative by this operation, it shall be set to zero.
- 2111 5. When the running thread with assigned priority equal to *sched_priority* reaches the limit
 2112 imposed on its execution time, it becomes the tail of the thread list for
 2113 *sched_ss_low_priority*, the execution time consumed is subtracted from the available
 2114 execution capacity (which becomes zero), and a replenishment operation is scheduled, as
 2115 described in 6 and 7.
- 2116 6. Each time a replenishment operation is scheduled, the amount of execution capacity to be
 2117 replenished, *replenish_amount*, is set equal to the execution time consumed by the thread
 2118 since the *activation_time*. The replenishment is scheduled to occur at *activation_time* plus
 2119 *sched_ss_repl_period*. If the scheduled time obtained is before the current time, the
 2120 replenishment operation is carried out immediately. Several replenishment operations may
 2121 be pending at the same time, each of which will be serviced at its respective scheduled
 2122 time. With the above rules, the number of replenishment operations simultaneously
 2123 pending for a given thread that is scheduled under the sporadic server policy shall not be
 2124 greater than *sched_ss_max_repl*.
- 2125 7. A replenishment operation consists of adding the corresponding *replenish_amount* to the
 2126 available execution capacity at the scheduled time. If, as a consequence of this operation,
 2127 the execution capacity would become larger than *sched_ss_initial_budget*, it shall be
 2128 rounded down to a value equal to *sched_ss_initial_budget*. Additionally, if the thread was
 2129 runnable or running, and had assigned priority equal to *sched_ss_low_priority*, then it
 2130 becomes the tail of the thread list for *sched_priority*.
- 2131 Execution time is defined in Section 2.2.2 (on page 513).
- 2132 For this policy, changing the value of a CPU-time clock via *clock_settime()* shall have no effect on
 2133 its behavior.
- 2134 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_min()*
 2135 and *sched_get_priority_max()* functions when SCHED_SPORADIC is provided as the parameter.
 2136 Conforming implementations shall provide a priority range of at least 32 distinct priorities for
 2137 this policy.

2138 **SCHED_OTHER**

2139 Conforming implementations shall include one scheduling policy identified as SCHED_OTHER
 2140 (which may execute identically with either the FIFO or round robin scheduling policy). The
 2141 effect of scheduling threads with the SCHED_OTHER policy in a system in which other threads
 2142 ss are executing under SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC is implementation-
 2143 defined.

2144 This policy is defined to allow conforming applications to be able to indicate that they no longer
 2145 need a realtime scheduling policy in a portable manner.

2146 For threads executing under this policy, the implementation shall use only priorities within the
 2147 range returned by the *sched_get_priority_max()* and *sched_get_priority_min()* functions when
 2148 SCHED_OTHER is provided as the parameter.

2149 **2.8.5 Clocks and Timers**

2150 TMR The functionality described in this section is dependent on support of the Timers option (and the
 2151 rest of this section is not further shaded for this option).

2152 The <time.h> header defines the types and manifest constants used by the timing facility.

2153 **Time Value Specification Structures**

2154 Many of the timing facility functions accept or return time value specifications. A time value
 2155 structure **timespec** specifies a single time value and includes at least the following members:

2156

2157

2158

2159

Member Type	Member Name	Description
time_t	<i>tv_sec</i>	Seconds.
long	<i>tv_nsec</i>	Nanoseconds.

2160 The *tv_nsec* member is only valid if greater than or equal to zero, and less than the number of
 2161 nanoseconds in a second (1,000 million). The time interval described by this structure is (*tv_sec* *
 2162 10⁹ + *tv_nsec*) nanoseconds.

2163 A time value structure **itimerspec** specifies an initial timer value and a repetition interval for use
 2164 by the per-process timer functions. This structure includes at least the following members:

2165

2166

2167

2168

Member Type	Member Name	Description
struct timespec	<i>it_interval</i>	Timer period.
struct timespec	<i>it_value</i>	Timer expiration.

2169 If the value described by *it_value* is non-zero, it indicates the time to or time of the next timer
 2170 expiration (for relative and absolute timer values, respectively). If the value described by *it_value*
 2171 is zero, the timer shall be disarmed.

2172 If the value described by *it_interval* is non-zero, it specifies an interval to be used in reloading the
 2173 timer when it expires; that is, a periodic timer is specified. If the value described by *it_interval* is
 2174 zero, the timer is disarmed after its next expiration; that is, a one-shot timer is specified.

2175 **Timer Event Notification Control Block**

2176 RTS Per-process timers may be created that notify the process of timer expirations by queuing a
 2177 realtime extended signal. The **sigevent** structure, defined in the Base Definitions volume of
 2178 IEEE Std. 1003.1-200x, <**signal.h**>, is used in creating such a timer. The **sigevent** structure
 2179 contains the signal number and an application-specific data value to be used when notifying the
 2180 calling process of timer expiration events.

2181 **Manifest Constants**

2182 The following constants are defined in the Base Definitions volume of IEEE Std. 1003.1-200x,
 2183 <**time.h**>:

2184 **CLOCK_REALTIME** The identifier for the system-wide realtime clock.

2185 **TIMER_ABSTIME** Flag indicating time is absolute with respect to the clock associated
 2186 with a timer.

2187 MON **CLOCK_MONOTONIC** The identifier for the system-wide monotonic clock, which is defined
 2188 as a clock whose value cannot be set via *clock_settime()* and which
 2189 cannot have backward clock jumps. The maximum possible clock
 2190 jump is implementation-defined.

2191 MON The maximum allowable resolution for the **CLOCK_REALTIME** and the
 2192 **CLOCK_MONOTONIC** clocks and all time services based on these clocks is represented by
 2193 {**_POSIX_CLOCKRES_MIN**} and is defined as 20ms (1/50 of a second). Implementations may
 2194 support smaller values of resolution for these clocks to provide finer granularity time bases. The
 2195 actual resolution supported by an implementation for a specific clock is obtained using the
 2196 *clock_getres()* function. If the actual resolution supported for a time service based on one of these
 2197 clocks differs from the resolution supported for that clock, the implementation shall document
 2198 this difference.

2199 MON The minimum allowable maximum value for the **CLOCK_REALTIME** and the
 2200 **CLOCK_MONOTONIC** clocks and all absolute time services based on them is the same as that
 2201 defined by the ISO C standard for the **time_t** type. If the maximum value supported by a time
 2202 service based on one of these clocks differs from the maximum value supported by that clock,
 2203 the implementation shall document this difference.

2204 **Execution Time Monitoring**

2205 CPT If **_POSIX_CPUTIME** is defined, process CPU-time clocks shall be supported in addition to the
 2206 clocks described in **Manifest Constants**.

2207 TCT If **_POSIX_THREAD_CPUTIME** is defined, thread CPU-time clocks shall be supported.

2208 CPT|TCT CPU-time clocks measure execution or CPU time, which is defined in the Base Definitions
 2209 volume of IEEE Std. 1003.1-200x, Section 3.120, CPU Time (Execution Time). The mechanism
 2210 used to measure execution time is described in the Base Definitions volume of
 2211 IEEE Std. 1003.1-200x, Section 4.7, Measurement of Execution Time.

2212 CPT If **_POSIX_CPUTIME** is defined, the following constant of the type **clockid_t** shall be defined in
 2213 <**time.h**>:

2214 **CLOCK_PROCESS_CPUTIME_ID**

2215 When this value of the type **clockid_t** is used in a *clock()* or *timer*()* function call, it is
 2216 interpreted as the identifier of the CPU-time clock associated with the process making the
 2217 function call.
 2218

2219 TCT If `_POSIX_THREAD_CPUTIME` is defined, the following constant of the type `clockid_t` shall be
2220 defined in `<time.h>`:

2221 `CLOCK_THREAD_CPUTIME_ID`

2222 When this value of the type `clockid_t` is used in a `clock()` or `timer*()` function call, it is
2223 interpreted as the identifier of the CPU-time clock associated with the thread making the
2224 function call.
2225

2226 **2.9 Threads**

2227 THR The functionality described in this section is dependent on support of the Threads option (and
2228 the rest of this section is not further shaded for this option).

2229 This section defines functionality to support multiple flows of control, called *threads*, within a
2230 process. For the definition of *threads*, see the Base Definitions volume of IEEE Std. 1003.1-200x,
2231 Section 3.395, Thread.

2232 The specific functional areas covered by threads and their scope includes:

- 2233 • Thread management: the creation, control, and termination of multiple flows of control in the
2234 same process under the assumption of a common shared address space
- 2235 • Synchronization primitives optimized for tightly coupled operation of multiple control flows
2236 in a common, shared address space

2237 **2.9.1 Thread-Safety**

2238 All functions defined by this volume of IEEE Std. 1003.1-200x shall be thread-safe, except that
2239 the following functions need not be thread-safe.

2240	<i>asctime()</i>	<i>gethostbyname()</i>	<i>getprotobyname()</i>	<i>inet_ntoa()</i>	<i>ttyname()</i>
2241	<i>ctime()</i>	<i>gethostent()</i>	<i>getprotoent()</i>	<i>localtime()</i>	<i>unsetenv()</i>
2242	<i>getc_unlocked()</i>	<i>getlogin()</i>	<i>getpwnam()</i>	<i>putc_unlocked()</i>	<i>wcstombs()</i>
2243	<i>getchar_unlocked()</i>	<i>getnetbyaddr()</i>	<i>getpwuid()</i>	<i>putchar_unlocked()</i>	<i>wctomb()</i>
2244	<i>getenv()</i>	<i>getnetbyname()</i>	<i>getservbyname()</i>	<i>rand()</i>	
2245	<i>getgrgid()</i>	<i>getnetent()</i>	<i>getservbyport()</i>	<i>readdir()</i>	
2246	<i>getgrnam()</i>	<i>getopt()</i>	<i>getservent()</i>	<i>setenv()</i>	
2247	<i>gethostbyaddr()</i>	<i>getprotobyname()</i>	<i>gmtime()</i>	<i>strtok()</i>	

2248 XSI	<i>basename()</i>	<i>dbm_open()</i>	<i>fcvt()</i>	<i>hdestroy()</i>	<i>setgrent()</i>
2249	<i>catgets()</i>	<i>dbm_store()</i>	<i>gcvt()</i>	<i>hsearch()</i>	<i>setkey()</i>
2250	<i>crypt()</i>	<i>dirname()</i>	<i>getdate()</i>	<i>l64a()</i>	<i>setpwent()</i>
2251	<i>dbm_clearerr()</i>	<i>derror()</i>	<i>getenv()</i>	<i>lgamma()</i>	<i>setutxent()</i>
2252	<i>dbm_close()</i>	<i>drand48()</i>	<i>getgrent()</i>	<i>lrand48()</i>	<i>strerror()</i>
2253	<i>dbm_delete()</i>	<i>ecvt()</i>	<i>getpwent()</i>	<i>mrnd48()</i>	
2254	<i>dbm_error()</i>	<i>encrypt()</i>	<i>getutxent()</i>	<i>nl_langinfo()</i>	
2255	<i>dbm_fetch()</i>	<i>endgrent()</i>	<i>getutxid()</i>	<i>ptsname()</i>	
2256	<i>dbm_firstkey()</i>	<i>endpwent()</i>	<i>getutxline()</i>	<i>putenv()</i>	
2257	<i>dbm_nextkey()</i>	<i>endutxent()</i>	<i>hcreate()</i>	<i>pututxline()</i>	

2258

2259 The *read()* function need not be thread-safe when reading from a pipe, FIFO, socket, or terminal
2260 device.

2261 **Note:** While a read from a pipe of {PIPE_MAX}*2 bytes may not generate a single atomic
2262 and thread-safe stream of bytes, it should generate “several” (individually atomic)
2263 thread-safe streams of bytes. Similarly, while reading from a terminal device may
2264 not generate a single atomic and thread-safe stream of bytes, it should generate some
2265 finite number of (individually atomic) and thread-safe streams of bytes. That is,
2266 concurrent calls to read for a pipe, FIFO, or terminal device are not allowed to result
2267 in corrupting the stream of bytes or other internal data. However, *read()*, in these
2268 cases, is not required to return a single contiguous and atomic stream of bytes.

2269 The *ctermid()* and *tmpnam()* functions need not be thread-safe if passed a NULL argument. The
 2270 *wcrtomb()* and *wcsrtombs()* functions need not be thread-safe if passed a NULL *ps* argument.

2271 Implementations shall provide internal synchronization as necessary in order to satisfy this
 2272 requirement.

2273 2.9.2 Thread IDs

2274 Although implementations may have thread IDs that are unique in a system, applications
 2275 should only assume that thread IDs are usable and unique within a single process. The effect of
 2276 calling any of the functions defined in this volume of IEEE Std. 1003.1-200x and passing as an
 2277 argument the thread ID of a thread from another process is unspecified. A conforming
 2278 implementation is free to reuse a thread ID after the thread terminates if it was created with the
 2279 *detachstate* attribute set to `PTHREAD_CREATE_DETACHED` or if *pthread_detach()* or
 2280 *pthread_join()* has been called for that thread. If a thread is detached, its thread ID is invalid for
 2281 use as an argument in a call to *pthread_detach()* or *pthread_join()*.

2282 2.9.3 Thread Mutexes

2283 A thread that has blocked shall not prevent any unblocked thread that is eligible to use the same
 2284 processing resources from eventually making forward progress in its execution. Eligibility for
 2285 processing resources is determined by the scheduling policy.

2286 A thread becomes the owner of a mutex, *m*, when one of the following occurs:

- 2287 • It returns successfully from *pthread_mutex_lock()* with *m* as the *mutex* argument.
- 2288 • It returns successfully from *pthread_mutex_trylock()* with *m* as the *mutex* argument.
- 2289 TMO • It returns successfully from *pthread_mutex_timedwait()* with *m* as the *mutex* argument.
- 2290 • It returns (successfully or not) from *pthread_cond_wait()* with *m* as the *mutex* argument
 2291 (except as explicitly indicated otherwise for certain errors).
- 2292 • It returns (successfully or not) from *pthread_cond_timedwait()* with *m* as the *mutex* argument
 2293 (except as explicitly indicated otherwise for certain errors).

2294 The thread remains the owner of *m* until one of the following occurs:

- 2295 • It executes *pthread_mutex_unlock()* with *m* as the *mutex* argument
- 2296 • It blocks in a call to *pthread_cond_wait()* with *m* as the *mutex* argument.
- 2297 • It blocks in a call to *pthread_cond_timedwait()* with *m* as the *mutex* argument.

2298 The implementation behaves as if at all times there is at most one owner of any mutex.

2299 A thread that becomes the owner of a mutex is said to have *acquired* the mutex and the mutex is
 2300 said to have become *locked*; when a thread gives up ownership of a mutex it is said to have
 2301 *released* the mutex and the mutex is said to have become *unlocked*.

2302 **2.9.4 Thread Scheduling**2303 **Thread Scheduling Attributes**

2304 Thread scheduling attributes are dependent on support of the Thread Execution Scheduling
2305 option.

2306 In support of the scheduling function, threads have attributes which are accessed through the
2307 *pthread_attr_t* thread creation attributes object.

2308 The *contentionscope* attribute defines the scheduling contention scope of the thread to be either
2309 PTHREAD_SCOPE_PROCESS or PTHREAD_SCOPE_SYSTEM.

2310 The *inheritsched* attribute specifies whether a newly created thread is to inherit the scheduling
2311 attributes of the creating thread or to have its scheduling values set according to the other
2312 scheduling attributes in the *pthread_attr_t* object.

2313 The *schedpolicy* attribute defines the scheduling policy for the thread. The *schedparam* attribute
2314 defines the scheduling parameters for the thread. The interaction of threads having different
2315 policies within a process is described as part of the definition of those policies.

2316 If the Thread Execution Scheduling option is defined, and the *schedpolicy* attribute specifies one
2317 of the priority-based policies defined under this option, the *schedparam* attribute contains the
2318 scheduling priority of the thread. A conforming implementation ensures that the priority value
2319 in *schedparam* is in the range associated with the scheduling policy when the thread attributes
2320 object is used to create a thread, or when the scheduling attributes of a thread are dynamically
2321 modified. The meaning of the priority value in *schedparam* is the same as that of *priority*.

2322 TSP If *_POSIX_THREAD_SPORADIC_SERVER* is defined, the *schedparam* attribute supports four
2323 new members that are used for the sporadic server scheduling policy. These members are
2324 *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and *sched_ss_max_repl*. The
2325 meaning of these attributes is the same as in the definitions that appear under Section 2.8.4 (on
2326 page 546).

2327 When a process is created, its single thread has a scheduling policy and associated attributes
2328 equal to the process' policy and attributes. The default scheduling contention scope value is
2329 implementation-defined. The default values of other scheduling attributes are implementation-
2330 defined.

2331 **Thread Scheduling Contention Scope**

2332 The scheduling contention scope of a thread defines the set of threads with which the thread
2333 competes for use of the processing resources. The scheduling operation selects at most one
2334 thread to execute on each processor at any point in time and the thread's scheduling attributes
2335 (for example, *priority*), whether under process scheduling contention scope or system scheduling
2336 contention scope, are the parameters used to determine the scheduling decision.

2337 The scheduling contention scope, in the context of scheduling a mixed scope environment,
2338 effects threads as follows:

- 2339 • A thread created with PTHREAD_SCOPE_SYSTEM scheduling contention scope contends
2340 for resources with all other threads in the same scheduling allocation domain relative to their
2341 system scheduling attributes. The system scheduling attributes of a thread created with
2342 PTHREAD_SCOPE_SYSTEM scheduling contention scope are the scheduling attributes with
2343 which the thread was created. The system scheduling attributes of a thread created with
2344 PTHREAD_SCOPE_PROCESS scheduling contention scope are the implementation-defined
2345 mapping into system attribute space of the scheduling attributes with which the thread was

- 2346 created.
- 2347 • Threads created with PTHREAD_SCOPE_PROCESS scheduling contention scope contend
2348 directly with other threads within their process that were created with
2349 PTHREAD_SCOPE_PROCESS scheduling contention scope. The contention is resolved
2350 based on the threads' scheduling attributes and policies. It is unspecified how such threads
2351 are scheduled relative to threads in other processes or threads with
2352 PTHREAD_SCOPE_SYSTEM scheduling contention scope.
 - 2353 • Conforming implementations shall support the PTHREAD_SCOPE_PROCESS scheduling
2354 contention scope, the PTHREAD_SCOPE_SYSTEM scheduling contention scope, or both.

2355 Scheduling Allocation Domain

2356 Implementations shall support scheduling allocation domains containing one or more
2357 processors. It should be noted that the presence of multiple processors does not automatically
2358 indicate a scheduling allocation domain size greater than one. Conforming implementations on
2359 multiprocessors may map all or any subset of the CPUs to one or multiple scheduling allocation
2360 domains, and could define these scheduling allocation domains on a per-thread, per-process, or
2361 per-system basis, depending on the types of applications intended to be supported by the
2362 implementation. The scheduling allocation domain is independent of scheduling contention
2363 scope, as the scheduling contention scope merely defines the set of threads with which a thread
2364 contends for processor resources, while scheduling allocation domain defines the set of
2365 processors for which it contends. The semantics of how this contention is resolved among
2366 threads for processors is determined by the scheduling policies of the threads.

2367 The choice of scheduling allocation domain size and the level of application control over
2368 scheduling allocation domains is implementation-defined. Conforming implementations may
2369 change the size of scheduling allocation domains and the binding of threads to scheduling
2370 allocation domains at any time.

2371 For application threads with scheduling allocation domains of size equal to one, the scheduling
2372 rules defined for SCHED_FIFO and SCHED_RR shall be used; see **Scheduling Policies** (on page
2373 546). All threads with system scheduling contention scope, regardless of the processes in which
2374 they reside, compete for the processor according to their priorities. Threads with process
2375 scheduling contention scope compete only with other threads with process scheduling
2376 contention scope within their process.

2377 For application threads with scheduling allocation domains of size greater than one, the rules
2378 TSP defined for SCHED_FIFO, SCHED_RR, and SCHED_SPORADIC shall be used in an
2379 implementation-defined manner. Each thread with system scheduling contention scope
2380 competes for the processors in its scheduling allocation domain in an implementation-defined
2381 manner according to its priority. Threads with process scheduling contention scope are
2382 scheduled relative to other threads within the same scheduling contention scope in the process.

2383 TSP If _POSIX_THREAD_SPORADIC_SERVER is defined, the rules defined for SCHED_SPORADIC
2384 in **Scheduling Policies** (on page 546) shall be used in an implementation-defined manner for
2385 application threads whose scheduling allocation domain size is greater than one.

2386 **Scheduling Documentation**

2387 If `_POSIX_PRIORITY_SCHEDULING` is defined, then any scheduling policies beyond
 2388 TSP `SCHED_OTHER`, `SCHED_FIFO`, `SCHED_RR`, and `SCHED_SPORADIC`, as well as the effects of
 2389 the scheduling policies indicated by these other values, and the attributes required in order to
 2390 support such a policy, are implementation-defined. Furthermore, the implementation shall
 2391 document the effect of all processor scheduling allocation domain values supported for these
 2392 policies.

2393 **2.9.5 Thread Cancellation**

2394 The thread cancellation mechanism allows a thread to terminate the execution of any other
 2395 thread in the process in a controlled manner. The target thread (that is, the one that is being
 2396 canceled) is allowed to hold cancellation requests pending in a number of ways and to perform
 2397 application-specific cleanup processing when the notice of cancellation is acted upon.

2398 Cancellation is controlled by the cancellation control functions. Each thread maintains its own
 2399 cancelability state. Cancellation may only occur at cancellation points or when the thread is
 2400 asynchronously cancelable.

2401 The thread cancellation mechanism described in this section depends upon programs having set
 2402 *deferred cancelability* state, which is specified as the default. Applications shall also carefully
 2403 follow static lexical scoping rules in their execution behavior. For example, use of *setjmp()*,
 2404 *return*, *goto*, and so on, to leave user-defined cancellation scopes without doing the necessary
 2405 scope pop operation results in undefined behavior.

2406 Use of asynchronous cancelability while holding resources which potentially need to be released
 2407 may result in resource loss. Similarly, cancellation scopes may only be safely manipulated
 2408 (pushed and popped) when the thread is in the *deferred* or *disabled* cancelability states.

2409 **2.9.5.1 Cancelability States**

2410 The cancelability state of a thread determines the action taken upon receipt of a cancellation
 2411 request. The thread may control cancellation in a number of ways.

2412 Each thread maintains its own cancelability state, which may be encoded in two bits:

- 2413 1. **Cancelability-Enable:** When cancelability is `PTHREAD_CANCEL_DISABLE` (as defined in
 2414 the Base Definitions volume of IEEE Std. 1003.1-200x, `<pthread.h>`), cancellation requests
 2415 against the target thread are held pending. By default, cancelability is set to
 2416 `PTHREAD_CANCEL_ENABLE` (as defined in `<pthread.h>`).
- 2417 2. **Cancelability Type:** When cancelability is enabled and the cancelability type is
 2418 `PTHREAD_CANCEL_ASYNCHRONOUS` (as defined in `<pthread.h>`), new or pending
 2419 cancellation requests may be acted upon at any time. When cancelability is enabled and the
 2420 cancelability type is `PTHREAD_CANCEL_DEFERRED` (as defined in `<pthread.h>`),
 2421 cancellation requests are held pending until a cancellation point (see below) is reached. If
 2422 cancelability is disabled, the setting of the cancelability type has no immediate effect as all
 2423 cancellation requests are held pending; however, once cancelability is enabled again the
 2424 new type is in effect. The cancelability type is `PTHREAD_CANCEL_DEFERRED` in all
 2425 newly created threads including the thread in which *main()* was first invoked.

2426 2.9.5.2 Cancellation Points

2427 Cancellation points occur when a thread is executing the following functions:

2428	<i>accept()</i>	<i>mq_timedsend()</i>	<i>putpmsg()</i>	<i>sigsuspend()</i>
2429	<i>aio_suspend()</i>	<i>msgrcv()</i>	<i>pwrite()</i>	<i>sigtimedwait()</i>
2430	<i>clock_nanosleep()</i>	<i>msgsnd()</i>	<i>read()</i>	<i>sigwait()</i>
2431	<i>close()</i>	<i>msync()</i>	<i>readv()</i>	<i>sigwaitinfo()</i>
2432	<i>connect()</i>	<i>nanosleep()</i>	<i>recv()</i>	<i>sleep()</i>
2433	<i>creat()</i>	<i>open()</i>	<i>recvfrom()</i>	<i>system()</i>
2434	<i>fcntl()</i> ¹	<i>pause()</i>	<i>recvmsg()</i>	<i>tcdrain()</i>
2435	<i>fsync()</i>	<i>poll()</i>	<i>select()</i>	<i>usleep()</i>
2436	<i>getmsg()</i>	<i>pread()</i>	<i>sem_timedwait()</i>	<i>wait()</i>
2437	<i>getpmsg()</i>	<i>pthread_cond_timedwait()</i>	<i>sem_wait()</i>	<i>waitid()</i>
2438	<i>lockf()</i>	<i>pthread_cond_wait()</i>	<i>send()</i>	<i>waitpid()</i>
2439	<i>mq_receive()</i>	<i>pthread_join()</i>	<i>sendmsg()</i>	<i>write()</i>
2440	<i>mq_send()</i>	<i>pthread_testcancel()</i>	<i>sendto()</i>	<i>writev()</i>
2441	<i>mq_timedreceive()</i>	<i>putmsg()</i>	<i>sigpause()</i>	

2442 _____

2443 1. When the *cmd* argument is F_SETLKW.

2444 A cancelation point may also occur when a thread is executing the following functions:

2445	<i>catclose()</i>	<i>ftell()</i>	<i>getwc()</i>	<i>pthread_rwlock_wrlock()</i>
2446	<i>catgets()</i>	<i>ftello()</i>	<i>getwchar()</i>	<i>putc()</i>
2447	<i>catopen()</i>	<i>ftw()</i>	<i>getwd()</i>	<i>putc_unlocked()</i>
2448	<i>closedir()</i>	<i>fwprintf()</i>	<i>glob()</i>	<i>putchar()</i>
2449	<i>closelog()</i>	<i>fwrite()</i>	<i>iconv_close()</i>	<i>putchar_unlocked()</i>
2450	<i>ctermid()</i>	<i>fwscanf()</i>	<i>iconv_open()</i>	<i>puts()</i>
2451	<i>dbm_close()</i>	<i>getc()</i>	<i>ioctl()</i>	<i>pututxline()</i>
2452	<i>dbm_delete()</i>	<i>getc_unlocked()</i>	<i>lseek()</i>	<i>putwc()</i>
2453	<i>dbm_fetch()</i>	<i>getchar()</i>	<i>mkstemp()</i>	<i>putwchar()</i>
2454	<i>dbm_nextkey()</i>	<i>getchar_unlocked()</i>	<i>nftw()</i>	<i>readdir()</i>
2455	<i>dbm_open()</i>	<i>getcwd()</i>	<i>opendir()</i>	<i>readdir_r()</i>
2456	<i>dbm_store()</i>	<i>getdate()</i>	<i>openlog()</i>	<i>remove()</i>
2457	<i>dlclose()</i>	<i>getgrent()</i>	<i>pclose()</i>	<i>rename()</i>
2458	<i>dlopen()</i>	<i>getgrgid()</i>	<i>perror()</i>	<i>rewind()</i>
2459	<i>endgrent()</i>	<i>getgrgid_r()</i>	<i>popen()</i>	<i>rewinddir()</i>
2460	<i>endhostent()</i>	<i>getgrnam()</i>	<i>posix_fadvise()</i>	<i>scanf()</i>
2461	<i>endnetent()</i>	<i>getgrnam_r()</i>	<i>posix_fallocate()</i>	<i>seekdir()</i>
2462	<i>endprotoent()</i>	<i>gethostbyaddr()</i>	<i>posix_madvise()</i>	<i>semop()</i>
2463	<i>endpwent()</i>	<i>gethostbyname()</i>	<i>posix_spawn()</i>	<i>setgrent()</i>
2464	<i>endservent()</i>	<i>gethostent()</i>	<i>posix_spawnnp()</i>	<i>sethostent()</i>
2465	<i>endutxent()</i>	<i>gethostname()</i>	<i>posix_trace_clear()</i>	<i>setnetent()</i>
2466	<i>fclose()</i>	<i>getlogin()</i>	<i>posix_trace_close()</i>	<i>setprotoent()</i>
2467	<i>fcntl()</i> ²	<i>getlogin_r()</i>	<i>posix_trace_create()</i>	<i>setpwent()</i>
2468	<i>fflush()</i>	<i>getnetbyaddr()</i>	<i>posix_trace_create_withlog()</i>	<i>setservent()</i>
2469	<i>fgetc()</i>	<i>getnetbyname()</i>	<i>posix_trace_eventtypelist_getnext_id()</i>	<i>setutxent()</i>
2470	<i>fgetpos()</i>	<i>getnetent()</i>	<i>posix_trace_eventtypelist_rewind()</i>	<i>strerror()</i>
2471	<i>fgets()</i>	<i>getprotobyname()</i>	<i>posix_trace_flush()</i>	<i>syslog()</i>
2472	<i>fgetwc()</i>	<i>getprotobyname_r()</i>	<i>posix_trace_get_attr()</i>	<i>tmpfile()</i>
2473	<i>fgetws()</i>	<i>getprotoent()</i>	<i>posix_trace_get_filter()</i>	<i>tmpnam()</i>
2474	<i>fopen()</i>	<i>getpwent()</i>	<i>posix_trace_get_status()</i>	<i>ttyname()</i>
2475	<i>fprintf()</i>	<i>getpwnam()</i>	<i>posix_trace_getnext_event()</i>	<i>ttyname_r()</i>
2476	<i>fputc()</i>	<i>getpwnam_r()</i>	<i>posix_trace_open()</i>	<i>ungetc()</i>
2477	<i>fputs()</i>	<i>getpwuid()</i>	<i>posix_trace_rewind()</i>	<i>ungetwc()</i>
2478	<i>fputwc()</i>	<i>getpwuid_r()</i>	<i>posix_trace_set_filter()</i>	<i>unlink()</i>
2479	<i>fputws()</i>	<i>gets()</i>	<i>posix_trace_shutdown()</i>	<i>vfprintf()</i>
2480	<i>fread()</i>	<i>getservbyname()</i>	<i>posix_trace_timedgetnext_event()</i>	<i>vwprintf()</i>
2481	<i>freopen()</i>	<i>getservbyport()</i>	<i>posix_typed_mem_open()</i>	<i>vprintf()</i>
2482	<i>fscanf()</i>	<i>getservent()</i>	<i>printf()</i>	<i>vwprintf()</i>
2483	<i>fseek()</i>	<i>getutxent()</i>	<i>pthread_rwlock_rdlock()</i>	<i>wprintf()</i>
2484	<i>fseeko()</i>	<i>getutxid()</i>	<i>pthread_rwlock_timedrdlock()</i>	<i>wscanf()</i>
2485	<i>fsetpos()</i>	<i>getutxline()</i>	<i>pthread_rwlock_timedwrlock()</i>	

2486 An implementation shall not introduce cancelation points into any other functions specified in
2487 this volume of IEEE Std. 1003.1-200x.

2488 _____
2489 2. For any value of the *cmd* argument.

2490 The side effects of acting upon a cancelation request while suspended during a call of a function
2491 are the same as the side effects that may be seen in a single-threaded program when a call to a
2492 function is interrupted by a signal and the given function returns [EINTR]. Any such side effects
2493 occur before any cancelation cleanup handlers are called.

2494 Whenever a thread has cancelability enabled and a cancelation request has been made with that
2495 thread as the target and the thread calls *pthread_testcancel()*, then the cancelation request is acted
2496 upon before *pthread_testcancel()* returns. If a thread has cancelability enabled and the thread has
2497 a cancelation request pending and the thread is suspended at a cancelation point waiting for an
2498 event to occur, then the cancelation request shall be acted upon. However, if the thread is
2499 suspended at a cancelation point and the event that it is waiting for occurs before the cancelation
2500 request is acted upon, it is unspecified whether the cancelation request is acted upon or whether
2501 the request remains pending and the thread resumes normal execution.

2502 2.9.5.3 Thread Cancelation Cleanup Handlers

2503 Each thread maintains a list of cancelation cleanup handlers. The programmer uses the
2504 *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions to place routines on and remove
2505 routines from this list.

2506 When a cancelation request is acted upon, the routines in the list are invoked one by one in LIFO
2507 sequence; that is, the last routine pushed onto the list (Last In) is the first to be invoked (First
2508 Out). The thread invokes the cancelation cleanup handler with cancelation disabled until the last
2509 cancelation cleanup handler returns. When the cancelation cleanup handler for a scope is
2510 invoked, the storage for that scope remains valid. If the last cancelation cleanup handler returns,
2511 thread execution is terminated and a status of PTHREAD_CANCELED is made available to any
2512 threads joining with the target. The symbolic constant PTHREAD_CANCELED expands to a
2513 constant expression of type (**void***) whose value matches no pointer to an object in memory nor
2514 the value NULL.

2515 The cancelation cleanup handlers are also invoked when the thread calls *pthread_exit()*.

2516 A side effect of acting upon a cancelation request while in a condition variable wait is that the
2517 mutex is re-acquired before calling the first cancelation cleanup handler. In addition, the thread
2518 is no longer considered to be waiting for the condition and the thread shall not have consumed
2519 any pending condition signals on the condition.

2520 A cancelation cleanup handler cannot exit via *longjmp()* or *siglongjmp()*.

2521 2.9.5.4 Async-Cancel Safety

2522 The *pthread_cancel()*, *pthread_setcancelstate()*, and *pthread_setcanceltype()* functions are defined to
2523 be async-cancel safe.

2524 No other functions in this volume of IEEE Std. 1003.1-200x are required to be async-cancel-safe.

2525 2.9.6 Thread Read-Write Locks

2526 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous
2527 read-only access to data while allowing only one thread to have exclusive write access at any
2528 given time. They are typically used to protect data that is read-only more frequently than it is
2529 changed.

2530 One or more readers acquire read access to the resource by performing a read lock operation on
2531 the associated read-write lock. A writer acquires exclusive write access by performing a write
2532 lock operation. Basically, all readers exclude any writers and a writer excludes all readers and
2533 any other writers.

2534 A thread that has blocked on a read-write lock (for example, has not yet returned from a
2535 *pthread_rwlock_rdlock()* or *pthread_rwlock_wrlock()* call) shall not prevent any unblocked thread
2536 that is eligible to use the same processing resources from eventually making forward progress in
2537 its execution. Eligibility for processing resources shall be determined by the scheduling policy.

2538 Read-write locks can be used to synchronize threads in the current process and other processes if
2539 they are allocated in memory that is writable and shared among the cooperating processes and
2540 have been initialized for this behavior.

2541 **2.9.7 Thread Interactions with Regular File Operations**

2542 All of the functions *chmod()*, *close()*, *fchmod()*, *fcntl()*, *fstat()*, *ftruncate()*, *lseek()*, *open()*, *read()*,
2543 *readlink()*, *stat()*, *symlink()*, and *write()* shall be atomic with respect to each other in the effects
2544 specified in IEEE Std. 1003.1-200x when they operate on regular files. If two threads each call one
2545 of these functions, each call shall either see all of the specified effects of the other call, or none of
2546 them.

2547 **2.10 Sockets**

2548 A socket is an endpoint for communication using the facilities described in this section. A socket
2549 is created with a specific socket type, described in Section 2.10.6 (on page 563), and is associated
2550 with a specific protocol, detailed in Section 2.10.2. A socket is accessed via a file descriptor
2551 obtained when the socket is created.

2552 **2.10.1 Protocol Families**

2553 All network protocols are associated with a specific protocol family. A protocol family provides
2554 basic services to the protocol implementation to allow it to function within a specific network
2555 environment. These services may include packet fragmentation and reassembly, routing,
2556 addressing, and basic transport. A protocol family may support multiple methods of addressing.
2557 Each method represents an address family. A protocol family is normally comprised of a
2558 number of protocols, one per socket type. Each protocol is characterized by an abstract socket
2559 type. It is not required that a protocol family support all socket types. A protocol family may
2560 contain multiple protocols supporting the same socket abstraction.

2561 Section 2.10.17 (on page 569), Section 2.10.18 (on page 569), and Section 2.10.19 (on page 570),
2562 respectively, describe the use of sockets for local UNIX connections, for Internet protocols based
2563 on IPv4, and for Internet protocols based on IPv6.

2564 **2.10.2 Protocols**

2565 A protocol supports one of the socket abstractions detailed in Section 2.10.6 (on page 563).
2566 Selecting a protocol involves specifying the protocol family, socket type, and protocol number to
2567 the *socket()* function. Protocols normally accept only one type of address format, usually
2568 determined by the addressing structure inherent in the design of the protocol family/network
2569 architecture. Certain semantics of the basic socket abstractions are protocol-specific. All
2570 protocols are expected to support the basic model for their particular socket type, but may, in
2571 addition, provide non-standard facilities or extensions to a mechanism.

2572 **2.10.3 Addressing**

2573 Associated with each protocol family is at least one address family. An address family defines
2574 the format of a socket address. All network addresses are described using a general structure,
2575 called a **sockaddr**, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x,
2576 <sys/socket.h>. However, each address family imposes finer and more specific structure,
2577 generally defining a structure with fields specific to the address family. The field *sa_family* in the
2578 **sockaddr** structure contains the address family identifier, specifying the format of the *sa_data*
2579 area. The size of the *sa_data* area is unspecified.

2580 **2.10.4 Routing**

2581 Sockets provides packet routing facilities. A routing information database is maintained, which
2582 is used in selecting the appropriate network interface when transmitting packets.

2583 2.10.5 Interfaces

2584 Each network interface in a system corresponds to a path through which messages can be sent
2585 and received. A network interface usually has a hardware device associated with it, though
2586 certain interfaces such as the loopback interface, do not.

2587 2.10.6 Socket Types

2588 A socket is created with a specific type, which defines the communication semantics and which
2589 allows the selection of an appropriate communication protocol. Three types are defined:
2590 SOCK_STREAM, SOCK_SEQPACKET, and SOCK_DGRAM. Implementations may specify
2591 additional socket types.

2592 The SOCK_STREAM socket type provides reliable, sequenced, full-duplex octet streams
2593 between the socket and a peer to which the socket is connected. A socket of type
2594 SOCK_STREAM must be in a connected state before any data may be sent or received. Record
2595 boundaries are not maintained; data sent on a stream socket using output operations of one size
2596 may be received using input operations of smaller or larger sizes without loss of data. Data may
2597 be buffered; successful return from an output function does not imply that the data has been
2598 delivered to the peer or even transmitted from the local system. If data cannot be successfully
2599 transmitted within a given time then the connection is considered broken, and subsequent
2600 operations shall fail. A SIGPIPE signal is raised if a thread sends on a broken stream (one that is
2601 no longer connected). Support for an out-of-band data transmission facility is protocol-specific.

2602 The SOCK_SEQPACKET socket type is similar to the SOCK_STREAM type, and is also
2603 connection-oriented. The only difference between these types is that record boundaries are
2604 maintained using the SOCK_SEQPACKET type. A record can be sent using one or more output
2605 operations and received using one or more input operations, but a single operation never
2606 transfers parts of more than one record. Record boundaries are visible to the receiver via the
2607 MSG_EOR flag in the received message flags returned by the *recvmsg()* function. It is protocol-
2608 specific whether a maximum record size is imposed.

2609 The SOCK_DGRAM socket type supports connectionless data transfer which is not necessarily
2610 acknowledged or reliable. Datagrams may be sent to a peer named in each output operation, and
2611 incoming datagrams may be received from multiple sources. The source address of each
2612 datagram is available when receiving the datagram. An application may also pre-specify a peer
2613 address, in which case calls to output functions shall send to the pre-specified peer. If a peer has
2614 been specified, only datagrams from that peer shall be received. A datagram must be sent in a
2615 single output operation, and must be received in a single input operation. The maximum size of
2616 a datagram is protocol-specific; with some protocols, the limit is implementation-defined.
2617 Output datagrams may be buffered within the system; thus, a successful return from an output
2618 function does not guarantee that a datagram is actually sent or received. However,
2619 implementations should attempt to detect any errors possible before the return of an output
2620 function, reporting any error by an unsuccessful return value.

2621 2.10.7 Socket I/O Mode

2622 The I/O mode of a socket is described by the O_NONBLOCK file status flag which pertains to
2623 the open file description for the socket. This flag is initially off when a socket is created, but may
2624 be set and cleared by the use of the F_SETFL command of the *fcntl()* function.

2625 When the O_NONBLOCK flag is set, functions that would normally block until they are
2626 complete either return immediately with an error, or they complete asynchronously to the
2627 execution of the calling process. Data transfer operations (the *read()*, *write()*, *send()*, and *recv()*
2628 functions) complete immediately, transfer only as much as is available, and then return without
2629 blocking, or return an error indicating that no transfer could be made without blocking. The

2630 *connect()* function initiates a connection and returns without blocking when `O_NONBLOCK` is
2631 set; it returns the error `[EINPROGRESS]` to indicate that the connection was initiated
2632 successfully, but that it has not yet completed.

2633 **2.10.8 Socket Owner**

2634 The owner of a socket is unset when a socket is created. The owner may be set to a process ID or
2635 process group ID using the `F_SETOWN` command of the *fcntl()* function.

2636 **2.10.9 Socket Queue Limits**

2637 The transmit and receive queue sizes for a socket are set when the socket is created. The default
2638 sizes used are both protocol-specific and implementation-defined. The sizes may be changed
2639 using the *setsockopt()* function.

2640 **2.10.10 Pending Error**

2641 Errors may occur asynchronously, and be reported to the socket in response to input from the
2642 network protocol. The socket stores the pending error to be reported to a user of the socket at the
2643 next opportunity. The error is returned in response to a subsequent *send()*, *recv()*, or *getsockopt()*
2644 operation on the socket, and the pending error is then cleared.

2645 **2.10.11 Socket Receive Queue**

2646 A socket has a receive queue that buffers data when they are received by the system until they
2647 are removed by a receive call. Depending on the type of the socket and the communication
2648 provider, the receive queue may also contain ancillary data such as the addressing and other
2649 protocol data associated with the normal data in the queue, and may contain out-of-band or
2650 expedited data. The limit on the queue size includes any normal, out-of-band data, datagram
2651 source addresses, and ancillary data in the queue. The description in this section applies to all
2652 sockets, even though some elements cannot be present in some instances.

2653 The contents of a receive buffer are logically structured as a series of data segments with
2654 associated ancillary data and other information. A data segment may contain normal data or
2655 out-of-band data, but never both. A data segment may complete a record if the protocol
2656 supports records (always true for types `SOCK_SEQPACKET` and `SOCK_DGRAM`). A record
2657 may be stored as more than one segment; the complete record might never be present in the
2658 receive buffer at one time, as a portion might already have been returned to the application, and
2659 another portion might not yet have been received from the communications provider. A data
2660 segment may contain ancillary protocol data, which is logically associated with the segment.
2661 Ancillary data is received as if it were queued along with the first normal data octet in the
2662 segment (if any). A segment may contain ancillary data only, with no normal or out-of-band
2663 data. For the purposes of this section, a datagram is considered to be a data segment that
2664 terminates a record, and that includes a source address as a special type of ancillary data. Data
2665 segments are placed into the queue as data is delivered to the socket by the protocol. Normal
2666 data segments are placed at the end of the queue as they are delivered. If a new segment
2667 contains the same type of data as the preceding segment and includes no ancillary data, and if
2668 the preceding segment does not terminate a record, the segments are logically merged into a
2669 single segment.

2670 The receive queue is logically terminated if an end-of-file indication has been received or a
2671 connection has been terminated. A segment shall be considered to be terminated if another
2672 segment follows it in the queue, if the segment completes a record, or if an end-of-file or other
2673 connection termination has been reported. The last segment in the receive queue shall also be
2674 considered to be terminated while the socket has a pending error to be reported.

2675 A receive operation shall never return data or ancillary data from more than one segment.

2676 **2.10.12 Socket Out-of-Band Data State**

2677 The handling of received out-of-band data is protocol-specific. Out-of-band data may be placed
2678 in the socket receive queue, either at the end of the queue or before all normal data in the queue.
2679 In this case, out-of-band data is returned to an application program by a normal receive call.
2680 Out-of-band data may also be queued separately rather than being placed in the socket receive
2681 queue, in which case it shall be returned only in response to a receive call that requests out-of-
2682 band data. It is protocol-specific whether an out-of-band data mark is placed in the receive
2683 queue to demarcate data preceding the out-of-band data and following the out-of-band data. An
2684 out-of-band data mark is logically an empty data segment that cannot be merged with other
2685 segments in the queue. An out-of-band data mark is never returned in response to an input
2686 operation. The *socketmark()* function can be used to test whether an out-of-band data mark is the
2687 first element in the queue. If an out-of-band data mark is the first element in the queue when an
2688 input function is called without the MSG_PEEK option, the mark is removed from the queue and
2689 the following data (if any) are processed as if the mark had not been present.

2690 **2.10.13 Connection Indication Queue**

2691 Sockets that are used to accept incoming connections maintain a queue of outstanding
2692 connection indications. This queue is a list of connections that are awaiting acceptance by the
2693 application. See *listen()*.

2694 **2.10.14 Signals**

2695 One category of event at the socket interface is the generation of signals. These signals report
2696 protocol events or process errors relating to the state of the socket. The generation or delivery of
2697 a signal does not change the state of the socket, although the generation of the signal may have
2698 been caused by a state change.

2699 The SIGPIPE signal shall be sent to a thread that attempts to send data on a socket that is no
2700 longer able to send. In addition, the send operation fails with the error [EPIPE].

2701 If a socket has an owner, the SIGURG signal is sent to the owner of the socket when it is notified
2702 of expedited or out-of-band data. The socket state at this time is protocol-dependent, and the
2703 status of the socket is specified in Section 2.10.17 (on page 569), Section 2.10.18 (on page 569),
2704 and Section 2.10.19 (on page 570). Depending on the protocol, the expedited data may or may
2705 not have arrived at the time of signal generation.

2706 **2.10.15 Asynchronous Errors**

2707 If any of the following conditions occur asynchronously for a socket, the corresponding value
2708 listed below shall become the pending error for the socket:

2709 [ECONNABORTED]

2710 The connection was aborted locally.

2711 [ECONNREFUSED]

2712 For a connection-mode socket attempting a non-blocking connection, the attempt to connect
2713 was forcefully rejected. For a connectionless-mode socket, an attempt to deliver a datagram
2714 was forcefully rejected.

2715 [ECONNRESET]

2716 The peer has aborted the connection.

- 2717 [EHOSTDOWN]
 2718 The destination host has been determined to be down or disconnected.
- 2719 [EHOSTUNREACH]
 2720 The destination host is not reachable.
- 2721 [EMSGSIZE]
 2722 For a connectionless-mode socket, the size of a previously sent datagram prevented
 2723 delivery.
- 2724 [ENETDOWN]
 2725 The local network connection is not operational.
- 2726 [ENETRESET]
 2727 The connection was aborted by the network.
- 2728 [ENETUNREACH]
 2729 The destination network is not reachable.

2730 2.10.16 Use of Options

2731 There are a number of socket options which either specialize the behavior of a socket or provide
 2732 useful information. These options may be set at different protocol levels and are always present
 2733 at the uppermost “socket” level.

2734 Socket options are manipulated by two functions, *getsockopt()* and *setsockopt()*. These functions
 2735 allow an application program to customize the behavior and characteristics of a socket to
 2736 provide the desired effect.

2737 All of the options have default values. The type and meaning of these values is defined by the
 2738 protocol level to which they apply. Instead of using the default values, an application program
 2739 may choose to customize one or more of the options. However, in the bulk of cases, the default
 2740 values are sufficient for the application.

2741 Some of the options are used to enable or disable certain behavior within the protocol modules
 2742 (for example, turn on debugging) while others may be used to set protocol-specific information
 2743 (for example, IP time-to-live on all the application’s outgoing packets). As each of the options is
 2744 introduced, its effect on the underlying protocol modules is described.

2745 Table 2-4 shows the value for the socket level.

2746 **Table 2-4** Value of Level for Socket Options

Name	Description
SOL_SOCKET	Options are intended for the sockets level.

2749 Table 2-5 (on page 567) lists those options present at the socket level; that is, when the *level*
 2750 parameter of the *getsockopt()* or *setsockopt()* function is SOL_SOCKET, the types of the option
 2751 value parameters associated with each option, and a brief synopsis of the meaning of the option
 2752 value parameter. Unless otherwise noted, each may be examined with *getsockopt()* and set with
 2753 *setsockopt()* on all types of socket.

2754

Table 2-5 Socket-Level Options

2755

2756

2757

2758

2759

2760

2761

2762

2763

2764

2765

2766

2767

2768

2769

2770

2771

2772

2773

2774

2775

2776

2777

2778

2779

2780

2781

2782

Option	Parameter Type	Parameter Meaning
SO_BROADCAST	(void *) int	Non-zero requests permission to transmit broadcast datagrams (SOCK_DGRAM sockets only).
SO_DEBUG	int	Non-zero requests debugging in underlying protocol modules.
SO_DONTROUTE	int	Non-zero requests bypass of normal routing; route based on destination address only.
SO_ERROR	int	Requests and clears pending error information on the socket (<i>getsockopt()</i> only).
SO_KEEPALIVE	int	Non-zero requests periodic transmission of keepalive messages (protocol-specific).
SO_LINGER	struct linger	Specify actions to be taken for queued, unsent data on <i>close()</i> : linger on/off and linger time in seconds.
SO_OOBINLINE	int	Non-zero requests that out-of-band data be placed into normal data input queue as received.
SO_RCVBUF	int	Size of receive buffer (in bytes).
SO_RCVLOWAT	int	Minimum amount of data to return to application for input operations (in bytes).
SO_RCVTIMEO	struct timeval	Timeout value for a socket receive operation.
SO_REUSEADDR	int	Non-zero requests reuse of local addresses in <i>bind()</i> (protocol-specific).
SO_SNDBUF	int	Size of send buffer (in bytes).
SO_SNDLOWAT	int	Minimum amount of data to send for output operations (in bytes).
SO_SNDTIMEO	struct timeval	Timeout value for a socket send operation.
SO_TYPE	int	Identify socket type (<i>getsockopt()</i> only).

2783

2784

2785

The SO_BROADCAST option requests permission to send broadcast datagrams on the socket. Support for SO_BROADCAST is protocol-specific. The default for SO_BROADCAST is that the ability to send broadcast datagrams on a socket is disabled.

2786

2787

2788

2789

SO_DEBUG enables debugging in the underlying protocol modules. This can be useful for tracing the behavior of the underlying protocol modules during normal system operation. The semantics of the debug reports are implementation-defined. The default value for SO_DEBUG is for debugging to be turned off.

2790

2791

2792

2793

2794

SO_DONTROUTE requests that outgoing messages bypass the standard routing facilities. The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. It is protocol-specific whether this option has any effect and how the outgoing network interface is chosen. Support for this option with each protocol is implementation-defined.

2795

2796

2797

2798

2799

SO_ERROR is used only on *getsockopt()*. When this option is specified, *getsockopt()* returns any pending error on the socket and clears the error status. It returns a value of 0 if there is no pending error. SO_ERROR may be used to check for asynchronous errors on connected connectionless-mode sockets or for other types of asynchronous errors. SO_ERROR has no default value.

2800 SO_KEEPALIVE enables the periodic transmission of messages on a connected socket. The
2801 behavior of this option is protocol-specific. The default value for SO_KEEPALIVE is zero,
2802 specifying that this capability is turned off.

2803 The SO_LINGER option controls the action of the interface when unsent messages are queued
2804 on a socket and a *close()* is performed. The details of this option are protocol-specific. The
2805 default value for SO_LINGER is zero, or off, for the *L_onoff* element of the option value and zero
2806 seconds for the linger time specified by the *L_linger* element.

2807 SO_OOBINLINE is valid only on protocols that support out-of-band data. The SO_OOBINLINE
2808 option requests that out-of-band data be placed in the normal data input queue as received; it is
2809 then accessible using the *read()* or *recv()* functions without the MSG_OOB flag set. The default
2810 for SO_OOBINLINE is off; that is, for out-of-band data not to be placed in the normal data input
2811 queue.

2812 SO_RCVBUF requests that the buffer space allocated for receive operations on this socket be set
2813 to the value, in bytes, of the option value. Applications may wish to increase buffer size for high
2814 volume connections, or may decrease buffer size to limit the possible backlog of incoming data.
2815 The default value for the SO_RCVBUF option value is implementation-defined, and may vary by
2816 protocol.

2817 The maximum value for the option for a socket may be obtained by the use of the *fpathconf()*
2818 function, using the value *_PC_SOCKET_MAXBUF*.

2819 SO_RCVLOWAT sets the minimum number of bytes to process for socket input operations. In
2820 general, receive calls block until any (non-zero) amount of data is received, then return the
2821 smaller of the amount available or the amount requested. The default value for SO_RCVLOWAT
2822 is 1, and does not affect the general case. If SO_RCVLOWAT is set to a larger value, blocking
2823 receive calls normally wait until they have received the smaller of the low water mark value or
2824 the requested amount. Receive calls may still return less than the low water mark if an error
2825 occurs, a signal is caught, or the type of data next in the receive queue is different than that
2826 returned (for example, out-of-band data). As mentioned previously, the default value for
2827 SO_RCVLOWAT is 1 byte. It is implementation-defined whether the SO_RCVLOWAT option
2828 can be set.

2829 SO_RCVTIMEO is an option to set a timeout value for input operations. It accepts a **timeval**
2830 structure with the number of seconds and microseconds specifying the limit on how long to wait
2831 for an input operation to complete. If a receive operation has blocked for this much time without
2832 receiving additional data, it returns with a partial count or *errno* set to [EWOULDBLOCK] if no
2833 data were received. The default for this option is the value zero, which indicates that a receive
2834 operation will not timeout. It is implementation-defined whether the SO_RCVTIMEO option can
2835 be set.

2836 SO_REUSEADDR indicates that the rules used in validating addresses supplied in a *bind()*
2837 should allow reuse of local addresses. Operation of this option is protocol-specific. The default
2838 value for SO_REUSEADDR is off; that is, reuse of local addresses is not permitted.

2839 SO_SNDBUF requests that the buffer space allocated for send operations on this socket be set to
2840 the value, in bytes, of the option value. The default value for the SO_SNDBUF option value is
2841 implementation-defined, and may vary by protocol. The maximum value for the option for a
2842 socket may be obtained by the use of the *fpathconf()* function, using the value
2843 *_PC_SOCKET_MAXBUF*.

2844 SO_SNDLOWAT sets the minimum number of bytes to process for socket output operations.
2845 Most output operations process all of the data supplied by the call, delivering data to the
2846 protocol for transmission and blocking as necessary for flow control. Non-blocking output
2847 operations process as much data as permitted subject to flow control without blocking, but

2848 process no data if flow control does not allow the smaller of the send low water mark value or
2849 the entire request to be processed. A *select()* operation testing the ability to write to a socket
2850 returns true only if the send low water mark could be processed. The default value for
2851 SO_SNDLOWAT is implementation-defined and protocol-specific. It is implementation-defined
2852 whether the SO_SNDLOWAT option can be set.

2853 SO_SNDTIMEO is an option to set a timeout value for the amount of time that an output
2854 function shall block because flow control prevents data from being sent. As noted in Table 2-5
2855 (on page 567), the option value is a **timeval** structure with the number of seconds and
2856 microseconds specifying the limit on how long to wait for an output operation to complete. If a
2857 send operation has blocked for this much time, it returns with a partial count or *errno* set to
2858 [EWOULDBLOCK] if no data were sent. The default for this option is the value zero, which
2859 indicates that a send operation will not timeout. It is implementation-defined whether the
2860 SO_SNDTIMEO option can be set.

2861 SO_TYPE is used only on *getsockopt()*. When this option is specified, *getsockopt()* returns the
2862 type of the socket (for example, SOCK_STREAM). This option is useful to servers that inherit
2863 sockets on start-up. SO_TYPE has no default value.

2864 2.10.17 Use of Sockets for Local UNIX Connections

2865 Support for UNIX domain sockets is mandatory.

2866 UNIX domain sockets provide process-to-process communication in a single system.

2867 2.10.17.1 Headers

2868 Symbolic constant AF_UNIX is defined in the `<sys/socket.h>` header to identify the UNIX
2869 domain address family. The `<sys/un.h>` header contains other definitions used in connection
2870 with UNIX domain sockets. See the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter
2871 13, Headers.

2872 The **sockaddr_storage** structure defined in `<sys/socket.h>` is large enough to accommodate a
2873 **sockaddr_un** structure (see the `<sys/un.h>` header defined in the Base Definitions volume of
2874 IEEE Std. 1003.1-200x, Chapter 13, Headers) and is aligned at an appropriate boundary so that
2875 pointers to it can be cast as pointers to **sockaddr_un** structures and used to access the fields of
2876 those structures without alignment problems. When a **sockaddr_storage** structure is cast as a
2877 **sockaddr_un** structure, the *ss_family* field maps onto the *sun_family* field.

2878 2.10.18 Use of Sockets over Internet Protocols Based on IPv4

2879 Support for sockets over Internet protocols based on IPv4 is mandatory.

2880 2.10.18.1 Headers

2881 Symbolic constant AF_INET is defined in the `<sys/socket.h>` header to identify the IPv4 Internet
2882 address family. The `<netinet/in.h>` header contains other definitions used in connection with
2883 IPv4 Internet sockets. See the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 13,
2884 Headers.

2885 The **sockaddr_storage** structure defined in `<sys/socket.h>` is large enough to accommodate a
2886 **sockaddr_in** structure (see the `<netinet/in.h>` header defined in the Base Definitions volume of
2887 IEEE Std. 1003.1-200x, Chapter 13, Headers) and is aligned at an appropriate boundary so that
2888 pointers to it can be cast as pointers to **sockaddr_in** structures and used to access the fields of
2889 those structures without alignment problems. When a **sockaddr_storage** structure is cast as a
2890 **sockaddr_in** structure, the *ss_family* field maps onto the *sin_family* field.

2891 2.10.19 Use of Sockets over Internet Protocols Based on IPv6

2892 IP6 This section describes extensions to support sockets over Internet protocols based on IPv6. This
2893 functionality is dependent on support of the IPV6 option (and the rest of this section is not
2894 further shaded for this option).

2895 To enable smooth transition from IPv4 to IPv6, the features defined in this section may, in certain
2896 circumstances, also be used in connection with IPv4; see Section 2.10.19.2 (on page 571).

2897 2.10.19.1 Addressing

2898 IPv6 overcomes the addressing limitations of previous versions by using 128-bit addresses
2899 instead of 32-bit addresses. The IPv6 address architecture is described in RFC 2373.

2900 There are three kinds of IPv6 address:

2901 Unicast

2902 Identifies a single interface.

2903 A unicast address can be global, link-local (designed for use on a single link), or site-local
2904 (designed for systems not connected to the Internet). Link-local and site-local addresses
2905 need not be globally unique.

2906 Anycast

2907 Identifies a set of interfaces such that a packet sent to the address can be delivered to any
2908 member of the set.

2909 An anycast address is similar to a unicast address; the nodes to which an anycast address is
2910 assigned must be explicitly configured to know that it is an anycast address.

2911 Multicast

2912 Identifies a set of interfaces such that a packet sent to the address should be delivered to
2913 every member of the set.

2914 An application can send multicast datagrams by simply specifying an IPv6 multicast
2915 address in the *address* argument of *sendto()*. To receive multicast datagrams, an application
2916 must join the multicast group (using *setsockopt()* with *IPV6_JOIN_GROUP*) and must bind
2917 to the socket the UDP port on which datagrams will be received. Some applications should
2918 also bind the multicast group address to the socket, to prevent other datagrams destined to
2919 that port from being delivered to the socket.

2920 A multicast address can be global, node-local, link-local, site-local, or organization-local.

2921 The following special IPv6 addresses are defined:

2922 Unspecified

2923 An address that is not assigned to any interface and is used to indicate the absence of an
2924 address.

2925 Loopback

2926 A unicast address that is not assigned to any interface and can be used by a node to send
2927 packets to itself.

2928 Two sets of IPv6 addresses are defined to correspond to IPv4 addresses:

2929 IPv4-compatible addresses

2930 These are assigned to nodes that support IPv6 and can be used when traffic is “tunneled”
2931 through IPv4.

2932 IPv4-mapped addresses

2933 These are used to represent IPv4 addresses in IPv6 address format; see Section 2.10.19.2 (on

2934 page 571).

2935 Note that the unspecified address and the loopback address must not be treated as IPv4-
2936 compatible addresses.

2937 2.10.19.2 Compatibility with IPv4

2938 The API provides the ability for IPv6 applications to interoperate with applications using IPv4,
2939 by using IPv4-mapped IPv6 addresses. These addresses can be generated automatically by the
2940 *getipnodebyname()* function when the specified host has only IPv4 addresses (as described in
2941 *endhostent()*).

2942 Applications may use AF_INET6 sockets to open TCP connections to IPv4 nodes, or send UDP
2943 packets to IPv4 nodes, by simply encoding the destination's IPv4 address as an IPv4-mapped
2944 IPv6 address, and passing that address, within a **sockaddr_in6** structure, in the *connect()*,
2945 *sendto()* or *sendmsg()* function. When applications use AF_INET6 sockets to accept TCP
2946 connections from IPv4 nodes, or receive UDP packets from IPv4 nodes, the system returns the
2947 peer's address to the application in the *accept()*, *recvfrom()*, *recvmsg()*, or *getpeername()*
2948 function using a **sockaddr_in6** structure encoded this way. If a node has an IPv4 address, then the
2949 implementation may allow applications to communicate using that address via an AF_INET6
2950 socket. In such a case, the address will be represented at the API by the corresponding IPv4-
2951 mapped IPv6 address. Also, the implementation may allow an AF_INET6 socket bound to
2952 **in6addr_any** to receive inbound connections and packets destined to one of the node's IPv4
2953 addresses.

2954 An application may use AF_INET6 sockets to bind to a node's IPv4 address by specifying the
2955 address as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure in the *bind()* function. For
2956 an AF_INET6 socket bound to a node's IPv4 address, the system returns the address in the
2957 *getsockname()* function as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure.

2958 2.10.19.3 Interface Identification

2959 Each local interface is assigned a unique positive integer as a numeric index. Indexes start at 1;
2960 zero is not used. There may be gaps so that there is no current interface for a particular positive
2961 index. Each interface also has a unique implementation-defined name.

2962 2.10.19.4 Options

2963 The following options apply at the IPPROTO_IPV6 level:

2964 IPV6_JOIN_GROUP

2965 When set via *setsockopt()*, it joins the application to a multicast group on an interface
2966 (identified by its index) and addressed by a given multicast address, enabling packets sent
2967 to that address to be read via the socket. If the interface index is specified as zero, the
2968 system selects the interface (for example, by looking up the address in a routing table and
2969 using the resulting interface).

2970 An attempt to read this option using *getsockopt()* results in an [EOPNOTSUPP] error.

2971 The value of this option is an **ipv6_mreq** structure.

2972 IPV6_LEAVE_GROUP

2973 When set via *setsockopt()*, it removes the application from the multicast group on an
2974 interface (identified by its index) and addressed by a given multicast address.

2975 An attempt to read this option using *getsockopt()* results in an [EOPNOTSUPP] error.

- 2976 The value of this option is an **ipv6_mreq** structure.
- 2977 **IPV6_MULTICAST_HOPS**
- 2978 The value of this option is the hop limit for outgoing multicast IPv6 packets sent via the
2979 socket. Its possible values are the same as those of **IPV6_UNICAST_HOPS**. If the
2980 **IPV6_MULTICAST_HOPS** option is not set, a value of 1 is assumed. This option can be set
2981 via *setsockopt()* and read via *getsockopt()*.
- 2982 **IPV6_MULTICAST_IF**
- 2983 The index of the interface to be used for outgoing multicast packets. It can be set via
2984 *setsockopt()* and read via *getsockopt()*.
- 2985 **IPV6_MULTICAST_LOOP**
- 2986 This option controls whether outgoing multicast packets should be delivered back to the
2987 local application when the sending interface is itself a member of the destination multicast
2988 group. If it is set to 1 they are delivered. If it is set to 0 they are not. Other values result in an
2989 [EINVAL] error. This option can be set via *setsockopt()* and read via *getsockopt()*.
- 2990 **IPV6_UNICAST_HOPS**
- 2991 The value of this option is the hop limit for outgoing unicast IPv6 packets sent via the
2992 socket. If the option is not set, or is set to -1, the system selects a default value. Attempts to
2993 set a value less than -1 or greater than 255 result in an [EINVAL] error. This option can be
2994 set via *setsockopt()* and read via *getsockopt()*.
- 2995 An [EOPNOTSUPP] error results if **IPV6_JOIN_GROUP** or **IPV6_LEAVE_GROUP** is used with
2996 *getsockopt()*.
- 2997 **2.10.19.5 Headers**
- 2998 Symbolic constant **AF_INET6** is defined in the **<sys/socket.h>** header to identify the IPv6
2999 Internet address family. See the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 13,
3000 Headers.
- 3001 The **sockaddr_storage** structure defined in **<sys/socket.h>** is large enough to accommodate a
3002 **sockaddr_in6** structure (see the **<netinet/in.h>** header defined in the Base Definitions volume of
3003 IEEE Std. 1003.1-200x, Chapter 13, Headers) and is aligned at an appropriate boundary so that
3004 pointers to it can be cast as pointers to **sockaddr_in6** structures and used to access the fields of
3005 those structures without alignment problems. When a **sockaddr_storage** structure is cast as a
3006 **sockaddr_in6** structure, the *ss_family* field maps onto the *sin6_family* field.
- 3007 The **<netinet/in.h>**, **<arpa/inet.h>**, and **<netdb.h>** headers contain other definitions used in
3008 connection with IPv6 Internet sockets; see the Base Definitions volume of IEEE Std. 1003.1-200x,
3009 Chapter 13, Headers.

3010 2.11 Tracing

3011 TRC This section describes extensions to support tracing of user applications. This functionality is
 3012 dependent on support of the Trace option (and the rest of this section is not further shaded for
 3013 this option).

3014 The tracing facilities defined in IEEE Std. 1003.1-200x allow a process to select a set of trace event
 3015 types, to activate a trace stream of the selected trace events as they occur in the flow of
 3016 execution, and to retrieve the recorded trace events.

3017 The tracing operation relies on three logically different components: the traced process, the
 3018 controller process, and the analyzer process. During the execution of the traced process, when a
 3019 trace point is reached, a trace event is recorded into the trace streams created for that process in
 3020 which the associated trace event type identifier is not being filtered out. The controller process
 3021 controls the operation of recording the trace events into the trace stream. It shall be able to:

- 3022 • Initialize the attributes of a trace stream
- 3023 • Create the trace stream (for a specified traced process) using those attributes
- 3024 • Start and stop tracing for the trace stream
- 3025 • Filter the type of trace events to be recorded, if the Trace Event Filter option is supported
- 3026 • Shut a trace stream down

3027 These operations can be done for an active trace stream. The analyzer process retrieves the
 3028 traced events either at runtime, when the trace stream has not yet been shut down, but is still
 3029 recording trace events; or after opening a trace log that had been previously recorded and shut
 3030 down. These three logically different operations can be performed by the same process, or can be
 3031 distributed into different processes.

3032 A trace stream identifier can be created by a call to *posix_trace_create()*,
 3033 *posix_trace_create_withlog()*, or *posix_trace_open()*. The *posix_trace_create()* and
 3034 *posix_trace_create_withlog()* functions should be used by a controller process. The
 3035 *posix_trace_open()* should be used by an analyzer process.

3036 The tracing functions can serve different purposes. One purpose is debugging the possibly pre-
 3037 instrumented code, while another is post-mortem fault analysis. These two potential uses differ
 3038 in that the first requires pre-filtering capabilities to avoid overwhelming the trace stream and
 3039 permits focusing on expected information; while the second needs comprehensive trace
 3040 capabilities in order to be able to record all types of information.

3041 The events to be traced belong to two classes:

- 3042 1. User trace events (generated by the application instrumentation)
- 3043 2. System trace events (generated by the operating system)

3044 The trace interface defines several system trace event types associated with control of and
 3045 operation of the trace stream. This small set of system trace events includes the minimum
 3046 required to interpret correctly the trace event information present in the stream. Other desirable
 3047 system trace events for some particular application profile may be implemented and are
 3048 encouraged; for example, process and thread scheduling, signal occurrence, and so on.

3049 Each traced process shall have a mapping of the trace event names to trace event type identifiers
 3050 that have been defined for that process. Each active trace stream shall have a mapping that
 3051 incorporates all the trace event type identifiers predefined by the trace system plus all the
 3052 mappings of trace event names to trace event type identifiers of the processes that are being
 3053 traced into that trace stream. These mappings are defined from the instrumented application by

3054 calling the *posix_trace_eventid_open()* function and from the controller process by calling the
 3055 *posix_trace_trid_eventid_open()* function. For a pre-recorded trace stream, the list of trace event
 3056 types is obtained from the pre-recorded trace log.

3057 The *st_ctime* and *st_mtime* fields of a file associated with an active trace stream shall be marked
 3058 for update every time any of the tracing operations modifies that file.

3059 The *st_atime* field of a file associated with a trace stream shall be marked for update every time
 3060 any of the tracing operations causes data to be read from that file.

3061 Results are undefined if the application performs any operation on a file descriptor associated
 3062 with an active or pre-recorded trace stream until *posix_trace_shutdown()* or *posix_trace_close()* is
 3063 called for that trace stream.

3064 The main purpose of this option is to define a complete set of functions and concepts that allow
 3065 a portable application to be traced from birth to death, whatever its realtime constraints and
 3066 properties.

3067 2.11.1 Tracing Data Definitions

3068 2.11.1.1 Structures

3069 The **<trace.h>** header shall define the *posix_trace_status_info* and *posix_trace_event_info* structures
 3070 described below. Implementations may add extensions to these structures.

3071 **posix_trace_status_info** Structure

3072 To facilitate control of a trace stream, information about the current state of an active trace
 3073 stream can be obtained dynamically. This structure is returned by a call to the
 3074 *posix_trace_get_status()* function.

3075 The **posix_trace_status_info** structure defined in **<trace.h>** shall contain at least the following
 3076 members:

3077

3078

3079

3080

3081

3082

Member Type	Member Name	Description
int	<i>posix_stream_status</i>	The operating mode of the trace stream.
int	<i>posix_stream_full_status</i>	The full status of the trace stream.
int	<i>posix_stream_overrun_status</i>	Indicates whether trace events were lost in the trace stream.

3083 If the Trace Log option is supported in addition to the Trace option, the **posix_trace_status_info**
 3084 structure defined in **<trace.h>** shall contain at least the following additional members:

3085

3086

3087

3088

3089

3090

3091

3092

Member Type	Member Name	Description
int	<i>posix_stream_flush_status</i>	Indicates whether a flush is in progress.
int	<i>posix_stream_flush_error</i>	Indicates whether any error occurred during the last flush operation.
int	<i>posix_log_overrun_status</i>	Indicates whether trace events were lost in the trace log.
int	<i>posix_log_full_status</i>	The full status of the trace log.

3093 The *posix_stream_status* member indicates the operating mode of the trace stream and shall have
 3094 one of the following values defined by manifest constants in the **<trace.h>** header:

- 3095 POSIX_TRACE_RUNNING
 3096 Tracing is in progress; that is, the trace stream is accepting trace events.
- 3097 POSIX_TRACE_SUSPENDED
 3098 The trace stream is not accepting trace events. The tracing operation has not yet started or
 3099 has stopped, either following a *posix_trace_stop()* function call or because the trace resources
 3100 are exhausted.
- 3101 The *posix_stream_full_status* member indicates the full status of the trace stream, and it shall have
 3102 one of the following values defined by manifest constants in the `<trace.h>` header:
- 3103 POSIX_TRACE_FULL
 3104 The space in the trace stream for trace events is exhausted.
- 3105 POSIX_TRACE_NOT_FULL
 3106 There is still space available in the trace stream.
- 3107 The combination of the *posix_stream_status* and *posix_stream_full_status* members also indicates
 3108 the actual status of the stream. The status shall be interpreted as follows:
- 3109 POSIX_TRACE_RUNNING and POSIX_TRACE_NOT_FULL
 3110 This status combination indicates that tracing is in progress, and there is space available for
 3111 recording more trace events.
- 3112 POSIX_TRACE_RUNNING and POSIX_TRACE_FULL
 3113 This status combination indicates that tracing is in progress and that the trace stream is full
 3114 of trace events. This status combination cannot occur unless the *stream-full-policy* is set to
 3115 POSIX_TRACE_LOOP. The trace stream contains trace events recorded during a moving
 3116 time window of prior trace events, and some older trace events may have been overwritten
 3117 and thus lost.
- 3118 POSIX_TRACE_SUSPENDED and POSIX_TRACE_NOT_FULL
 3119 This status combination indicates that tracing has not yet been started, has been stopped by
 3120 the *posix_trace_stop()* function, or has been cleared by the *posix_trace_clear()* function.
- 3121 POSIX_TRACE_SUSPENDED and POSIX_TRACE_FULL
 3122 This status combination indicates that tracing has been stopped by the implementation
 3123 because the *stream-full-policy* attribute was POSIX_TRACE_UNTIL_FULL and trace
 3124 resources were exhausted, or that the trace stream was stopped by the function
 3125 *posix_trace_stop()* at a time when trace resources were exhausted.
- 3126 The *posix_stream_overrun_status* member indicates whether trace events were lost in the trace
 3127 stream, and shall have one of the following values defined by manifest constants in the
 3128 `<trace.h>` header:
- 3129 POSIX_TRACE_OVERRUN
 3130 At least one trace event was lost and thus was not recorded in the trace stream.
- 3131 POSIX_TRACE_NO_OVERRUN
 3132 No trace events were lost.
- 3133 When the corresponding trace stream is created, the *posix_stream_overrun_status* member shall be
 3134 set to POSIX_TRACE_NO_OVERRUN.
- 3135 Whenever an overrun occurs, *posix_stream_overrun_status* member shall be set to
 3136 POSIX_TRACE_OVERRUN.
- 3137 An overrun occurs when:

- 3138 • The policy is `POSIX_TRACE_LOOP` and a recorded trace event is overwritten.
- 3139 • The policy is `POSIX_TRACE_UNTIL_FULL` and the trace stream is full when a trace event is
- 3140 generated.
- 3141 • If the Trace Log option is supported, the policy is `POSIX_TRACE_FLUSH` and at least one
- 3142 trace event is lost while flushing the trace stream to the trace log.
- 3143 The *posix_stream_overrun_status* member is reset to zero after its value is read.
- 3144 If the Trace Log option is supported in addition to the Trace option, the *posix_stream_flush_status*,
- 3145 *posix_stream_flush_error*, *posix_log_overrun_status*, and *posix_log_full_status* members are defined
- 3146 as follows; otherwise, they are undefined.
- 3147 The *posix_stream_flush_status* member indicates whether a flush operation is being performed
- 3148 and shall have one of the following values defined by manifest constants in the header
- 3149 <**trace.h**>:
 - 3150 `POSIX_TRACE_FLUSHING`
 - 3151 The trace stream is currently being flushed to the trace log.
 - 3152 `POSIX_TRACE_NOT_FLUSHING`
 - 3153 No flush operation is in progress.
- 3154 The *posix_stream_flush_status* member shall be set to `POSIX_TRACE_FLUSHING` if a flush
- 3155 operation is in progress either due to a call to the *posix_trace_flush()* function (explicit or caused
- 3156 by a trace stream shutdown operation) or because the trace stream has become full with the
- 3157 *stream-full-policy* attribute set to `POSIX_TRACE_FLUSH`. The *posix_stream_flush_status* member
- 3158 shall be set to `POSIX_TRACE_NOT_FLUSHING` if no flush operation is in progress.
- 3159 The *posix_stream_flush_error* member shall be set to zero if no error occurred during flushing. If
- 3160 an error occurred during a previous flushing operation, the *posix_stream_flush_error* member
- 3161 shall be set to the value of the first error that occurred. If more than one error occurs while
- 3162 flushing, error values after the first shall be discarded. The *posix_stream_flush_error* member is
- 3163 reset to zero after its value is read.
- 3164 The *posix_log_overrun_status* member indicates whether trace events were lost in the trace log,
- 3165 and shall have one of the following values defined by manifest constants in the <**trace.h**>
- 3166 header:
 - 3167 `POSIX_TRACE_OVERRUN`
 - 3168 At least one trace event was lost.
 - 3169 `POSIX_TRACE_NO_OVERRUN`
 - 3170 No trace events were lost.
- 3171 When the corresponding trace stream is created, the *posix_log_overrun_status* member shall be set
- 3172 to `POSIX_TRACE_NO_OVERRUN`. Whenever an overrun occurs, this status shall be set to
- 3173 `POSIX_TRACE_OVERRUN`. The *posix_log_overrun_status* member is reset to zero after its value
- 3174 is read.
- 3175 The *posix_log_full_status* member indicates the full status of the trace log, and it shall have one of
- 3176 the following values defined by manifest constants in the <**trace.h**> header:
 - 3177 `POSIX_TRACE_FULL`
 - 3178 The space in the trace log is exhausted.
 - 3179 `POSIX_TRACE_NOT_FULL`
 - 3180 There is still space available in the trace log.

3181 The *posix_log_full_status* member is only meaningful if the *log-full-policy* attribute is either
3182 POSIX_TRACE_UNTIL_FULL or POSIX_TRACE_LOOP.

3183 For an active trace stream without log, that is created by the *posix_trace_create()* function, the
3184 *posix_log_overrun_status* member shall be set to POSIX_TRACE_NO_OVERRUN and the
3185 *posix_log_full_status* member shall be set to POSIX_TRACE_NOT_FULL.

3186 **posix_trace_event_info Structure**

3187 The trace event structure **posix_trace_event_info** contains the information for one recorded
3188 trace event. This structure is returned by the set of functions *posix_trace_getnext_event()*,
3189 *posix_trace_timedgetnext_event()*, and *posix_trace_trygetnext_event()*.

3190 The **posix_trace_event_info** structure defined in <trace.h> shall contain at least the following
3191 members:

3192

3193

Member Type	Member Name	Description
trace_event_id_t	<i>posix_event_id</i>	Trace event type identification.
pid_t	<i>posix_pid</i>	Process ID of the process that generated the trace event.
void*	<i>posix_prog_address</i>	Address at which the trace point was invoked.
int	<i>posix_truncation_status</i>	Status about the truncation of the data associated with this trace event.
struct timespec	<i>posix_timestamp</i>	Time at which the trace event was generated.

3196

3197

3198

3199

3200

3201 In addition, if the Trace option and the Threads option are both supported, the
3202 **posix_trace_event_info** structure defined in <trace.h> shall contain the following additional
3203 member:

3204

3205

Member Type	Member Name	Description
pthread_t	<i>posix_thread_id</i>	Thread ID of the thread that generated the trace event.

3206

3207

3208

3209 The *posix_event_id* member represents the identification of the trace event type and its value is
3210 not directly defined by the user. This identification is returned by a call to one of the following
3211 functions: *posix_trace_trid_eventid_open()*, *posix_trace_eventtypelist_getnext_id()*, or
3212 *posix_trace_eventid_open()*. The name of the trace event type can be obtained by calling
3213 *posix_trace_eventid_get_name()*.

3214 The *posix_pid* is the process identifier of the traced process which generated the trace event. If
3215 the *posix_event_id* member is one of the implementation-defined system trace events and that
3216 trace event is not associated with any process, the *posix_pid* member shall be set to zero.

3217 For a user trace event, the *posix_prog_address* member is the process mapped address of the point
3218 at which the associated call to the *posix_trace_event()* function was made. For a system trace
3219 event, if the trace event is caused by a system service explicitly called by the application, the
3220 *posix_prog_address* member shall be the address of the process at the point where the call to that
3221 system service was made.

3222 The *posix_truncation_status* member defines whether the data associated with a trace event has
3223 been truncated at the time the trace event was generated, or at the time the trace event was read
3224 from the trace stream, or (if the Trace Log option is supported) from the trace log (see the *event*
3225 argument from the *posix_trace_getnext_event()* function). The *posix_truncation_status* member

3226 shall have one of the following values defined by manifest constants in the `<trace.h>` header:

3227 `POSIX_TRACE_NOT_TRUNCATED`

3228 All the traced data is available.

3229 `POSIX_TRACE_TRUNCATED_RECORD`

3230 Data was truncated at the time the trace event was generated.

3231 `POSIX_TRACE_TRUNCATED_READ`

3232 Data was truncated at the time the trace event was read from a trace stream or a trace log
3233 because the reader's buffer was too small. This truncation status overrides the
3234 `POSIX_TRACE_TRUNCATED_RECORD` status.

3235 The *posix_timestamp* member shall be the time at which the trace event was generated. The clock
3236 used is implementation-defined, but the resolution of this clock can be retrieved by a call to the
3237 *posix_trace_attr_getclockres()* function.

3238 If the Threads option is supported in addition to the Trace option:

- 3239 • The *posix_thread_id* member is the identifier of the thread that generated the trace event. If
- 3240 the *posix_event_id* member is one of the implementation-defined system trace events and that
- 3241 trace event is not associated with any thread, the *posix_thread_id* member shall be set to zero.

3242 Otherwise, this member is undefined.

3243 2.11.1.2 Trace Stream Attributes

3244 Trace streams have attributes that compose the `posix_trace_attr_t` trace stream attributes object.
3245 This object shall contain at least the following attributes:

- 3246 • The *generation-version* attribute identifies the origin and version of the trace system.
- 3247 • The *trace-name* attribute is a character string defined by the trace controller, and that
3248 identifies the trace stream.
- 3249 • The *creation-time* attribute represents the time of the creation of the trace stream.
- 3250 • The *clock-resolution* attribute defines the clock resolution of the clock used to generate
3251 timestamps.
- 3252 • The *stream-min-size* attribute defines the minimum size in bytes of the trace stream strictly
3253 reserved for the trace events.
- 3254 • The *stream-full-policy* attribute defines the policy followed when the trace stream is full; its
3255 value is `POSIX_TRACE_LOOP`, `POSIX_TRACE_UNTIL_FULL`, or `POSIX_TRACE_FLUSH`.
- 3256 • The *max-data-size* attribute defines the maximum record size in bytes of a trace event.

3257 In addition, if the Trace option and the Trace Inherit option are both supported, the
3258 `posix_trace_attr_t` trace stream creation attributes object shall contain at least the following
3259 attributes:

- 3260 • The *inheritance* attribute specifies whether a newly created trace stream will inherit tracing in
3261 its parent's process trace stream. It is either `POSIX_TRACE_INHERITED` or
3262 `POSIX_TRACE_CLOSE_FOR_CHILD`.

3263 In addition, if the Trace option and the Trace Log option are both supported, the
3264 `posix_trace_attr_t` trace stream creation attributes object shall contain at least the following
3265 attribute:

- 3266 • If the file type corresponding to the trace log supports the `POSIX_TRACE_LOOP` or the
3267 `POSIX_TRACE_UNTIL_FULL` policies, the *log-max-size* attribute defines the maximum size

- 3268 in bytes of the trace log associated with an active trace stream. Other stream data—for
 3269 example, trace attribute values—shall not be included in this size.
- 3270 • The *log-full-policy* attribute defines the policy of a trace log associated with an active trace
 3271 stream to be POSIX_TRACE_LOOP, POSIX_TRACE_UNTIL_FULL, or
 3272 POSIX_TRACE_APPEND.

3273 2.11.2 Trace Event Type Definitions

3274 2.11.2.1 System Trace Event Type Definitions

3275 The following system trace event types, defined in the `<trace.h>` header, track the invocation of
 3276 the trace operations:

- 3277 • POSIX_TRACE_START shall be associated with a trace start operation.
- 3278 • POSIX_TRACE_STOP shall be associated with a trace stop operation.
- 3279 • if the Trace Event Filter option is supported, POSIX_TRACE_FILTER shall be associated with
 3280 a trace event type filter change operation.

3281 The following system trace event types, defined in the `<trace.h>` header, report operational trace
 3282 events:

- 3283 • POSIX_TRACE_OVERFLOW shall mark the beginning of a trace overflow condition.
- 3284 • POSIX_TRACE_RESUME shall mark the end of a trace overflow condition.
- 3285 • If the Trace Log option is supported, POSIX_TRACE_FLUSH_START shall mark the
 3286 beginning of a flush operation.
- 3287 • If the Trace Log option is supported, POSIX_TRACE_FLUSH_STOP shall mark the end of a
 3288 flush operation.
- 3289 • If an implementation-defined trace error condition is reported, it shall be marked
 3290 POSIX_TRACE_ERROR.

3291 The interpretation of a trace stream or a trace log by a trace analyzer process relies on the
 3292 information recorded for each trace event, and also on system trace events that indicate the
 3293 invocation of trace control operations and trace system operational trace events.

3294 The POSIX_TRACE_START and POSIX_TRACE_STOP trace events specify the time windows
 3295 during which the trace stream is running.

3296 The POSIX_TRACE_STOP trace event with an associated data that is equal to zero indicates
 3297 a call of the function *posix_trace_stop()*.

3298 The POSIX_TRACE_STOP trace event with an associated data that is different from zero
 3299 indicates an automatic stop of the trace stream (see *posix_trace_attr_getstreamfullpolicy()*
 3300 defined in the System Interfaces volume of IEEE Std. 1003.1-200x).

3301 The POSIX_TRACE_FILTER trace event indicates that a trace event type filter value changed
 3302 while the trace stream was running.

3303 The POSIX_TRACE_ERROR serves to inform the analyzer process that an implementation-
 3304 defined internal error of the trace system occurred.

3305 The POSIX_TRACE_OVERFLOW trace event shall be reported with a timestamp equal to the
 3306 timestamp of the first trace event overwritten. This is an indication that some generated trace
 3307 events have been lost.

3308 The POSIX_TRACE_RESUME trace event shall be reported with a timestamp equal to the
 3309 timestamp of the first valid trace event reported after the overflow condition ends and shall be
 3310 reported before this first valid trace event. This is an indication that the trace system is reliably
 3311 recording trace events after an overflow condition.

3312 Each of these trace event types is defined by a constant trace event name and a **trace_event_id_t**
 3313 constant; trace event data is associated with some of these trace events.

3314 If the Trace option is supported and the Trace Event Filter option and the Trace Log option are
 3315 not supported, the following predefined system trace events in Table 2-6 shall be defined:

3316 **Table 2-6** Trace Option: System Trace Events

Event Name	Constant	Associated Data
		Data Type
"posix_trace_error"	POSIX_TRACE_ERROR	error int
"posix_trace_start"	POSIX_TRACE_START	None.
"posix_trace_stop"	POSIX_TRACE_STOP	auto int
"posix_trace_overflow"	POSIX_TRACE_OVERFLOW	None.
"posix_trace_resume"	POSIX_TRACE_RESUME	None.

3326 If the Trace option and the Trace Event Filter option are both supported, and if the Trace Log
 3327 option is not supported, the following predefined system trace events in Table 2-7 shall be
 3328 defined:

3329 **Table 2-7** Trace and Trace Event Filter Options: System Trace Events

Event Name	Constant	Associated Data
		Data Type
"posix_trace_error"	POSIX_TRACE_ERROR	error int
"posix_trace_start"	POSIX_TRACE_START	event_filter trace_event_set_t
"posix_trace_stop"	POSIX_TRACE_STOP	auto int
"posix_trace_filter"	POSIX_TRACE_FILTER	old_event_filter new_event_filter trace_event_set_t
"posix_trace_overflow"	POSIX_TRACE_OVERFLOW	None.
"posix_trace_resume"	POSIX_TRACE_RESUME	None.

3343 If the Trace option and the Trace Log option are both supported, and if the Trace Event Filter
 3344 option is not supported, the following predefined system trace events in Table 2-8 (on page 581)
 3345 shall be defined:

3346

Table 2-8 Trace and Trace Log Options: System Trace Events

3347

3348

3349

3350

3351

3352

3353

3354

3355

3356

3357

3358

Event Name	Constant	Associated Data
		Data Type
"posix_trace_error"	POSIX_TRACE_ERROR	error - int
"posix_trace_start"	POSIX_TRACE_START	None.
"posix_trace_stop"	POSIX_TRACE_STOP	auto
		int
"posix_trace_overflow"	POSIX_TRACE_OVERFLOW	None.
"posix_trace_resume"	POSIX_TRACE_RESUME	None.
"posix_trace_flush_start"	POSIX_TRACE_FLUSH_START	None.
"posix_trace_flush_stop"	POSIX_TRACE_FLUSH_STOP	None.

3359

3360

If the Trace option, the Trace Event Filter option, and the Trace Log option are all supported, the following predefined system trace events in Table 2-9 shall be defined:

3361

Table 2-9 Trace, Trace Log, and Trace Event Filter Options: System Trace Events

3362

3363

3364

3365

3366

3367

3368

3369

3370

3371

3372

3373

3374

3375

3376

Event Name	Constant	Associated Data
		Data Type
"posix_trace_error"	POSIX_TRACE_ERROR	error
		int
"posix_trace_start"	POSIX_TRACE_START	event_filter
		trace_event_set_t
"posix_trace_stop"	POSIX_TRACE_STOP	auto
		int
"posix_trace_filter"	POSIX_TRACE_FILTER	old_event_filter
		new_event_filter
		trace_event_set_t
"posix_trace_overflow"	POSIX_TRACE_OVERFLOW	None.
"posix_trace_resume"	POSIX_TRACE_RESUME	None.
"posix_trace_flush_start"	POSIX_TRACE_FLUSH_START	None.
"posix_trace_flush_stop"	POSIX_TRACE_FLUSH_STOP	None.

3377 2.11.2.2 User Trace Event Type Definitions

3378

3379

3380

3381

3382

The user trace event POSIX_TRACE_UNNAMED_USEREVENT shall be defined in the `<trace.h>` header. If the limit of per-process user trace event names represented by {TRACE_USER_EVENT_MAX} has already been reached, this predefined user event shall be returned when the application tries to register more events than allowed. The data associated with this trace event is application-defined.

3383

The following predefined user trace event in Table 2-10 (on page 582) shall be defined:

3384

Table 2-10 Trace Option: User Trace Event

3385

3386

Event Name	Constant
"posix_trace_unnamed_userevent"	POSIX_TRACE_UNNAMED_USEREVENT

3387 **2.11.3 Trace Functions**

3388

3389

3390

3391

3392

3393

The trace interface is built and structured to improve portability through use of trace data of opaque type. The object-oriented approach for the manipulation of trace attributes and trace event type identifiers requires definition of many constructor and selector functions which operate on these opaque types. Also, the trace interface must support several different tracing roles. To facilitate reading the trace interface, the trace functions are grouped into small functional sets supporting the three different roles:

3394

3395

- A trace controller process requires functions to set up and customize all the resources needed to run a trace stream, including:

3396

- Attribute initialization and destruction (*posix_trace_attr_init()*)

3397

- Identification information manipulation (*posix_trace_attr_getgenversion()*)

3398

- Trace system behavior modification (*posix_trace_attr_getinherited()*)

3399

- Trace stream and trace log size set (*posix_trace_attr_getmaxusersize()*)

3400

- Trace stream creation, flush, and shutdown (*posix_trace_create()*)

3401

- Trace stream and trace log clear (*posix_trace_clear()*)

3402

- Trace event type identifier manipulation (*posix_trace_trid_eventid_open()*)

3403

- Trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id()*)

3404

- Trace event type set manipulation (*posix_trace_eventset_empty()*)

3405

- Trace event type filter set (*posix_trace_set_filter()*)

3406

- Trace stream start and stop (*posix_trace_start()*)

3407

- Trace stream information and status read (*posix_trace_get_attr()*)

3408

- A traced process requires functions to instrument trace points:

3409

- Trace event type identifiers definition and trace points insertion (*posix_trace_event()*)

3410

3411

- A trace analyzer process requires functions to retrieve information from a trace stream and trace log:

3412

- Identification information read (*posix_trace_attr_getgenversion()*)

3413

- Trace system behavior information read (*posix_trace_attr_getinherited()*)

3414

- Trace stream and trace log size get (*posix_trace_attr_getmaxusersize()*)

3415

- Trace event type identifier manipulation (*posix_trace_trid_eventid_open()*)

3416

- Trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id()*)

3417

- Trace log open, rewind, and close (*posix_trace_open()*)

3418

- Trace stream information and status read (*posix_trace_get_attr()*)

3419

- trace event read (*posix_trace_getnext_event()*)

3420 **2.12 Data Types**

3421 All of the data types used by various functions are defined by the implementation. The
 3422 following table describes some of these types. Other types referenced in the description of a
 3423 function, not mentioned here, can be found in the appropriate header for that function.

3424

3425

Defined Type	Description
3426 cc_t	Type used for terminal special characters.
3427 clock_t	Arithmetic type used for processor times, as defined in the ISO C standard.
3428	
3429 clockid_t	Used for clock ID type in some timer functions.
3430 dev_t	Arithmetic type used for device numbers.
3431 DIR	Type representing a directory stream.
3432 div_t	Structure type returned by the <i>div()</i> function.
3433 FILE	Structure containing information about a file.
3434 glob_t	Structure type used in path name pattern matching.
3435 fpos_t	Type containing all information needed to specify uniquely every position within a file.
3436	
3437 gid_t	Arithmetic type used for group IDs.
3438 iconv_t	Type used for conversion descriptors.
3439 id_t	Arithmetic type used as a general identifier; can be used to contain at least the largest of a pid_t , uid_t , or gid_t .
3440	
3441 ino_t	Arithmetic type used for file serial numbers.
3442 key_t	Arithmetic type used for XSI interprocess communication.
3443 ldiv_t	Structure type returned by the <i>ldiv()</i> function.
3444 mode_t	Arithmetic type used for file attributes.
3445 mqd_t	Used for message queue descriptors.
3446 nfds_t	Integer type used for the number of file descriptors.
3447 nlink_t	Arithmetic type used for link counts.
3448 off_t	Signed arithmetic type used for file sizes.
3449 pid_t	Signed arithmetic type used for process and process group IDs.
3450 pthread_attr_t	Used to identify a thread attribute object.
3451 pthread_cond_t	Used for condition variables.
3452 pthread_condattr_t	Used to identify a condition attribute object.
3453 pthread_key_t	Used for thread-specific data keys.
3454 pthread_mutex_t	Used for mutexes.
3455 pthread_mutexattr_t	Used to identify a mutex attribute object.
3456 pthread_once_t	Used for dynamic package initialization.
3457 pthread_rwlock_t	Used for read-write locks.
3458 pthread_rwlockattr_t	Used for read-write lock attributes.
3459 pthread_t	Used to identify a thread.
3460 ptrdiff_t	Signed integer type of the result of subtracting two pointers.
3461 regex_t	Structure type used in regular expression matching.
3462 regmatch_t	Structure type used in regular expression matching.
3463 rlim_t	Unsigned arithmetic type used for limit values, to which objects of type int and off_t can be cast without loss of value.
3464	
3465 sem_t	Type used in performing semaphore operations.
3466 sig_atomic_t	Integer type of an object that can be accessed as an atomic

3467
 3468
 3469
 3470
 3471
 3472
 3473
 3474
 3475
 3476
 3477
 3478
 3479
 3480
 3481
 3482
 3483
 3484
 3485
 3486
 3487
 3488
 3489
 3490

Defined Type	Description
sigset_t	entity, even in the presence of asynchronous interrupts. Integer or structure type of an object used to represent sets of signals.
size_t	Unsigned integer type used for size of objects.
speed_t	Type used for terminal baud rates.
ssize_t	Arithmetic type used for a count of bytes or an error indication.
suseconds_t	Signed arithmetic type used for time in microseconds.
tcflag_t	Type used for terminal modes.
time_t	Arithmetic type used for time in seconds, as defined in the ISO C standard.
timer_t	Used for timer ID returned by the <i>timer_create()</i> function.
uid_t	Arithmetic type used for user IDs.
useconds_t	Integer type used for time in microseconds.
va_list	Type used for traversing variable argument lists.
wchar_t	Integer type whose range of values can represent distinct codes for all members of the largest extended character set specified by the supported locales.
wctype_t	Scalar type which represents a character class descriptor.
wint_t	Integer type capable of storing any valid value of wchar_t or WEOF.
wordexp_t	Structure type used in word expansion.

System Interfaces

3491

3492 This chapter describes the functions, macros, and external variables to support applications
3493 portability at the C-language source level.

3494 **NAME**3495 **FD_CLR** — macros for synchronous I/O multiplexing3496 **SYNOPSIS**3497 `#include <sys/time.h>`3498 `FD_CLR(int fd, fd_set *fdset);`3499 `FD_ISSET(int fd, fd_set *fdset);`3500 `FD_SET(int fd, fd_set *fdset);`3501 `FD_ZERO(fd_set *fdset);`3502 **DESCRIPTION**3503 Refer to *select()*.

3504 **NAME**
3505 `_Exit, _exit` — terminate a process

3506 **SYNOPSIS**
3507 `#include <unistd.h>`

3508 `void _Exit(int status);`
3509 `void _exit(int status);`

3510 **DESCRIPTION**
3511 Refer to *exit()*.

3512 **NAME**

3513 _longjmp, _setjmp — non-local goto

3514 **SYNOPSIS**

3515 xSI #include <setjmp.h>

3516 void _longjmp(jmp_buf env, int val);

3517 int _setjmp(jmp_buf env);

3518

3519 **DESCRIPTION**

3520 The *_longjmp()* and *_setjmp()* functions are identical to *longjmp()* and *setjmp()*, respectively, with
3521 the additional restriction that *_longjmp()* and *_setjmp()* do not manipulate the signal mask.

3522 If *_longjmp()* is called even though *env* was never initialized by a call to *_setjmp()*, or when the
3523 last such call was in a function that has since returned, the results are undefined.

3524 **RETURN VALUE**

3525 Refer to *longjmp()* and *setjmp()*.

3526 **ERRORS**

3527 No errors are defined.

3528 **EXAMPLES**

3529 None.

3530 **APPLICATION USAGE**

3531 If *_longjmp()* is executed and the environment in which *_setjmp()* was executed no longer exists,
3532 errors can occur. The conditions under which the environment of the *_setjmp()* no longer exists
3533 include exiting the function that contains the *_setjmp()* call, and exiting an inner block with
3534 temporary storage. This condition might not be detectable, in which case the *_longjmp()* occurs
3535 and, if the environment no longer exists, the contents of the temporary storage of an inner block
3536 are unpredictable. This condition might also cause unexpected process termination. If the
3537 function has returned, the results are undefined.

3538 Passing *longjmp()* a pointer to a buffer not created by *setjmp()*, passing *_longjmp()* a pointer to a
3539 buffer not created by *_setjmp()*, passing *siglongjmp()* a pointer to a buffer not created by
3540 *sigsetjmp()*, or passing any of these three functions a buffer that has been modified by the user
3541 can cause all the problems listed above, and more.

3542 The *_longjmp()* and *_setjmp()* functions are included to support programs written to historical
3543 system interfaces. New applications should use *siglongjmp()* and *sigsetjmp()* respectively.

3544 **RATIONALE**

3545 None.

3546 **FUTURE DIRECTIONS**

3547 The *_longjmp()* and *_setjmp()* functions may be marked LEGACY in a future version.

3548 **SEE ALSO**

3549 *longjmp()*, *setjmp()*, *siglongjmp()*, *sigsetjmp()*, the Base Definitions volume of
3550 IEEE Std. 1003.1-200x, <setjmp.h>

3551 **CHANGE HISTORY**

3552 First released in Issue 4, Version 2.

3553 **Issue 5**

3554 Moved from X/OPEN UNIX extension to BASE.

3555 **NAME**

3556 _setjmp — set jump point for a non-local goto

3557 **SYNOPSIS**

3558 xSI #include <setjmp.h>

3559 int _setjmp(jmp_buf env);

3560

3561 **DESCRIPTION**

3562 Refer to *_longjmp()*.

3563 **NAME**

3564 _toupper — transliterate uppercase characters to lowercase

3565 **SYNOPSIS**

3566 XSI #include <ctype.h>

3567 int _tolower(int c);

3568

3569 **DESCRIPTION**3570 The *_tolower()* macro shall be equivalent to *tolower(c)* except that the application shall ensure
3571 that the argument *c* is an uppercase letter.3572 **RETURN VALUE**3573 Upon successful completion, *_tolower()* shall return the lowercase letter corresponding to the
3574 argument passed.3575 **ERRORS**

3576 No errors are defined.

3577 **EXAMPLES**

3578 None.

3579 **APPLICATION USAGE**

3580 None.

3581 **RATIONALE**

3582 None.

3583 **FUTURE DIRECTIONS**

3584 None.

3585 **SEE ALSO**3586 *tolower()*, *isupper()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>, the Base
3587 Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale3588 **CHANGE HISTORY**

3589 First released in Issue 1. Derived from Issue 1 of the SVID.

3590 **Issue 4**

3591 The RETURN VALUE section is expanded.

3592 **Issue 6**

3593 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

3594 **NAME**

3595 _toupper — transliterate lowercase characters to uppercase

3596 **SYNOPSIS**

3597 xSI #include <ctype.h>

3598 int _toupper(int c);

3599

3600 **DESCRIPTION**

3601 The *_toupper()* macro shall be equivalent to *toupper()* except that the application shall ensure
3602 that the argument *c* is a lowercase letter.

3603 **RETURN VALUE**

3604 Upon successful completion, *_toupper()* shall return the uppercase letter corresponding to the
3605 argument passed.

3606 **ERRORS**

3607 No errors are defined.

3608 **EXAMPLES**

3609 None.

3610 **APPLICATION USAGE**

3611 None.

3612 **RATIONALE**

3613 None.

3614 **FUTURE DIRECTIONS**

3615 None.

3616 **SEE ALSO**

3617 *islower()*, *toupper()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>, the Base
3618 Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

3619 **CHANGE HISTORY**

3620 First released in Issue 1. Derived from Issue 1 of the SVID.

3621 **Issue 4**

3622 The RETURN VALUE section is expanded.

3623 **Issue 6**

3624 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

3625 **NAME**

3626 a64l, l64a — convert between a 32-bit integer and a radix-64 ASCII string

3627 **SYNOPSIS**

```
3628 xSI      #include <stdlib.h>
3629
3629         long a64l(const char *s);
3630         char *l64a(long value);
3631
```

3632 **DESCRIPTION**

3633 These functions are used to maintain numbers stored in radix-64 ASCII characters. This is a
 3634 notation by which 32-bit integers can be represented by up to six characters; each character
 3635 represents a digit in radix-64 notation. If the type **long** contains more than 32 bits, only the low-
 3636 order 32 bits shall be used for these operations.

3637 The characters used to represent digits are '.' (dot) for 0, '/' for 1, '0' through '9' for 2-11,
 3638 'A' through 'Z' for 12-37, and 'a' through 'z' for 38-63.

3639 The *a64l()* function shall take a pointer to a radix-64 representation, in which the first digit is the
 3640 least significant, and return a corresponding **long** value. If the string pointed to by *s* contains
 3641 more than six characters, *a64l()* shall use the first six. If the first six characters of the string
 3642 contain a null terminator, *a64l()* shall use only characters preceding the null terminator. The
 3643 *a64l()* function scans the character string from left to right with the least significant digit on the
 3644 left, decoding each character as a 6-bit radix-64 number. If the type **long** contains more than 32
 3645 bits, the resulting value is sign-extended. The behavior of *a64l()* is unspecified if *s* is a null
 3646 pointer or the string pointed to by *s* was not generated by a previous call to *l64a()*.

3647 The *l64a()* function shall take a **long** argument and return a pointer to the corresponding radix-
 3648 64 representation. The behavior of *l64a()* is unspecified if *value* is negative.

3649 The value returned by *l64a()* may be a pointer into a static buffer. Subsequent calls to *l64a()* may
 3650 overwrite the buffer.

3651 The *l64a()* function need not be reentrant. A function that is not required to be reentrant is not
 3652 required to be thread-safe.

3653 **RETURN VALUE**

3654 Upon successful completion, *a64l()* shall return the **long** value resulting from conversion of the
 3655 input string. If a string pointed to by *s* is an empty string, *a64l()* shall return 0L.

3656 The *l64a()* function shall return a pointer to the radix-64 representation. If *value* is 0L, *l64a()* shall
 3657 return a pointer to an empty string.

3658 **ERRORS**

3659 No errors are defined.

3660 **EXAMPLES**

3661 None.

3662 **APPLICATION USAGE**

3663 If the type **long** contains more than 32 bits, the result of *a64l(l64a(x))* is *x* in the low-order 32 bits.

3664 **RATIONALE**

3665 This is not the same encoding as used by either encoding variant of the *uuencode* utility.

3666 **FUTURE DIRECTIONS**

3667 None.

3668 **SEE ALSO**3669 *strtoul()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<stdlib.h>`, the Shell and Utilities
3670 volume of IEEE Std. 1003.1-200x, *uuencode*3671 **CHANGE HISTORY**

3672 First released in Issue 4, Version 2.

3673 **Issue 5**

3674 Moved from X/OPEN UNIX extension to BASE.

3675 Normative text previously in the APPLICATION USAGE section moved to the DESCRIPTION.

3676 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

3677 **NAME**

3678 abort — generate an abnormal process abort

3679 **SYNOPSIS**

3680 #include <stdlib.h>

3681 void abort(void);

3682 **DESCRIPTION**

3683 CX The functionality described on this reference page is aligned with the ISO C standard. Any
3684 conflict between the requirements described here and the ISO C standard is unintentional. This
3685 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

3686 The *abort()* function shall cause abnormal process termination to occur, unless the signal
3687 SIGABRT is being caught and the signal handler does not return.

3688 CX The abnormal termination processing shall include at least the effect of *fclose()* on all open
3689 streams and the default actions defined for SIGABRT.

3690 XSI On XSI-conformant systems, in addition the abnormal termination processing shall include the
3691 effect of *fclose()* on message catalog descriptors.

3692 The SIGABRT signal shall be sent to the calling process as if by means of *raise()* with the
3693 argument SIGABRT.

3694 CX The status made available to *wait()* or *waitpid()* by *abort()* shall be that of a process terminated
3695 by the SIGABRT signal. The *abort()* function shall override blocking or ignoring the SIGABRT
3696 signal.

3697 **RETURN VALUE**3698 The *abort()* function shall not return.3699 **ERRORS**

3700 No errors are defined.

3701 **EXAMPLES**

3702 None.

3703 **APPLICATION USAGE**

3704 Catching the signal is intended to provide the application writer with a portable means to abort
3705 processing, free from possible interference from any implementation-defined library functions. If
3706 SIGABRT is neither caught nor ignored, then the actions associated with SIGABRT defined in
3707 Section 2.4.1 (on page 528) will be taken.

3708 **RATIONALE**

3709 None.

3710 **FUTURE DIRECTIONS**

3711 None.

3712 **SEE ALSO**

3713 *exit()*, *kill()*, *raise()*, *signal()*, *wait()*, *waitpid()*, the Base Definitions volume of
3714 IEEE Std. 1003.1-200x, <stdlib.h>

3715 **CHANGE HISTORY**

3716 First released in Issue 1. Derived from Issue 1 of the SVID.

3717 Issue 4

3718 The following changes are incorporated in this issue for alignment with the ISO C standard and
3719 the ISO POSIX-1 standard:

- 3720 • The argument list is explicitly defined as **void**.
- 3721 • The DESCRIPTION is revised to identify the correct order in which operations occur. It also
3722 identifies:
 - 3723 — How the calling process is signaled
 - 3724 — How status information is made available to the host environment
 - 3725 — That *abort()* overrides blocking or ignoring of the SIGABRT signal
- 3726 • The APPLICATION USAGE section is replaced.

3727 Issue 6

3728 Extensions beyond the ISO C standard are now marked.

3729 **NAME**

3730 abs — return an integer absolute value

3731 **SYNOPSIS**

3732 #include <stdlib.h>

3733 int abs(int *i*);

3734 **DESCRIPTION**

3735 cx The functionality described on this reference page is aligned with the ISO C standard. Any
3736 conflict between the requirements described here and the ISO C standard is unintentional. This
3737 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

3738 The *abs()* function shall compute the absolute value of its integer operand, *i*. If the result cannot
3739 be represented, the behavior is undefined.

3740 **RETURN VALUE**

3741 The *abs()* function shall return the absolute value of its integer operand.

3742 **ERRORS**

3743 No errors are defined.

3744 **EXAMPLES**

3745 None.

3746 **APPLICATION USAGE**

3747 In two's-complement representation, the absolute value of the negative integer with largest
3748 magnitude {INT_MIN} might not be representable.

3749 **RATIONALE**

3750 None.

3751 **FUTURE DIRECTIONS**

3752 None.

3753 **SEE ALSO**

3754 *fabs()*, *labs()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>

3755 **CHANGE HISTORY**

3756 First released in Issue 1. Derived from Issue 1 of the SVID.

3757 **Issue 4**

3758 In the APPLICATION USAGE section, the phrase “{INT_MIN} is undefined” is replaced with
3759 “{INT_MIN} might not be representable”.

3760 **Issue 6**

3761 Extensions beyond the ISO C standard are now marked.

3762 NAME

3763 accept — accept a new connection on a socket

3764 SYNOPSIS

3765 #include <sys/socket.h>

```
3766 int accept(int socket, struct sockaddr *restrict address,
3767           socklen_t *restrict address_len);
```

3768 DESCRIPTION

3769 The *accept()* function shall extract the first connection on the queue of pending connections,
 3770 create a new socket with the same socket type protocol and address family as the specified
 3771 socket, and allocate a new file descriptor for that socket.

3772 The *accept()* function takes the following arguments:

3773 *socket* Specifies a socket that was created with *socket()*, has been bound to an address
 3774 with *bind()*, and has issued a successful call to *listen()*.

3775 *address* Either a null pointer, or a pointer to a **sockaddr** structure where the address of
 3776 the connecting socket shall be returned.

3777 *address_len* Points to a **socklen_t** structure which on input specifies the length of the
 3778 supplied **sockaddr** structure, and on output specifies the length of the stored
 3779 address.

3780 If *address* is not a null pointer, the address of the peer for the accepted connection shall be stored
 3781 in the **sockaddr** structure pointed to by *address*, and the length of this address shall be stored in
 3782 the object pointed to by *address_len*.

3783 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
 3784 the stored address shall be truncated.

3785 If the protocol permits connections by unbound clients, and the peer is not bound, then the value
 3786 stored in the object pointed to by *address* is unspecified.

3787 If the listen queue is empty of connection requests and O_NONBLOCK is not set on the file
 3788 descriptor for the socket, *accept()* shall block until a connection is present. If the *listen()* queue is
 3789 empty of connection requests and O_NONBLOCK is set on the file descriptor for the socket,
 3790 *accept()* shall fail and set *errno* to [EAGAIN] or [EWOULDBLOCK].

3791 The accepted socket cannot itself accept more connections. The original socket remains open and
 3792 can accept more connections.

3793 RETURN VALUE

3794 Upon successful completion, *accept()* shall return the non-negative file descriptor of the accepted
 3795 socket. Otherwise, -1 shall be returned and *errno* set to indicate the error.

3796 ERRORS

3797 The *accept()* function shall fail if:

3798 [EAGAIN] or [EWOULDBLOCK]

3799 O_NONBLOCK is set for the socket file descriptor and no connections are
 3800 present to be accepted.

3801 [EBADF] The *socket* argument is not a valid file descriptor.

3802 [ECONNABORTED]

3803 A connection has been aborted.

3804	[EINTR]	The <i>accept()</i> function was interrupted by a signal that was caught before a valid connection arrived.
3805		
3806	[EINVAL]	The <i>socket</i> is not accepting connections.
3807	[EMFILE]	{OPEN_MAX} file descriptors are currently open in the calling process.
3808	[ENFILE]	The maximum number of file descriptors in the system are already open.
3809	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
3810	[EOPNOTSUPP]	The <i>socket</i> type of the specified socket does not support accepting connections.
3811		
3812		The <i>accept()</i> function may fail if:
3813	[ENOBUFS]	No buffer space is available.
3814	[ENOMEM]	There was insufficient memory available to complete the operation.
3815	XSR [EPROTO]	A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.
3816		
3817	EXAMPLES	
3818		None.
3819	APPLICATION USAGE	
3820		When a connection is available, <i>select()</i> indicates that the file descriptor for the socket is ready for reading.
3821		
3822	RATIONALE	
3823		None.
3824	FUTURE DIRECTIONS	
3825		None.
3826	SEE ALSO	
3827		<i>bind()</i> , <i>connect()</i> , <i>listen()</i> , <i>socket()</i> , the Base Definitions volume of IEEE Std. 1003.1-200x,
3828		< sys/socket.h >
3829	CHANGE HISTORY	
3830		First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
3831		The restrict keyword is added to the <i>accept()</i> prototype for alignment with the
3832		ISO/IEC 9899:1999 standard.

3833 **NAME**

3834 access — determine accessibility of a file

3835 **SYNOPSIS**

3836 #include <unistd.h>

3837 int access(const char *path, int amode);

3838 **DESCRIPTION**

3839 The `access()` function shall check the file named by the `path` name pointed to by the `path`
3840 argument for accessibility according to the bit pattern contained in `amode`, using the real user ID
3841 in place of the effective user ID and the real group ID in place of the effective group ID.

3842 The value of `amode` is either the bitwise-inclusive OR of the access permissions to be checked
3843 (R_OK, W_OK, X_OK) or the existence test (F_OK).

3844 If any access permissions are checked, each shall be checked individually, as described in the
3845 Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 3, Definitions. If the process has
3846 appropriate privileges, an implementation may indicate success for X_OK even if none of the
3847 execute file permission bits are set.

3848 **RETURN VALUE**

3849 If the requested access is permitted, `access()` succeeds and shall return 0; otherwise, -1 shall be
3850 returned and `errno` shall be set to indicate the error.

3851 **ERRORS**3852 The `access()` function shall fail if:

3853 [EACCES] Permission bits of the file mode do not permit the requested access, or search
3854 permission is denied on a component of the path prefix.

3855 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`
3856 argument.

3857 [ENAMETOOLONG]
3858 The length of the `path` argument exceeds {PATH_MAX} or a path name
3859 component is longer than {NAME_MAX}.

3860 [ENOENT] A component of `path` does not name an existing file or `path` is an empty string.

3861 [ENOTDIR] A component of the path prefix is not a directory.

3862 [EROFS] Write access is requested for a file on a read-only file system.

3863 The `access()` function may fail if:

3864 [EINVAL] The value of the `amode` argument is invalid.

3865 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
3866 resolution of the `path` argument.

3867 [ENAMETOOLONG]
3868 As a result of encountering a symbolic link in resolution of the `path` argument,
3869 the length of the substituted path name string exceeded {PATH_MAX}.

3870 [ETXTBSY] Write access is requested for a pure procedure (shared text) file that is being
3871 executed.

3872 **EXAMPLES**3873 **Testing for the Existence of a File**

3874 The following example tests whether a file named **myfile** exists in the **/tmp** directory.

```
3875 #include <unistd.h>
3876 ...
3877 int result;
3878 const char *filename = "/tmp/myfile";
3879 result = access (filename, F_OK);
```

3880 **APPLICATION USAGE**

3881 Additional values of *amode* other than the set defined in the description may be valid; for
3882 example, if a system has extended access controls.

3883 **RATIONALE**

3884 In early proposals, some inadequacies in the *access()* function led to the creation of an *eaccess()*
3885 function because:

- 3886 1. Historical implementations of *access()* do not test file access correctly when the process'
3887 real user ID is superuser. In particular, they always return zero when testing execute
3888 permissions without regard to whether the file is executable.
- 3889 2. The superuser has complete access to all files on a system. As a consequence, programs
3890 started by the superuser and switched to the effective user ID with lesser privileges cannot
3891 use *access()* to test their file access permissions.

3892 However, the historical model of *eaccess()* does not resolve problem (1), so this volume of
3893 IEEE Std. 1003.1-200x now allows *access()* to behave in the desired way because several
3894 implementations have corrected the problem. It was also argued that problem (2) is more easily
3895 solved by using *open()*, *chdir()*, or one of the *exec* functions as appropriate and responding to the
3896 error, rather than creating a new function that would not be as reliable. Therefore, *eaccess()* is not
3897 included in this volume of IEEE Std. 1003.1-200x.

3898 The sentence concerning appropriate privileges and execute permission bits reflects the two
3899 possibilities implemented by historical implementations when checking superuser access for
3900 *X_OK*.

3901 **FUTURE DIRECTIONS**

3902 None.

3903 **SEE ALSO**

3904 *chmod()*, *stat()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<unistd.h>**

3905 **CHANGE HISTORY**

3906 First released in Issue 1. Derived from Issue 1 of the SVID.

3907 **Issue 4**

3908 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 3909 • The type of argument *path* is changed from **char*** to **const char***.

3910 The following change is incorporated for alignment with the FIPS requirements:

- 3911 • In the **ERRORS** section, the condition whereby [ENAMETOOLONG] is returned if a path
3912 name component is larger than {NAME_MAX} is now defined as mandatory and marked as
3913 an extension.

3914 **Issue 4, Version 2**

3915 The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- 3916 • It states that [ELOOP] is returned if too many symbolic links are encountered during path
- 3917 name resolution.
- 3918 • A second [ENAMETOOLONG] condition is defined that may report excessive length of an
- 3919 intermediate result of path name resolution of a symbolic link.

3920 **Issue 6**

3921 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 3922 • The [ENAMETOOLONG] error is restored as an error dependent on _POSIX_NO_TRUNC.
- 3923 This is since behavior may vary from one file system to another.

3924 The following new requirements on POSIX implementations derive from alignment with the
3925 Single UNIX Specification:

- 3926 • The [ELOOP] mandatory error condition is added.
- 3927 • A second [ENAMETOOLONG] is added as an optional error condition.
- 3928 • The [ETXTBSY] optional error condition is added.

3929 The following changes were made to align with the IEEE P1003.1a draft standard:

- 3930 • The [ELOOP] optional error condition is added.

3931 **NAME**

3932 acos, acosf, acosl — arc cosine function

3933 **SYNOPSIS**

3934 #include <math.h>

3935 double acos(double x);

3936 float acosf(float x);

3937 long double acosl(long double x);

3938 **DESCRIPTION**

3939 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 3940 conflict between the requirements described here and the ISO C standard is unintentional. This
 3941 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

3942 The *acos* family of functions shall compute the principal value of the arc cosine of *x*. The value
 3943 of *x* should be in the range $[-1,1]$.

3944 An application wishing to check for error situations should set *errno* to 0 before calling *acos()*. If
 3945 *errno* is non-zero on return, or the value NaN is returned, an error has occurred.

3946 **RETURN VALUE**

3947 Upon successful completion, the *acos* family of functions shall return the arc cosine of *x*, in the
 3948 **XSI** range $[0,\pi]$ radians. If the value of *x* is not in the range $[-1,1]$, and is not $\pm\text{Inf}$ or NaN, either 0.0 or
 3949 NaN shall be returned and *errno* shall be set to [EDOM].

3950 **XSI** If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM]. If *x* is $\pm\text{Inf}$, either 0.0 shall be
 3951 returned and *errno* shall be set to [EDOM], or NaN shall be returned and *errno* may be set to
 3952 [EDOM].

3953 **ERRORS**3954 The *acos* family of functions shall fail if:

3955 **XSI** [EDOM] The value *x* is not $\pm\text{Inf}$ or NaN and is not in the range $[-1,1]$.

3956 The *acos* family of functions may fail if:

3957 **XSI** [EDOM] The value *x* is $\pm\text{Inf}$ or NaN.

3958 **XSI** No other errors shall occur.3959 **EXAMPLES**

3960 None.

3961 **APPLICATION USAGE**

3962 None.

3963 **RATIONALE**

3964 None.

3965 **FUTURE DIRECTIONS**

3966 None.

3967 **SEE ALSO**3968 *cos()*, *isnan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>3969 **CHANGE HISTORY**

3970 First released in Issue 1. Derived from Issue 1 of the SVID.

3971 **Issue 4**

3972 Removed references to *matherr()*.

3973 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the
3974 ISO C standard and to rationalize error handling in the mathematics functions.

3975 The return value specified for [EDOM] is marked as an extension.

3976 **Issue 5**

3977 The DESCRIPTION is updated to indicate how an application should check for an error. This
3978 text was previously published in the APPLICATION USAGE section.

3979 **Issue 6**

3980 The *acosf()* and *acosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

3981 **NAME**3982 `acosh`, `acoshf`, `acoshl`, `asinh`, `asinhf`, `asinhf`, `atanh`, `atanhf`, `atanhl` — inverse hyperbolic functions3983 **SYNOPSIS**3984 `#include <math.h>`

```

3985     double acosh(double x);
3986     float  acoshf(float x);
3987     long double acoshl(long double x);
3988     double asinh(double x);
3989     float  asinhf(float x);
3990     long double asinhf(long double x);
3991     double atanh(double x);
3992     float  atanhf(float x);
3993     long double atanhf(long double x);

```

3994 **DESCRIPTION**

3995 The `acosh()`, `asinh()`, and `atanh()` functions shall compute the inverse hyperbolic cosine, sine, and
 3996 tangent of their argument, respectively.

3997 **RETURN VALUE**

3998 The `acosh()`, `asinh()`, and `atanh()` functions shall return the inverse hyperbolic cosine, sine, and
 3999 tangent of their argument, respectively.

4000 The `acosh()`, `acoshf()`, and `acoshl()` functions shall return an implementation-defined value (NaN
 4001 or equivalent if available) and set `errno` to [EDOM] when its argument is less than 1.0.

4002 The `atanh()`, `atanhf()`, and `atanhl()` functions shall return an implementation-defined value (NaN
 4003 or equivalent if available) and set `errno` to [EDOM] when its argument has absolute value greater
 4004 than 1.0.

4005 If `x` is NaN, the `asinh()`, `acosh()`, and `atanh()` functions shall return NaN and may set `errno` to
 4006 [EDOM].

4007 **ERRORS**

4008 The `acosh()`, `acoshf()`, and `acoshl()` functions shall fail if:

4009 [EDOM] The `x` argument is less than 1.0.

4010 The `atanh()`, `atanhf()`, and `atanhl()` functions shall fail if:

4011 [EDOM] The `x` argument has an absolute value greater than 1.0.

4012 The `atanh()`, `atanhf()`, and `atanhl()` functions shall fail if:

4013 [ERANGE] The `x` argument has an absolute value equal to 1.0

4014 The `asinh()`, `acosh()`, and `atanh()` functions may fail if:

4015 [EDOM] The value of `x` is NaN.

4016 **EXAMPLES**

4017 None.

4018 **APPLICATION USAGE**

4019 None.

4020 **RATIONALE**

4021 None.

4022 **FUTURE DIRECTIONS**

4023 None.

4024 **SEE ALSO**4025 *cosh()*, *sinh()*, *tanh()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>4026 **CHANGE HISTORY**

4027 First released in Issue 4, Version 2.

4028 **Issue 5**

4029 Moved from X/OPEN UNIX extension to BASE.

4030 **Issue 6**4031 The *acoshf()*, *acoshl()*, *asinhf()*, *asinhf()*, *atanhf()*, and *atanhl()* functions are added for alignment
4032 with the ISO/IEC 9899:1999 standard.

4033 **NAME**4034 aio_cancel — cancel an asynchronous I/O request (**REALTIME**)4035 **SYNOPSIS**

4036 AIO #include <aio.h>

4037 int aio_cancel(int *fildes*, struct aiocb **aiocbp*);

4038

4039 **DESCRIPTION**

4040 The *aio_cancel()* function shall attempt to cancel one or more asynchronous I/O requests
 4041 currently outstanding against file descriptor *fildes*. The *aiocbp* argument points to the
 4042 asynchronous I/O control block for a particular request to be canceled. If *aiocbp* is NULL, then
 4043 all outstanding cancelable asynchronous I/O requests against *fildes* shall be canceled.

4044 Normal asynchronous notification shall occur for asynchronous I/O operations that are
 4045 successfully canceled. If there are requests that cannot be canceled, then the normal
 4046 asynchronous completion process shall take place for those requests when they are completed.

4047 For requested operations that are successfully canceled, the associated error status shall be set to
 4048 [ECANCELED] and the return status shall be -1. For requested operations that are not
 4049 successfully canceled, the *aiocbp* shall not be modified by *aio_cancel()*.

4050 If *aiocbp* is not NULL, then if *fildes* does not have the same value as the file descriptor with which
 4051 the asynchronous operation was initiated, unspecified results occur.

4052 Which operations are cancelable is implementation-defined.

4053 **RETURN VALUE**

4054 The *aio_cancel()* function shall return the value AIO_CANCELED to the calling process if the
 4055 requested operation(s) were canceled. The value AIO_NOTCANCELED shall be returned if at
 4056 least one of the requested operation(s) cannot be canceled because it is in progress. In this case,
 4057 the state of the other operations, if any, referenced in the call to *aio_cancel()* is not indicated by
 4058 the return value of *aio_cancel()*. The application may determine the state of affairs for these
 4059 operations by using *aio_error()*. The value AIO_ALLDONE is returned if all of the operations
 4060 have already completed. Otherwise, the function shall return -1 and set *errno* to indicate the
 4061 error.

4062 **ERRORS**4063 The *aio_cancel()* function shall fail if:4064 [EBADF] The *fildes* argument is not a valid file descriptor.4065 **EXAMPLES**

4066 None.

4067 **APPLICATION USAGE**

4068 The *aio_cancel()* function is part of the Asynchronous Input and Output option and need not be
 4069 available on all implementations.

4070 **RATIONALE**

4071 None.

4072 **FUTURE DIRECTIONS**

4073 None.

4074 **SEE ALSO**4075 *aio_read()*, *aio_write()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**aio.h**>4076 **CHANGE HISTORY**

4077 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4078 **Issue 6**4079 The [ENOSYS] error condition has been removed as stubs need not be provided if an
4080 implementation does not support the Asynchronous Input and Output option.

4081 The APPLICATION USAGE section is added.

4082 **NAME**4083 aio_error — retrieve errors status for an asynchronous I/O operation (**REALTIME**)4084 **SYNOPSIS**4085 AIO `#include <aio.h>`4086 `int aio_error(const struct aiocb *aiocbp);`

4087

4088 **DESCRIPTION**

4089 The `aio_error()` function shall return the error status associated with the **aiocb** structure
 4090 referenced by the `aiocbp` argument. The error status for an asynchronous I/O operation is the
 4091 SIO `errno` value that would be set by the corresponding `read()`, `write()`, `fdatasync()`, or `fsync()`
 4092 operation. If the operation has not yet completed, then the error status shall be equal to
 4093 `[EINPROGRESS]`.

4094 **RETURN VALUE**

4095 If the asynchronous I/O operation has completed successfully, then 0 shall be returned. If the
 4096 asynchronous operation has completed unsuccessfully, then the error status, as described for
 4097 SIO `read()`, `write()`, `fdatasync()`, and `fsync()`, shall be returned. If the asynchronous I/O operation has
 4098 not yet completed, then `[EINPROGRESS]` shall be returned.

4099 **ERRORS**4100 The `aio_error()` function may fail if:

4101 `[EINVAL]` The `aiocbp` argument does not refer to an asynchronous operation whose
 4102 return status has not yet been retrieved.

4103 **EXAMPLES**

4104 None.

4105 **APPLICATION USAGE**

4106 The `aio_error()` function is part of the Asynchronous Input and Output option and need not be
 4107 available on all implementations.

4108 **RATIONALE**

4109 None.

4110 **FUTURE DIRECTIONS**

4111 None.

4112 **SEE ALSO**

4113 `aio_cancel()`, `aio_fsync()`, `aio_read()`, `aio_return()`, `aio_write()`, `close()`, `exec`, `exit()`, `fork()`, `lio_listio()`,
 4114 `lseek()`, `read()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<aio.h>`

4115 **CHANGE HISTORY**

4116 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4117 **Issue 6**

4118 The `[ENOSYS]` error condition has been removed as stubs need not be provided if an
 4119 implementation does not support the Asynchronous Input and Output option.

4120 The APPLICATION USAGE section is added.

4121 NAME

4122 aio_fsync — asynchronous file synchronization (**REALTIME**)

4123 SYNOPSIS

4124 AIO #include <aio.h>

4125 int aio_fsync(int op, struct aiocb *aiocbp);

4126

4127 DESCRIPTION

4128 The *aio_fsync()* function asynchronously forces all I/O operations associated with the file
 4129 indicated by the file descriptor *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp*
 4130 argument and queued at the time of the call to *aio_fsync()* to the synchronized I/O completion
 4131 state. The function call shall return when the synchronization request has been initiated or
 4132 queued to the file or device (even when the data cannot be synchronized immediately).

4133 If *op* is O_DSYNC, all currently queued I/O operations shall be completed as if by a call to
 4134 *fdatasync()*; that is, as defined for synchronized I/O data integrity completion. If *op* is O_SYNC,
 4135 all currently queued I/O operations shall be completed as if by a call to *fsync()*; that is, as
 4136 defined for synchronized I/O file integrity completion. If the *aio_fsync()* function fails, or if the
 4137 operation queued by *aio_fsync()* fails, then, as for *fsync()* and *fdatasync()*, outstanding I/O
 4138 operations are not guaranteed to have been completed.

4139 If *aio_fsync()* succeeds, then it is only the I/O that was queued at the time of the call to
 4140 *aio_fsync()* that is guaranteed to be forced to the relevant completion state. The completion of
 4141 subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized
 4142 fashion.

4143 The *aiocbp* argument refers to an asynchronous I/O control block. The *aiocbp* value may be used
 4144 as an argument to *aio_error()* and *aio_return()* in order to determine the error status and return
 4145 status, respectively, of the asynchronous operation while it is proceeding. When the request is
 4146 queued, the error status for the operation is [EINPROGRESS]. When all data has been
 4147 successfully transferred, the error status shall be reset to reflect the success or failure of the
 4148 operation. If the operation does not complete successfully, the error status for the operation shall
 4149 be set to indicate the error. The *aio_sigevent* member determines the asynchronous notification to
 4150 occur as specified in Section 2.4.1 (on page 528) when all operations have achieved synchronized
 4151 I/O completion. All other members of the structure referenced by *aiocbp* are ignored. If the
 4152 control block referenced by *aiocbp* becomes an illegal address prior to asynchronous I/O
 4153 completion, then the behavior is undefined.

4154 If the *aio_fsync()* function fails or the *aiocbp* indicates an error condition, data is not guaranteed
 4155 to have been successfully transferred.

4156 RETURN VALUE

4157 The *aio_fsync()* function shall return the value 0 to the calling process if the I/O operation is
 4158 successfully queued; otherwise, the function shall return the value -1 and set *errno* to indicate
 4159 the error.

4160 ERRORS

4161 The *aio_fsync()* function shall fail if:

4162 [EAGAIN] The requested asynchronous operation was not queued due to temporary
 4163 resource limitations.

4164 [EBADF] The *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument
 4165 is not a valid file descriptor open for writing.

- 4166 [EINVAL] This implementation does not support synchronized I/O for this file. |
- 4167 [EINVAL] A value of *op* other than O_DSYNC or O_SYNC was specified.
- 4168 In the event that any of the queued I/O operations fail, *aio_fsync()* shall return the error
4169 condition defined for *read()* and *write()*. The error is returned in the error status for the
4170 asynchronous *fsync()* operation, which can be retrieved using *aio_error()*.
- 4171 **EXAMPLES**
- 4172 None.
- 4173 **APPLICATION USAGE**
- 4174 The *aio_fsync()* function is part of the Asynchronous Input and Output option and need not be |
4175 available on all implementations.
- 4176 **RATIONALE**
- 4177 None.
- 4178 **FUTURE DIRECTIONS**
- 4179 None.
- 4180 **SEE ALSO**
- 4181 *fcntl()*, *fdatasync()*, *fsync()*, *open()*, *read()*, *write()*, the Base Definitions volume of |
4182 IEEE Std. 1003.1-200x, <**aio.h**>
- 4183 **CHANGE HISTORY**
- 4184 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
- 4185 **Issue 6**
- 4186 The [ENOSYS] error condition has been removed as stubs need not be provided if an
4187 implementation does not support the Asynchronous Input and Output option. |
- 4188 The APPLICATION USAGE section is added.

4189 **NAME**4190 aio_read — asynchronous read from a file (**REALTIME**)4191 **SYNOPSIS**

4192 AIO #include <aio.h>

4193 int aio_read(struct aiocb *aiocbp);

4194

4195 **DESCRIPTION**

4196 The *aio_read()* function allows the calling process to read *aiocbp->aio_nbytes* from the file
 4197 associated with *aiocbp->aio_fildes* into the buffer pointed to by *aiocbp->aio_buf*. The function call
 4198 shall return when the read request has been initiated or queued to the file or device (even when
 4199 the data cannot be delivered immediately).

4200 PIO If prioritized I/O is supported for this file, then the asynchronous operation is submitted at a
 4201 priority equal to the scheduling priority of the process minus *aiocbp->aio_reqprio*.

4202 The *aiocbp* value may be used as an argument to *aio_error()* and *aio_return()* in order to
 4203 determine the error status and return status, respectively, of the asynchronous operation while it
 4204 is proceeding. If an error condition is encountered during queuing, the function call shall return
 4205 without having initiated or queued the request. The requested operation takes place at the
 4206 absolute position in the file as given by *aio_offset*, as if *lseek()* were called immediately prior to
 4207 the operation with an *offset* equal to *aio_offset* and a *whence* equal to {SEEK_SET}. After a
 4208 successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file
 4209 is unspecified.

4210 The *aiocbp->aio_lio_opcode* field shall be ignored by *aio_read()*.

4211 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio_buf* or
 4212 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O
 4213 completion, then the behavior is undefined.

4214 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

4215 SIO If synchronized I/O is enabled on the file associated with *aiocbp->aio_fildes*, the behavior of this
 4216 function shall be according to the definitions of synchronized I/O data integrity completion and
 4217 synchronized I/O file integrity completion.

4218 For any system action that changes the process memory space while an asynchronous I/O is
 4219 outstanding to the address range being changed, the result of that action is undefined.

4220 For regular files, no data transfer shall occur past the offset maximum established in the open
 4221 file description associated with *aiocbp->aio_fildes*.

4222 **RETURN VALUE**

4223 The *aio_read()* function shall return the value zero to the calling process if the I/O operation is
 4224 successfully queued; otherwise, the function shall return the value -1 and set *errno* to indicate
 4225 the error.

4226 **ERRORS**

4227 The *aio_read()* function shall fail if:

4228 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
 4229 resource limitations.

4230 Each of the following conditions may be detected synchronously at the time of the call to
 4231 *aio_read()*, or asynchronously. If any of the conditions below are detected synchronously, the
 4232 *aio_read()* function shall return -1 and set *errno* to the corresponding value. If any of the
 4233 conditions below are detected asynchronously, the return status of the asynchronous operation

- 4234 is set to `-1`, and the error status of the asynchronous operation is set to the corresponding value.
- 4235 [EBADF] The `aiocbp->aio_fildes` argument is not a valid file descriptor open for reading.
- 4236 [EINVAL] The file offset value implied by `aiocbp->aio_offset` would be invalid, `aiocbp->aio_reqprio` is not a valid value, or `aiocbp->aio_nbytes` is an invalid value.
- 4237
- 4238 In the case that the `aio_read()` successfully queues the I/O operation but the operation is subsequently canceled or encounters an error, the return status of the asynchronous operation is one of the values normally returned by the `read()` function call. In addition, the error status of the asynchronous operation is set to one of the error statuses normally set by the `read()` function call, or one of the following values:
- 4239
- 4240
- 4241
- 4242
- 4243 [EBADF] The `aiocbp->aio_fildes` argument is not a valid file descriptor open for reading.
- 4244 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit `aio_cancel()` request.
- 4245
- 4246 [EINVAL] The file offset value implied by `aiocbp->aio_offset` would be invalid.
- 4247 The following condition may be detected synchronously or asynchronously:
- 4248 [EOVERFLOW] The file is a regular file, `aiocbp->aio_nbytes` is greater than 0, and the starting offset in `aiocbp->aio_offset` is before the end-of-file and is at or beyond the offset maximum in the open file description associated with `aiocbp->aio_fildes`.
- 4249
- 4250
- 4251 **EXAMPLES**
- 4252 None.
- 4253 **APPLICATION USAGE**
- 4254 The `aio_read()` function is part of the Asynchronous Input and Output option and need not be available on all implementations.
- 4255
- 4256 **RATIONALE**
- 4257 None.
- 4258 **FUTURE DIRECTIONS**
- 4259 None.
- 4260 **SEE ALSO**
- 4261 `aio_cancel()`, `aio_error()`, `lio_listio()`, `aio_return()`, `aio_write()`, `close()`, `exec`, `exit()`, `fork()`, `lseek()`, `read()`, the Base Definitions volume of IEEE Std. 1003.1-200x, <`aio.h`>
- 4262
- 4263 **CHANGE HISTORY**
- 4264 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
- 4265 Large File Summit extensions are added.
- 4266 **Issue 6**
- 4267 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.
- 4268
- 4269 The APPLICATION USAGE section is added.
- 4270 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:
- 4271
- 4272 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
 - 4273
 - 4274 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.
 - 4275

4276 **NAME**4277 aio_return — retrieve return status of an asynchronous I/O operation (**REALTIME**)4278 **SYNOPSIS**

4279 AIO #include <aio.h>

4280 ssize_t aio_return(struct aiocb *aiocbp);

4281

4282 **DESCRIPTION**

4283 The *aio_return()* function shall return the return status associated with the **aiocb** structure
 4284 referenced by the *aiocbp* argument. The return status for an asynchronous I/O operation is the
 4285 value that would be returned by the corresponding *read()*, *write()*, or *fsync()* function call. If the
 4286 error status for the operation is equal to [EINPROGRESS], then the return status for the
 4287 operation is undefined. The *aio_return()* function may be called exactly once to retrieve the
 4288 return status of a given asynchronous operation; thereafter, if the same **aiocb** structure is used in
 4289 a call to *aio_return()* or *aio_error()*, an error may be returned. When the **aiocb** structure referred
 4290 to by *aiocbp* is used to submit another asynchronous operation, then *aio_return()* may be
 4291 successfully used to retrieve the return status of that operation.

4292 **RETURN VALUE**

4293 If the asynchronous I/O operation has completed, then the return status, as described for *read()*,
 4294 *write()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has not yet completed,
 4295 the results of *aio_return()* are undefined.

4296 **ERRORS**4297 The *aio_return()* function may fail if:

4298 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose
 4299 return status has not yet been retrieved.

4300 **EXAMPLES**

4301 None.

4302 **APPLICATION USAGE**

4303 The *aio_return()* function is part of the Asynchronous Input and Output option and need not be
 4304 available on all implementations.

4305 **RATIONALE**

4306 None.

4307 **FUTURE DIRECTIONS**

4308 None.

4309 **SEE ALSO**

4310 *aio_cancel()*, *aio_error()*, *aio_fsync()*, *aio_read()*, *aio_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*,
 4311 *lseek()*, *read()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**aio.h**>

4312 **CHANGE HISTORY**

4313 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4314 **Issue 6**

4315 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 4316 implementation does not support the Asynchronous Input and Output option.

4317 The APPLICATION USAGE section is added.

4318 The [EINVAL] error condition is updated as a “may fail”. This is for consistency with the
 4319 DESCRIPTION.

4320 NAME

4321 aio_suspend — wait for an asynchronous I/O request (**REALTIME**)

4322 SYNOPSIS

4323 AIO #include <aio.h>

```
4324 int aio_suspend(const struct aiocb * const list[], int nent,
4325               const struct timespec *timeout);
4326
```

4327 DESCRIPTION

4328 The *aio_suspend()* function shall suspend the calling thread until at least one of the asynchronous
 4329 I/O operations referenced by the *list* argument has completed, until a signal interrupts the
 4330 function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed. If any
 4331 of the **aiocb** structures in the list correspond to completed asynchronous I/O operations (that is,
 4332 the error status for the operation is not equal to [EINPROGRESS]) at the time of the call, the
 4333 function shall return without suspending the calling thread. The *list* argument is an array of
 4334 pointers to asynchronous I/O control blocks. The *nent* argument indicates the number of
 4335 elements in the array. Each **aiocb** structure pointed to has been used in initiating an
 4336 asynchronous I/O request via *aio_read()*, *aio_write()*, or *lio_listio()*. This array may contain
 4337 NULL pointers, which are ignored. If this array contains pointers that refer to **aiocb** structures
 4338 that have not been used in submitting asynchronous I/O, the effect is undefined.

4339 If the time interval indicated in the **timespec** structure pointed to by *timeout* passes before any of
 4340 the I/O operations referenced by *list* are completed, then *aio_suspend()* shall return with an
 4341 error. If the Monotonic Clock option is supported, the clock that shall be used to measure this
 4342 time interval shall be the CLOCK_MONOTONIC clock.

4343 RETURN VALUE

4344 If the *aio_suspend()* function returns after one or more asynchronous I/O operations have
 4345 completed, the function shall return zero. Otherwise, the function shall return a value of -1 and
 4346 set *errno* to indicate the error.

4347 The application may determine which asynchronous I/O completed by scanning the associated
 4348 error and return status using *aio_error()* and *aio_return()*, respectively.

4349 ERRORS

4350 The *aio_suspend()* function shall fail if:

4351 [EAGAIN] No asynchronous I/O indicated in the list referenced by *list* completed in the
 4352 time interval indicated by *timeout*.

4353 [EINTR] A signal interrupted the *aio_suspend()* function. Note that, since each
 4354 asynchronous I/O operation may possibly provoke a signal when it
 4355 completes, this error return may be caused by the completion of one (or more)
 4356 of the very I/O operations being awaited.

4357 EXAMPLES

4358 None.

4359 APPLICATION USAGE

4360 The *aio_suspend()* function is part of the Asynchronous Input and Output option and need not
 4361 be available on all implementations.

4362 RATIONALE

4363 None.

4364 **FUTURE DIRECTIONS**

4365 None.

4366 **SEE ALSO**4367 *aio_read()*, *aio_write()*, *lio_listio()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**aio.h**>4368 **CHANGE HISTORY**

4369 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4370 **Issue 6**4371 The [ENOSYS] error condition has been removed as stubs need not be provided if an
4372 implementation does not support the Asynchronous Input and Output option.

4373 The APPLICATION USAGE section is added.

4374 The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by specifying that the
4375 CLOCK_MONOTONIC clock, if supported, is used.

4376 NAME

4377 aio_write — asynchronous write to a file (**REALTIME**)

4378 SYNOPSIS

4379 AIO #include <aio.h>

4380 int aio_write(struct aiocb *aiocbp);

4381

4382 DESCRIPTION

4383 The *aio_write()* function allows the calling process to write *aiocbp->aio_nbytes* to the file
 4384 associated with *aiocbp->aio_fildes* from the buffer pointed to by *aiocbp->aio_buf*. The function call
 4385 shall return when the write request has been initiated or, at a minimum, queued to the file or
 4386 device.

4387 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted
 4388 at a priority equal to the scheduling priority of the process minus *aiocbp->aio_reqprio*.

4389 The *aiocbp* may be used as an argument to *aio_error()* and *aio_return()* in order to determine the
 4390 error status and return status, respectively, of the asynchronous operation while it is proceeding.

4391 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio_buf* or
 4392 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O
 4393 completion, then the behavior is undefined.

4394 If **O_APPEND** is not set for the file descriptor *aio_fildes*, then the requested operation takes place
 4395 at the absolute position in the file as given by *aio_offset*, as if *lseek()* were called immediately
 4396 prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to **{SEEK_SET}**. If
 4397 **O_APPEND** is set for the file descriptor, write operations append to the file in the same order as
 4398 the calls were made. After a successful call to enqueue an asynchronous I/O operation, the value
 4399 of the file offset for the file is unspecified.

4400 The *aiocbp->aio_lio_opcode* field shall be ignored by *aio_write()*.

4401 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

4402 SIO If synchronized I/O is enabled on the file associated with *aiocbp->aio_fildes*, the behavior of this
 4403 function shall be according to the definitions of synchronized I/O data integrity completion, and
 4404 synchronized I/O file integrity completion.

4405 For any system action that changes the process memory space while an asynchronous I/O is
 4406 outstanding to the address range being changed, the result of that action is undefined.

4407 For regular files, no data transfer shall occur past the offset maximum established in the open
 4408 file description associated with *aiocbp->aio_fildes*.

4409 RETURN VALUE

4410 The *aio_write()* function shall return the value zero to the calling process if the I/O operation is
 4411 successfully queued; otherwise, the function shall return the value **-1** and set *errno* to indicate
 4412 the error.

4413 ERRORS

4414 The *aio_write()* function shall fail if:

4415 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
 4416 resource limitations.

4417 Each of the following conditions may be detected synchronously at the time of the call to
 4418 *aio_write()*, or asynchronously. If any of the conditions below are detected synchronously, the
 4419 *aio_write()* function shall return **-1** and set *errno* to the corresponding value. If any of the

4420 conditions below are detected asynchronously, the return status of the asynchronous operation
 4421 shall be set to -1, and the error status of the asynchronous operation is set to the corresponding
 4422 value.

4423 [EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for writing.

4424 [EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid, *aiocbp-*
 4425 *>aio_reqprio* is not a valid value, or *aiocbp->aio_nbytes* is an invalid value.

4426 In the case that the *aio_write()* successfully queues the I/O operation, the return status of the
 4427 asynchronous operation shall be one of the values normally returned by the *write()* function call.
 4428 If the operation is successfully queued but is subsequently canceled or encounters an error, the
 4429 error status for the asynchronous operation contains one of the values normally set by the
 4430 *write()* function call, or one of the following:

4431 [EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for writing.

4432 [EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid.

4433 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
 4434 *aio_cancel()* request.

4435 The following condition may be detected synchronously or asynchronously:

4436 [EFBIG] The file is a regular file, *aiocbp->aio_nbytes* is greater than 0, and the starting
 4437 offset in *aiocbp->aio_offset* is at or beyond the offset maximum in the open file
 4438 description associated with *aiocbp->aio_fildes*.

4439 EXAMPLES

4440 None.

4441 APPLICATION USAGE

4442 The *aio_write()* function is part of the Asynchronous Input and Output option and need not be
 4443 available on all implementations.

4444 RATIONALE

4445 None.

4446 FUTURE DIRECTIONS

4447 None.

4448 SEE ALSO

4449 *aio_cancel()*, *aio_error()*, *aio_read()*, *aio_return()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*, *lseek()*,
 4450 *write()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**aio.h**>

4451 CHANGE HISTORY

4452 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4453 Large File Summit extensions are added.

4454 Issue 6

4455 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 4456 implementation does not support the Asynchronous Input and Output option.

4457 The APPLICATION USAGE section is added.

4458 The following new requirements on POSIX implementations derive from alignment with the
 4459 Single UNIX Specification:

- 4460 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs
 4461 past the offset maximum established in the open file description associated with *aiocbp-*
 4462 *>aio_fildes*.

4463

- The [EFBIG] error is added as part of the large file support extensions.

4464 **NAME**

4465 alarm — schedule an alarm signal

4466 **SYNOPSIS**

4467 #include <unistd.h>

4468 unsigned alarm(unsigned *seconds*);4469 **DESCRIPTION**

4470 The *alarm()* function shall cause the system to generate a SIGALRM signal for the process after
4471 the number of realtime seconds specified by *seconds* have elapsed. Processor scheduling delays
4472 may prevent the process from handling the signal as soon as it is generated.

4473 If *seconds* is 0, a pending alarm request, if any, is canceled.

4474 Alarm requests are not stacked; only one SIGALRM generation can be scheduled in this manner.
4475 If the SIGALRM signal has not yet been generated, the call shall result in rescheduling the time
4476 at which the SIGALRM signal is generated.

4477 Interactions between *alarm()* and any of *setitimer()*, *ualarm()*, or *usleep()* are unspecified.

4478 **RETURN VALUE**

4479 If there is a previous *alarm()* request with time remaining, *alarm()* shall return a non-zero value
4480 that is the number of seconds until the previous request would have generated a SIGALRM
4481 signal. Otherwise, *alarm()* shall return 0.

4482 **ERRORS**

4483 The *alarm()* function is always successful, and no return value is reserved to indicate an error.

4484 **EXAMPLES**

4485 None.

4486 **APPLICATION USAGE**

4487 The *fork()* function clears pending alarms in the child process. A new process image created by
4488 one of the *exec* functions inherits the time left to an alarm signal in the old process' image.

4489 Application writers should note that the type of the argument *seconds* and the return value of
4490 *alarm()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces
4491 Application cannot pass a value greater than the minimum guaranteed value for {UINT_MAX},
4492 which the ISO C standard sets as 65 535, and any application passing a larger value is restricting
4493 its portability. A different type was considered, but historical implementations, including those
4494 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

4495 Application writers should be aware of possible interactions when the same process uses both
4496 the *alarm()* and *sleep()* functions.

4497 **RATIONALE**

4498 Many historical implementations (including Version 7 and System V) allow an alarm to occur up
4499 to a second early. Other implementations allow alarms up to half a second or one clock tick
4500 early or do not allow them to occur early at all. The latter is considered most appropriate, since it
4501 gives the most predictable behavior, especially since the signal can always be delayed for an
4502 indefinite amount of time due to scheduling. Applications can thus choose the *seconds* argument
4503 as the minimum amount of time they wish to have elapse before the signal.

4504 The term *realtime* here and elsewhere (*sleep()*, *times()*) is intended to mean “wall clock” time as
4505 common English usage, and has nothing to do with “realtime operating systems”. It is in
4506 contrast to *virtual time*, which could be misinterpreted if just *time* were used.

4507 In some implementations, including 4.3 BSD, very large values of the *seconds* argument are
4508 silently rounded down to an implementation-defined maximum value. This maximum is large

4509 enough (on the order of several months) that the effect is not noticeable.

4510 There were two possible choices for alarm generation in multi-threaded applications: generation
4511 for the calling thread or generation for the process. The first option would not have been
4512 particularly useful since the alarm state is maintained on a per-process basis and the alarm that
4513 is established by the last invocation of *alarm()* is the only one that would be active.

4514 Furthermore, allowing generation of an asynchronous signal for a thread would have introduced
4515 an exception to the overall signal model. This requires a compelling reason in order to be
4516 justified.

4517 **FUTURE DIRECTIONS**

4518 None.

4519 **SEE ALSO**

4520 *alarm()*, *exec*, *fork()*, *getitimer()*, *pause()*, *sigaction()*, *sleep()*, *ualarm()*, *usleep()*, the Base
4521 Definitions volume of IEEE Std. 1003.1-200x, <**signal.h**>, <**unistd.h**>

4522 **CHANGE HISTORY**

4523 First released in Issue 1. Derived from Issue 1 of the SVID.

4524 **Issue 4**

4525 The <**unistd.h**> header is included in the SYNOPSIS section.

4526 **Issue 4, Version 2**

4527 The DESCRIPTION is updated to indicate that interactions with the *setitimer()*, *ualarm()*, and
4528 *usleep()* functions are unspecified.

4529 **Issue 6**

4530 The following new requirements on POSIX implementations derive from alignment with the
4531 Single UNIX Specification:

- 4532 • The DESCRIPTION is updated to indicate that interactions with the *setitimer()*, *ualarm()*, and
4533 *usleep()* functions are unspecified.

4534 **NAME**

4535 asctime, asctime_r — convert date and time to a string

4536 **SYNOPSIS**

4537 #include <time.h>

4538 char *asctime(const struct tm *timeptr);

4539 TSF char *asctime_r(const struct tm *restrict tm, char *restrict buf);

4540

4541 **DESCRIPTION**

4542 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 4543 conflict between the requirements described here and the ISO C standard is unintentional. This
 4544 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

4545 The *asctime()* function shall convert the broken-down time in the structure pointed to by *timeptr*
 4546 into a string in the form:

4547 Sun Sep 16 01:03:52 1973\n\0

4548 using the equivalent of the following algorithm:

```

4549 char *asctime(const struct tm *timeptr)
4550 {
4551     static char wday_name[7][3] = {
4552         "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
4553     };
4554     static char mon_name[12][3] = {
4555         "Jan", "Feb", "Mar", "Apr", "May", "Jun",
4556         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
4557     };
4558     static char result[26];

4559     sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
4560             wday_name[timeptr->tm_wday],
4561             mon_name[timeptr->tm_mon],
4562             timeptr->tm_mday, timeptr->tm_hour,
4563             timeptr->tm_min, timeptr->tm_sec,
4564             1900 + timeptr->tm_year);
4565     return result;
4566 }
```

4567 The **tm** structure is defined in the **<time.h>** header.

4568 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
 4569 objects: a broken-down time structure and an array of type **char**. Execution of any of the
 4570 functions may overwrite the information returned in either of these objects by any of the other
 4571 functions.

4572 The *asctime()* function need not be reentrant. A function that is not required to be reentrant is not
 4573 required to be thread-safe.

4574 TSF The *asctime_r()* function shall convert the broken-down time in the structure pointed to by *tm*
 4575 into a string (of the same form as that returned by *asctime()*) that is placed in the user-supplied
 4576 buffer pointed to by *buf* (which contains at least 26 bytes) and then return *buf*.

4577 **RETURN VALUE**

4578 Upon successful completion, *asctime()* shall return a pointer to the string.

4579 TSF Upon successful completion, *asctime_r()* shall return a pointer to a character string containing
4580 the date and time. This string is pointed to by the argument *buf*. If the function is unsuccessful,
4581 it shall return NULL.

4582 **ERRORS**

4583 No errors are defined.

4584 **EXAMPLES**

4585 None.

4586 **APPLICATION USAGE**

4587 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.
4588 This function is included for compatibility with older implementations, and does not support
4589 localized date and time formats. Applications should use *strptime()* to achieve maximum
4590 portability.

4591 The *asctime_r()* function is thread-safe and shall return values in a user-supplied buffer instead
4592 of possibly using a static data area that may be overwritten by each call.

4593 **RATIONALE**

4594 None.

4595 **FUTURE DIRECTIONS**

4596 None.

4597 **SEE ALSO**

4598 *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*, *strptime()*, *time()*, *utime()*,
4599 the Base Definitions volume of IEEE Std. 1003.1-200x, <**time.h**>

4600 **CHANGE HISTORY**

4601 First released in Issue 1. Derived from Issue 1 of the SVID.

4602 **Issue 4**

4603 The location of the **tm** structure is now defined.

4604 The APPLICATION USAGE section is expanded to describe the time-handling functions
4605 generally and to refer users to *strptime()*, which is a locale-dependent time-handling function.

4606 The following change is incorporated for alignment with the ISO C standard:

- 4607 • The type of argument *timeptr* is changed from **struct tm*** to **const struct tm***.

4608 **Issue 5**

4609 Normative text previously in the APPLICATION USAGE section is moved to the
4610 DESCRIPTION.

4611 The *asctime_r()* function is included for alignment with the POSIX Threads Extension.

4612 A note indicating that the *asctime()* function need not be reentrant is added to the
4613 DESCRIPTION.

4614 **Issue 6**

4615 The *asctime_r()* function is marked as part of the Thread-Safe Functions option.

4616 Extensions beyond the ISO C standard are now marked.

4617 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
4618 its avoidance of possibly using a static data area.

4619 The DESCRIPTION of *asctime_r()* is updated to describe the format of the string returned. |

4620 The **restrict** keyword is added to the *asctime_r()* prototype for alignment with the |

4621 ISO/IEC 9899:1999 standard. |

4622 **NAME**

4623 asin, asinf, asinl — arc sine function

4624 **SYNOPSIS**

4625 #include <math.h>

4626 double asin(double x);

4627 float asinf(float x);

4628 long double asinl(long double x);

4629 **DESCRIPTION**

4630 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 4631 conflict between the requirements described here and the ISO C standard is unintentional. This
 4632 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

4633 The *asin()*, *asinf()*, and *asinl()* functions shall compute the principal value of the arc sine of *x*.
 4634 The value of *x* should be in the range $[-1,1]$.

4635 An application wishing to check for error situations should set *errno* to 0, then call *asin()*. If *errno*
 4636 is non-zero on return, or the return value is NaN, an error has occurred.

4637 **RETURN VALUE**

4638 Upon successful completion, the *asin()*, *asinf()*, and *asinl()* functions shall return the arc sine of
 4639 **XSI** *x*, in the range $[-\pi/2, \pi/2]$ radians. If the value of *x* is not in the range $[-1,1]$, and is not $\pm\text{Inf}$ or
 4640 NaN, either 0.0 or NaN shall be returned and *errno* shall be set to [EDOM].

4641 **XSI** If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

4642 If *x* is $\pm\text{Inf}$, either 0.0 shall be returned and *errno* set to [EDOM], or NaN shall be returned and
 4643 *errno* may be set to [EDOM].

4644 If the result underflows, 0.0 shall be returned and *errno* may be set to [ERANGE].

4645 **ERRORS**

4646 The *asin()*, *asinf()*, and *asinl()* functions shall fail if:

4647 **XSI** [EDOM] The value *x* is not $\pm\text{Inf}$ or NaN and is not in the range $[-1,1]$.

4648 The *asin()*, *asinf()*, and *asinl()* functions may fail if:

4649 **XSI** [EDOM] The value of *x* is $\pm\text{Inf}$ or NaN.

4650 [ERANGE] The result underflows

4651 **XSI** No other errors shall occur.

4652 **EXAMPLES**

4653 None.

4654 **APPLICATION USAGE**

4655 None.

4656 **RATIONALE**

4657 None.

4658 **FUTURE DIRECTIONS**

4659 None.

4660 **SEE ALSO**

4661 *isnan()*, *sin()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

4662 **CHANGE HISTORY**

4663 First released in Issue 1. Derived from Issue 1 of the SVID.

4664 **Issue 4**

4665 References to *matherr()* are removed.

4666 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the
4667 ISO C standard and to rationalize error handling in the mathematics functions.

4668 The return value specified for [EDOM] is marked as an extension.

4669 **Issue 5**

4670 The DESCRIPTION is updated to indicate how an application should check for an error. This
4671 text was previously published in the APPLICATION USAGE section.

4672 **Issue 6**

4673 The *asinf()* and *asinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

4674 **NAME**

4675 asinh — hyperbolic arc sine

4676 **SYNOPSIS**

4677 xSI #include <math.h>

4678 double asinh(double x);

4679

4680 **DESCRIPTION**

4681 Refer to *acosh()*.

4682 **NAME**

4683 assert — insert program diagnostics

4684 **SYNOPSIS**

4685 #include <assert.h>

4686 void assert(*scalar expression*);4687 **DESCRIPTION**

4688 **cx** The functionality described on this reference page is aligned with the ISO C standard. Any
4689 conflict between the requirements described here and the ISO C standard is unintentional. This
4690 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

4691 The *assert()* macro inserts diagnostics into programs; it expands to a **void** expression. When it is
4692 executed, if *expression* (which shall have a **scalar** type) is false (that is, compares equal to 0),
4693 *assert()* shall write information about the particular call that failed on *stderr* and shall call *abort()*.

4694 The information written about the call that failed shall include the text of the argument, the
4695 name of the source file, the source file line number, and the name of the enclosing function, the
4696 latter are, respectively, the values of the preprocessing macros `__FILE__` and `__LINE__` and of
4697 the identifier `__func__`.

4698 Forcing a definition of the name `NDEBUG`, either from the compiler command line or with the
4699 preprocessor control statement `#define NDEBUG` ahead of the `#include <assert.h>` statement,
4700 shall stop assertions from being compiled into the program.

4701 **RETURN VALUE**4702 The *assert()* macro shall return no value.4703 **ERRORS**

4704 No errors are defined.

4705 **EXAMPLES**

4706 None.

4707 **APPLICATION USAGE**

4708 None.

4709 **RATIONALE**

4710 None.

4711 **FUTURE DIRECTIONS**

4712 None.

4713 **SEE ALSO**4714 *abort()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<assert.h>`, *stderr*4715 **CHANGE HISTORY**

4716 First released in Issue 1. Derived from Issue 1 of the SVID.

4717 **Issue 4**

4718 The APPLICATION USAGE section is merged into the DESCRIPTION.

4719 **Issue 6**4720 The prototype for the *expression* argument to *assert()* is changed from **int** to **scalar** for alignment
4721 with the ISO/IEC 9899:1999 standard.4722 The DESCRIPTION of *assert()* is updated for alignment with the ISO/IEC 9899:1999 standard.

4723 **NAME**

4724 atan, atanf, atanl — arc tangent function

4725 **SYNOPSIS**

4726 #include <math.h>

4727 double atan(double x);

4728 float atanf(float x);

4729 long double atanl(long double x);

4730 **DESCRIPTION**

4731 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 4732 conflict between the requirements described here and the ISO C standard is unintentional. This
 4733 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

4734 The *atan()*, *atanf()*, and *atanl()* functions shall compute the principal value of the arc tangent of
 4735 *x*.

4736 An application wishing to check for error situations should set *errno* to 0 before calling *atan()*. If
 4737 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

4738 **RETURN VALUE**

4739 Upon successful completion, the *atan()*, *atanf()*, and *atanl()* functions shall return the arc
 4740 tangent of *x* in the range $[-\pi/2, \pi/2]$ radians.

4741 **XSI** If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

4742 If the result underflows, 0.0 shall be returned and *errno* may be set to [ERANGE].

4743 **ERRORS**

4744 The *atan()*, *atanf()*, and *atanl()* functions may fail if:

4745 **XSI** [EDOM] The value of *x* is NaN.

4746 [ERANGE] The result underflows

4747 **XSI** No other errors shall occur.

4748 **EXAMPLES**

4749 None.

4750 **APPLICATION USAGE**

4751 None.

4752 **RATIONALE**

4753 None.

4754 **FUTURE DIRECTIONS**

4755 None.

4756 **SEE ALSO**4757 *atan2()*, *isnan()*, *tan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>4758 **CHANGE HISTORY**

4759 First released in Issue 1. Derived from Issue 1 of the SVID.

4760 **Issue 4**4761 References to *matherr()* are removed.

4762 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the
 4763 ISO C standard and to rationalize error handling in the mathematics functions.

- 4764 The return value specified for [EDOM] is marked as an extension.
- 4765 **Issue 5**
- 4766 The DESCRIPTION is updated to indicate how an application should check for an error. This
- 4767 text was previously published in the APPLICATION USAGE section.
- 4768 **Issue 6**
- 4769 The *atanf()* and *atanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

4770 **NAME**4771 `atan2` — arc tangent function4772 **SYNOPSIS**4773 `#include <math.h>`4774 `double atan2(double y, double x);`4775 `float atan2f(float y, float x);`4776 `long double atan2l(long double y, long double x);`4777 **DESCRIPTION**

4778 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 4779 conflict between the requirements described here and the ISO C standard is unintentional. This
 4780 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

4781 The `atan2()`, `atan2f()`, and `atan2l()` functions shall compute the principal value of the arc tangent
 4782 of y/x , using the signs of both arguments to determine the quadrant of the return value.

4783 An application wishing to check for error situations should set `errno` to 0 before calling `atan2()`.
 4784 If `errno` is non-zero on return, or the return value is NaN, an error has occurred.

4785 **RETURN VALUE**

4786 Upon successful completion, the `atan2()`, `atan2f()`, and `atan2l()` functions shall return the arc
 4787 tangent of y/x in the range $[-\pi, \pi]$ radians. If both arguments are 0.0, an implementation-defined
 4788 value is returned and `errno` may be set to [EDOM].

4789 **XSI** If x or y is NaN, NaN shall be returned and `errno` may be set to [EDOM].

4790 If the result underflows, 0.0 shall be returned and `errno` may be set to [ERANGE].

4791 **ERRORS**

4792 The `atan2()`, `atan2f()`, and `atan2l()` functions may fail if:

4793 **XSI** [EDOM] Both arguments are 0.0 or one or more of the arguments is NaN.

4794 [ERANGE] The result underflows

4795 **XSI** No other errors shall occur.

4796 **EXAMPLES**

4797 None.

4798 **APPLICATION USAGE**

4799 None.

4800 **RATIONALE**

4801 None.

4802 **FUTURE DIRECTIONS**

4803 None.

4804 **SEE ALSO**4805 `atan()`, `isnan()`, `tan()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<math.h>`4806 **CHANGE HISTORY**

4807 First released in Issue 1. Derived from Issue 1 of the SVID.

4808 **Issue 4**4809 References to `matherr()` are removed.

4810 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the
 4811 ISO C standard and to rationalize error handling in the mathematics functions.

4812 The return value specified for [EDOM] is marked as an extension.

4813 **Issue 5**

4814 The DESCRIPTION is updated to indicate how an application should check for an error. This
4815 text was previously published in the APPLICATION USAGE section.

4816 **NAME**

4817 atanh — hyperbolic arc tangent

4818 **SYNOPSIS**

4819 xSI #include <math.h>

4820 double atanh(double x);

4821

4822 **DESCRIPTION**

4823 Refer to *acosh()*.

4824 **NAME**

4825 atexit — register a function to run at process termination

4826 **SYNOPSIS**

4827 #include <stdlib.h>

4828 int atexit(void (**func*)(void));4829 **DESCRIPTION**4830 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
4831 conflict between the requirements described here and the ISO C standard is unintentional. This
4832 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.4833 The *atexit()* function registers the function pointed to by *func*, to be called without arguments at
4834 normal program termination. At normal program termination, all functions registered by the
4835 **CX** *atexit()* function shall be called, in the reverse order of their registration. Normal termination
4836 occurs either by a call to *exit()* or a return from *main()*.4837 At least 32 functions can be registered with *atexit()*.4838 **CX** After a successful call to any of the *exec* functions, any functions previously registered by *atexit()*
4839 shall no longer be registered.4840 **RETURN VALUE**4841 Upon successful completion, *atexit()* shall return 0; otherwise, it shall return a non-zero value.4842 **ERRORS**

4843 No errors are defined.

4844 **EXAMPLES**

4845 None.

4846 **APPLICATION USAGE**4847 The functions registered by a call to *atexit()* must return to ensure that all registered functions
4848 are called.4849 The application should call *sysconf()* to obtain the value of {ATEXIT_MAX}, the number of
4850 functions that can be registered. There is no way for an application to tell how many functions
4851 have already been registered with *atexit()*.4852 **RATIONALE**

4853 None.

4854 **FUTURE DIRECTIONS**

4855 None.

4856 **SEE ALSO**4857 *exit()*, *sysconf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>4858 **CHANGE HISTORY**

4859 First released in Issue 4. Derived from the ANSI C standard.

4860 **Issue 4, Version 2**4861 The APPLICATION USAGE section is updated to indicate how an application can determine the
4862 setting of {ATEXIT_MAX}, which is a constant added for X/OPEN UNIX conformance.4863 **Issue 6**

4864 Extensions beyond the ISO C standard are now marked.

4865 **NAME**

4866 atof — convert a string to double-precision number

4867 **SYNOPSIS**

4868 #include <stdlib.h>

4869 double atof(const char *str);

4870 **DESCRIPTION**

4871 cx The functionality described on this reference page is aligned with the ISO C standard. Any
4872 conflict between the requirements described here and the ISO C standard is unintentional. This
4873 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

4874 The call *atof(str)* shall be equivalent to:

4875 strtod(str, (char **)NULL),

4876 except that the handling of errors may differ. If the value cannot be represented, the behavior is
4877 undefined.

4878 **RETURN VALUE**4879 The *atof()* function shall return the converted value if the value can be represented.4880 **ERRORS**

4881 No errors are defined.

4882 **EXAMPLES**

4883 None.

4884 **APPLICATION USAGE**

4885 The *atof()* function is subsumed by *strtod()* but is retained because it is used extensively in
4886 existing code. If the number is not known to be in range, *strtod()* should be used because *atof()* is
4887 not required to perform any error checking.

4888 **RATIONALE**

4889 None.

4890 **FUTURE DIRECTIONS**

4891 None.

4892 **SEE ALSO**4893 *strtod()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>4894 **CHANGE HISTORY**

4895 First released in Issue 1. Derived from Issue 1 of the SVID.

4896 **Issue 4**4897 Reference to how *str* is converted is removed from the DESCRIPTION.

4898 The APPLICATION USAGE section is added.

4899 The following change is incorporated for alignment with the ISO C standard:

- 4900
- The type of argument *str* is changed from **char*** to **const char***.

4901 **NAME**

4902 atoi — convert a string to an integer

4903 **SYNOPSIS**

4904 #include <stdlib.h>

4905 int atoi(const char *str);

4906 **DESCRIPTION**

4907 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 4908 conflict between the requirements described here and the ISO C standard is unintentional. This
 4909 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

4910 The call *atoi(str)* shall be equivalent to:

4911 (int) strtol(str, (char **)NULL, 10)

4912 except that the handling of errors may differ. If the value cannot be represented, the behavior is
 4913 undefined.

4914 **RETURN VALUE**4915 The *atoi()* function shall return the converted value if the value can be represented.4916 **ERRORS**

4917 No errors are defined.

4918 **EXAMPLES**4919 **Converting an Argument**

4920 The following example checks for proper usage of the program. If there is an argument and the
 4921 decimal conversion of this argument (obtained using *atoi()*) is greater than 0, then the program
 4922 has a valid number of minutes to wait for an event.

```
4923         #include <stdlib.h>
4924         #include <stdio.h>
4925         ...
4926         int minutes_to_event;
4927         ...
4928         if (argc < 2 || ((minutes_to_event = atoi (argv[1]))) <= 0) {
4929             fprintf(stderr, "Usage: %s minutes\n", argv[0]); exit(1);
4930         }
4931         ...
```

4932 **APPLICATION USAGE**

4933 The *atoi()* function is subsumed by *strtol()* but is retained because it is used extensively in
 4934 existing code. If the number is not known to be in range, *strtol()* should be used because *atoi()* is
 4935 not required to perform any error checking.

4936 **RATIONALE**

4937 None.

4938 **FUTURE DIRECTIONS**

4939 None.

4940 **SEE ALSO**4941 *strtol()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>

4942 **CHANGE HISTORY**

4943 First released in Issue 1. Derived from Issue 1 of the SVID.

4944 **Issue 4**

4945 Reference to how *str* is converted is removed from the DESCRIPTION.

4946 The APPLICATION USAGE section is added.

4947 The following change is incorporated for alignment with the ISO C standard:

- 4948 • The type of argument *str* is changed from **char*** to **const char***.

4949 **NAME**4950 `atol, atoll` — convert a string to a long integer4951 **SYNOPSIS**4952 `#include <stdlib.h>`4953 `long atol(const char *str);`4954 `long long atoll(const char *nptr);`4955 **DESCRIPTION**

4956 **cx** The functionality described on this reference page is aligned with the ISO C standard. Any
 4957 conflict between the requirements described here and the ISO C standard is unintentional. This
 4958 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

4959 The call `atol(str)` shall be equivalent to:4960 `strtoul(str, (char **)NULL, 10)`4961 The call `atoll(str)` shall be equivalent to:4962 `strtoll(nptr, (char **)NULL, 10)`

4963 except that the handling of errors may differ. If the value cannot be represented, the behavior is
 4964 undefined.

4965 **RETURN VALUE**

4966 These functions shall return the converted value if the value can be represented.

4967 **ERRORS**

4968 No errors are defined.

4969 **EXAMPLES**

4970 None.

4971 **APPLICATION USAGE**

4972 The `atol()` function is subsumed by `strtoul()` but is retained because it is used extensively in
 4973 existing code. If the number is not known to be in range, `strtoul()` should be used because `atol()` is
 4974 not required to perform any error checking.

4975 **RATIONALE**

4976 None.

4977 **FUTURE DIRECTIONS**

4978 None.

4979 **SEE ALSO**4980 `strtoul()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<stdlib.h>`4981 **CHANGE HISTORY**

4982 First released in Issue 1. Derived from Issue 1 of the SVID.

4983 **Issue 4**4984 Reference to how `str` is converted is removed from the DESCRIPTION.

4985 The APPLICATION USAGE section is added.

4986 The following changes are incorporated for alignment with the ISO C standard:

- 4987 • The type of argument `str` is changed from `char*` to `const char*`.
- 4988 • The return type of the function is expanded to `long`.

4989 **Issue 6**
4990

The *atoll()* function is added for alignment with the ISO/IEC 9899:1999 standard.

4991 **NAME**4992 **basename** — return the last component of a path name4993 **SYNOPSIS**4994 **XSI** #include <libgen.h>

4995 char *basename(char *path);

4996

4997 **DESCRIPTION**4998 The *basename()* function shall take the path name pointed to by *path* and return a pointer to the
4999 final component of the path name, deleting any trailing '/' characters.5000 If the string consists entirely of the '/' character, *basename()* shall return a pointer to the string
5001 "/". If the string is exactly "/", it is implementation-defined whether '/' or "/" is
5002 returned.5003 If *path* is a null pointer or points to an empty string, *basename()* shall return a pointer to the
5004 string ".".5005 The *basename()* function may modify the string pointed to by *path*, and may return a pointer to
5006 static storage that may then be overwritten by a subsequent call to *basename()*.5007 The *basename()* function need not be reentrant. A function that is not required to be reentrant is
5008 not required to be thread-safe.5009 **RETURN VALUE**5010 The *basename()* function shall return a pointer to the final component of *path*.5011 **ERRORS**

5012 No errors are defined.

5013 **EXAMPLES**5014 **Using basename()**5015 The following program fragment returns a pointer to the value *lib*, which is the base name of
5016 */usr/lib*.5017 #include <libgen.h>
5018 ...
5019 char *name = "/usr/lib";
5020 char *base;

5021 base = basename(name);
5022 ...5023 **Sample Input and Output Strings for basename()**5024 In the following table, the input string is the value pointed to by *path*, and the output string is
5025 the return value of the *basename()* function.

5026

5027

5028

5029

Input String	Output String
"/usr/lib"	"lib"
"/usr/"	"usr"
"/"	"/"

5030 **APPLICATION USAGE**

5031 None.

5032 **RATIONALE**

5033 None.

5034 **FUTURE DIRECTIONS**

5035 None.

5036 **SEE ALSO**5037 *dirname()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**libgen.h**>, the Shell and5038 Utilities volume of IEEE Std. 1003.1-200x, *basename*5039 **CHANGE HISTORY**

5040 First released in Issue 4, Version 2.

5041 **Issue 5**

5042 Moved from X/OPEN UNIX extension to BASE.

5043 Normative text previously in the APPLICATION USAGE section is moved to the
5044 DESCRIPTION.

5045 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

5046 **Issue 6**

5047 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

5048 **NAME**5049 bcmp — memory operations (**LEGACY**)5050 **SYNOPSIS**

5051 xSI #include <strings.h>

5052 int bcmp(const void *s1, const void *s2, size_t n);

5053

5054 **DESCRIPTION**5055 The *bcmp()* function shall compare the first *n* bytes of the area pointed to by *s1* with the area pointed to by *s2*.5057 **RETURN VALUE**5058 The *bcmp()* function shall return 0 if *s1* and *s2* are identical; otherwise, it shall return non-zero. Both areas are assumed to be *n* bytes long. If the value of *n* is 0, *bcmp()* shall return 0.5060 **ERRORS**

5061 No errors are defined.

5062 **EXAMPLES**

5063 None.

5064 **APPLICATION USAGE**5065 *memcmp()* is preferred over this function.5066 For maximum portability, it is recommended to replace the function call to *bcmp()* as follows:

5067 #define bcmp(b1,b2,len) memcmp((b1), (b2), (size_t)(len))

5068 **RATIONALE**

5069 None.

5070 **FUTURE DIRECTIONS**

5071 This function may be withdrawn in a future version.

5072 **SEE ALSO**5073 *memcmp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <strings.h>5074 **CHANGE HISTORY**

5075 First released in Issue 4, Version 2.

5076 **Issue 5**

5077 Moved from X/OPEN UNIX extension to BASE.

5078 **Issue 6**

5079 This function is marked LEGACY.

5080 **NAME**

5081 **bcopy** — memory operations (**LEGACY**)

5082 **SYNOPSIS**

```
5083 xSI        #include <strings.h>
```

```
5084        void bcopy(const void *s1, void *s2, size_t n);
```

5085

5086 **DESCRIPTION**

5087 The *bcopy()* function shall copy *n* bytes from the area pointed to by *s1* to the area pointed to by *s2*.

5088 The bytes are copied correctly even if the area pointed to by *s1* overlaps the area pointed to by *s2*.

5091 **RETURN VALUE**

5092 The *bcopy()* function shall return no value.

5093 **ERRORS**

5094 No errors are defined.

5095 **EXAMPLES**

5096 None.

5097 **APPLICATION USAGE**

5098 *memmove()* is preferred over this function.

5099 The following are approximately equivalent (note the order of the arguments):

```
5100        bcopy(s1,s2,n) ~ memmove(s2,s1,n)
```

5101 For maximum portability, it is recommended to replace the function call to *bcopy()* as follows:

```
5102        #define bcopy(b1,b2,len) (memmove((b2), (b1), (len)), (void) 0)
```

5103 **RATIONALE**

5104 None.

5105 **FUTURE DIRECTIONS**

5106 This function may be withdrawn in a future version.

5107 **SEE ALSO**

5108 *memmove()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**strings.h**>

5109 **CHANGE HISTORY**

5110 First released in Issue 4, Version 2.

5111 **Issue 5**

5112 Moved from X/OPEN UNIX extension to BASE.

5113 **Issue 6**

5114 This function is marked LEGACY.

5115 **NAME**

5116 bind — bind a name to a socket

5117 **SYNOPSIS**

5118 #include <sys/socket.h>

5119 int bind(int *socket*, const struct sockaddr **address*,
5120 socklen_t *address_len*);5121 **DESCRIPTION**5122 The *bind()* function shall assign a local socket address *address* to a socket identified by descriptor
5123 *socket* that has no local socket address assigned. Sockets created with the *socket()* function are
5124 initially unnamed; they are identified only by their address family.5125 The *bind()* function takes the following arguments:5126 *socket* Specifies the file descriptor of the socket to be bound.5127 *address* Points to a **sockaddr** structure containing the address to be bound to the
5128 socket. The length and format of the address depend on the address family of
5129 the socket.5130 *address_len* Specifies the length of the **sockaddr** structure pointed to by the *address*
5131 argument.5132 The socket specified by *socket* may require the process to have appropriate privileges to use the
5133 *bind()* function.5134 **RETURN VALUE**5135 Upon successful completion, *bind()* shall return 0; otherwise, -1 shall be returned and *errno* set
5136 to indicate the error.5137 **ERRORS**5138 The *bind()* function shall fail if:

5139 [EADDRINUSE]

5140 The specified address is already in use.

5141 [EADDRNOTAVAIL]

5142 The specified address is not available from the local machine.

5143 [EAFNOSUPPORT]

5144 The specified address is not a valid address for the address family of the
5145 specified socket.

5146 [EBADF]

5146 The *socket* argument is not a valid file descriptor.

5147 [EINVAL]

5147 The socket is already bound to an address, and the protocol does not support
5148 binding to a new address; or the socket has been shut down.

5149 [ENOTSOCK]

5149 The *socket* argument does not refer to a socket.

5150 [EOPNOTSUPP]

5150 The socket type of the specified socket does not support binding to an
5151 address.5152 If the address family of the socket is AF_UNIX, then *bind()* shall fail if:

5153 [EACCES]

5153 A component of the path prefix denies search permission, or the requested
5154 name requires writing in a directory with a mode that denies write
5155 permission.

5156	[EDESTADDRREQ] or [EISDIR]	
5157		The <i>address</i> argument is a null pointer.
5158	[EIO]	An I/O error occurred.
5159	[ELOOP]	A loop exists in symbolic links encountered during resolution of the path name in <i>address</i> .
5160		
5161	[ENAMETOOLONG]	
5162		A component of a path name exceeded {NAME_MAX} characters, or an entire path name exceeded {PATH_MAX} characters.
5163		
5164	[ENOENT]	A component of the path name does not name an existing file or the path name is an empty string.
5165		
5166	[ENOTDIR]	A component of the path prefix of the path name in <i>address</i> is not a directory.
5167	[EROFS]	The name would reside on a read-only file system.
5168		The <i>bind()</i> function may fail if:
5169	[EACCES]	The specified address is protected and the current user does not have permission to bind to it.
5170		
5171	[EINVAL]	The <i>address_len</i> argument is not a valid length for the address family.
5172	[EISCONN]	The socket is already connected.
5173	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the path name in <i>address</i> .
5174		
5175	[ENAMETOOLONG]	
5176		Path name resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.
5177		
5178	[ENOBUFS]	Insufficient resources were available to complete the call.
5179	EXAMPLES	
5180		None.
5181	APPLICATION USAGE	
5182		An application program can retrieve the assigned socket name with the <i>getsockname()</i> function.
5183	RATIONALE	
5184		None.
5185	FUTURE DIRECTIONS	
5186		None.
5187	SEE ALSO	
5188		<i>connect()</i> , <i>getsockname()</i> , <i>listen()</i> , <i>socket()</i> , the Base Definitions volume of IEEE Std. 1003.1-200x,
5189		< sys/socket.h >
5190	CHANGE HISTORY	
5191		First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

5192 **NAME**5193 **bsd_signal** — simplified signal facilities5194 **SYNOPSIS**5195 **OB** #include <signal.h>5196 void (*bsd_signal(int *sig*, void (**func*)(int)))(int);

5197

5198 **DESCRIPTION**5199 The *bsd_signal()* function provides a partially compatible interface for programs written to
5200 historical system interfaces (see APPLICATION USAGE).5201 The function call *bsd_signal(sig, func)* has an effect as if implemented as:5202 void (*bsd_signal(int *sig*, void (**func*)(int)))(int)

5203 {

5204 struct sigaction *act*, *oact*;5205 *act*.sa_handler = *func*;5206 *act*.sa_flags = SA_RESTART;5207 sigemptyset(&*act*.sa_mask);5208 sigaddset(&*act*.sa_mask, *sig*);5209 if (sigaction(*sig*, &*act*, &*oact*) == -1)

5210 return(SIG_ERR);

5211 return(*oact*.sa_handler);

5212 }

5213 The handler function should be declared:

5214 void handler(int *sig*);5215 where *sig* is the signal number. The behavior is undefined if *func* is a function that takes more
5216 than one argument, or an argument of a different type.5217 **RETURN VALUE**5218 Upon successful completion, *bsd_signal()* shall return the previous action for *sig*. Otherwise,
5219 SIG_ERR shall be returned and *errno* shall be set to indicate the error.5220 **ERRORS**5221 Refer to *sigaction()*.5222 **EXAMPLES**

5223 None.

5224 **APPLICATION USAGE**5225 This function is a direct replacement for the BSD *signal()* function for simple applications that
5226 are installing a single-argument signal handler function. If a BSD signal handler function is being
5227 installed that expects more than one argument, the application has to be modified to use
5228 *sigaction()*. The *bsd_signal()* function differs from *signal()* in that the SA_RESTART flag is set
5229 and the SA_RESETHAND is clear when *bsd_signal()* is used. The state of these flags is not
5230 specified for *signal()*.5231 It is recommended that new applications use the *sigaction()* function.5232 **RATIONALE**

5233 None.

5234 **FUTURE DIRECTIONS**

5235 None.

5236 **SEE ALSO**5237 *sigaction()*, *sigaddset()*, *sigemptyset()*, *signal()*, the Base Definitions volume of
5238 IEEE Std. 1003.1-200x, <**signal.h**>5239 **CHANGE HISTORY**

5240 First released in Issue 4, Version 2.

5241 **Issue 5**

5242 Moved from X/OPEN UNIX extension to BASE.

5243 **Issue 6**

5244 This function is marked obsolescent.


```

5291     int node_compare(const void *, const void *);
5292     char str_space[20]; /* Space to read string into. */
5293     .
5294     .
5295     .
5296     node.string = str_space;
5297     while (scanf("%s", node.string) != EOF) {
5298         node_ptr = (struct node *)bsearch((void *)&node,
5299             (void *)table, TABSIZE,
5300             sizeof(struct node), node_compare);
5301         if (node_ptr != NULL) {
5302             (void)printf("string = %20s, length = %d\n",
5303                 node_ptr->string, node_ptr->length);
5304         } else {
5305             (void)printf("not found: %s\n", node.string);
5306         }
5307     }
5308 }
5309 /*
5310     This routine compares two nodes based on an
5311     alphabetical ordering of the string field.
5312 */
5313 int
5314 node_compare(const void *node1, const void *node2)
5315 {
5316     return strcoll(((const struct node *)node1)->string,
5317         ((const struct node *)node2)->string);
5318 }

```

5319 APPLICATION USAGE

5320 The pointers to the key and the element at the base of the table should be of type pointer-to-
5321 element.

5322 The comparison function need not compare every byte, so arbitrary data may be contained in
5323 the elements in addition to the values being compared.

5324 In practice, the array is usually sorted according to the comparison function.

5325 RATIONALE

5326 None.

5327 FUTURE DIRECTIONS

5328 None.

5329 SEE ALSO

5330 *hcreate()*, *lsearch()*, *qsort()*, *tsearch()*, the Base Definitions volume of IEEE Std. 1003.1-200x,
5331 `<stdlib.h>`

5332 CHANGE HISTORY

5333 First released in Issue 1. Derived from Issue 1 of the SVID.

5334 Issue 4

5335 Text indicating the need for various casts is removed from the APPLICATION USAGE section.

5336 The code in the EXAMPLES section is changed to use *strcoll()* instead of *strcmp()* in
5337 *node_compare()*.

- 5338 The return value and the contents of the array are now requirements on the application.
- 5339 The DESCRIPTION is changed to specify the order of arguments.
- 5340 The following changes are incorporated for alignment with the ISO C standard:
- 5341 • The type of arguments *key* and *base*, and the type of arguments to *compar*, are changed from
 - 5342 **void*** to **const void***.
 - 5343 • The requirement that the table be sorted according to *compar* is removed from the
 - 5344 DESCRIPTION.
- 5345 **Issue 6**
- 5346 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

5347 **NAME**

5348 btowc — single-byte to wide-character conversion

5349 **SYNOPSIS**

5350 #include <stdio.h>

5351 #include <wchar.h>

5352 wint_t btowc(int c);

5353 **DESCRIPTION**

5354 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5355 conflict between the requirements described here and the ISO C standard is unintentional. This
5356 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

5357 The *btowc()* function shall determine whether *c* constitutes a valid (one-byte) character in the
5358 initial shift state.

5359 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

5360 **RETURN VALUE**

5361 The *btowc()* function shall return WEOF if *c* has the value EOF or if (**unsigned char**) *c* does not
5362 constitute a valid (one-byte) character in the initial shift state. Otherwise, it shall return the
5363 wide-character representation of that character.

5364 **ERRORS**

5365 No errors are defined.

5366 **EXAMPLES**

5367 None.

5368 **APPLICATION USAGE**

5369 None.

5370 **RATIONALE**

5371 None.

5372 **FUTURE DIRECTIONS**

5373 None.

5374 **SEE ALSO**5375 *wctob()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <wchar.h>5376 **CHANGE HISTORY**

5377 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
5378 (E).

5379 **NAME**5380 bzero — memory operations (**LEGACY**)5381 **SYNOPSIS**

5382 XSI #include <strings.h>

5383 void bzero(void *s, size_t n);

5384

5385 **DESCRIPTION**5386 The *bzero()* function shall place *n* zero-valued bytes in the area pointed to by *s*.5387 **RETURN VALUE**5388 The *bzero()* function shall return no value.5389 **ERRORS**

5390 No errors are defined.

5391 **EXAMPLES**

5392 None.

5393 **APPLICATION USAGE**5394 *memset()* is preferred over this function.5395 For maximum portability, it is recommended to replace the function call to *bzero()* as follows:

5396 #define bzero(b,len) (memset((b), '\0', (len)), (void) 0)

5397 **RATIONALE**

5398 None.

5399 **FUTURE DIRECTIONS**

5400 This function may be withdrawn in a future version.

5401 **SEE ALSO**5402 *memset()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <strings.h>5403 **CHANGE HISTORY**

5404 First released in Issue 4, Version 2.

5405 **Issue 5**

5406 Moved from X/OPEN UNIX extension to BASE.

5407 **Issue 6**

5408 This function is marked LEGACY.

5409 **NAME**

5410 cabs, cabsf, cabsl — return a complex absolute value

5411 **SYNOPSIS**

5412 #include <complex.h>

5413 double cabs(double complex *z*);5414 float cabsf(float complex *z*);5415 long double cabsl(long double complex *z*);5416 **DESCRIPTION**

5417 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5418 conflict between the requirements described here and the ISO C standard is unintentional. This
5419 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

5420 These functions shall compute the complex absolute value (also called norm, modulus, or
5421 magnitude) of *z*.

5422 **RETURN VALUE**

5423 These functions shall return the complex absolute value.

5424 **ERRORS**

5425 No errors are defined.

5426 **EXAMPLES**

5427 None.

5428 **APPLICATION USAGE**

5429 None.

5430 **RATIONALE**

5431 None.

5432 **FUTURE DIRECTIONS**

5433 None.

5434 **SEE ALSO**

5435 The Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>

5436 **CHANGE HISTORY**

5437 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5438 **NAME**5439 `acos`, `acosf`, `acosl` — complex arc cosine functions5440 **SYNOPSIS**5441 `#include <complex.h>`5442 `double complex cacos(double complex z);`5443 `float complex cacosf(float complex z);`5444 `long double complex cacosl(long double complex z);`5445 **DESCRIPTION**5446 `CX` The functionality described on this reference page is aligned with the ISO C standard. Any
5447 conflict between the requirements described here and the ISO C standard is unintentional. This
5448 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.5449 These functions shall compute the complex arc cosine of z , with branch cuts outside the interval
5450 $[-1, +1]$ along the real axis.5451 **RETURN VALUE**5452 These functions shall return the complex arc cosine value, in the range of a strip mathematically
5453 unbounded along the imaginary axis and in the interval $[0, \pi]$ along the real axis.5454 **ERRORS**

5455 No errors are defined.

5456 **EXAMPLES**

5457 None.

5458 **APPLICATION USAGE**

5459 None.

5460 **RATIONALE**

5461 None.

5462 **FUTURE DIRECTIONS**

5463 None.

5464 **SEE ALSO**5465 `ccos()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<complex.h>`5466 **CHANGE HISTORY**

5467 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5468 **NAME**

5469 cacosh, cacoshf, cacoshl — complex arc hyperbolic cosine functions

5470 **SYNOPSIS**

5471 #include <complex.h>

5472 double complex cacosh(double complex *z*);5473 float complex cacoshf(float complex *z*);5474 long double complex cacoshl(long double complex *z*);5475 **DESCRIPTION**

5476 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5477 conflict between the requirements described here and the ISO C standard is unintentional. This
5478 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

5479 These functions shall compute the complex arc hyperbolic cosine of *z*, with a branch cut at
5480 values less than 1 along the real axis.

5481 **RETURN VALUE**

5482 These functions shall return the complex arc hyperbolic cosine value, in the range of a half-strip
5483 of non-negative values along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary axis.

5484 **ERRORS**

5485 No errors are defined.

5486 **EXAMPLES**

5487 None.

5488 **APPLICATION USAGE**

5489 None.

5490 **RATIONALE**

5491 None.

5492 **FUTURE DIRECTIONS**

5493 None.

5494 **SEE ALSO**5495 *ccosh()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>5496 **CHANGE HISTORY**

5497 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5498 **NAME**

5499 calloc — a memory allocator

5500 **SYNOPSIS**

5501 #include <stdlib.h>

5502 void *calloc(size_t *nelem*, size_t *elsize*);5503 **DESCRIPTION**5504 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5505 conflict between the requirements described here and the ISO C standard is unintentional. This
5506 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.5507 The *calloc()* function allocates unused space for an array of *nelem* elements each of whose size in
5508 bytes is *elsize*. The space is initialized to all bits 0.5509 The order and contiguity of storage allocated by successive calls to *calloc()* is unspecified. The
5510 pointer returned if the allocation succeeds is suitably aligned so that it may be assigned to a
5511 pointer to any type of object and then used to access such an object or an array of such objects in
5512 the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall
5513 yield a pointer to an object disjoint from any other object. The pointer returned points to the start
5514 (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer is
5515 returned. If the size of the space requested is 0, the behavior is implementation-defined; the
5516 value returned shall be either a null pointer or a unique pointer.5517 **RETURN VALUE**5518 Upon successful completion with both *nelem* and *elsize* non-zero, *calloc()* shall return a pointer to
5519 the allocated space. If either *nelem* or *elsize* is 0, then either a null pointer or a unique pointer
5520 value that can be successfully passed to *free()* shall be returned. Otherwise, it shall return a null
5521 **CX** pointer and set *errno* to indicate the error.5522 **ERRORS**5523 The *calloc()* function shall fail if:5524 **CX** [ENOMEM] Insufficient memory is available.5525 **EXAMPLES**

5526 None.

5527 **APPLICATION USAGE**

5528 There is now no requirement for the implementation to support the inclusion of <malloc.h>.

5529 **RATIONALE**

5530 None.

5531 **FUTURE DIRECTIONS**

5532 None.

5533 **SEE ALSO**5534 *free()*, *malloc()*, *realloc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>5535 **CHANGE HISTORY**

5536 First released in Issue 1. Derived from Issue 1 of the SVID.

5537 **Issue 4**5538 The setting of *errno* and the [ENOMEM] error are marked as extensions.5539 The APPLICATION USAGE section is changed to record that <malloc.h> need no longer be
5540 supported on XSI-conformant systems.

- 5541 The following changes are incorporated in this issue for alignment with the ISO C standard:
- 5542 • The DESCRIPTION is updated to indicate:
- 5543 — The order and contiguity of storage allocated by successive calls to this function is
5544 unspecified.
- 5545 — Each allocation yields a pointer to an object disjoint from any other object.
- 5546 — The returned pointer points to the lowest byte address of the allocation.
- 5547 — The behavior if space is requested of zero size.
- 5548 • The RETURN VALUE section is updated to indicate what is returned if either *nelem* or *elsize*
5549 is 0.

5550 Issue 6

5551 Extensions beyond the ISO C standard are now marked.

5552 The following new requirements on POSIX implementations derive from alignment with the
5553 Single UNIX Specification:

- 5554 • The setting of *errno* and the [ENOMEM] error condition are mandatory if an insufficient
5555 memory condition occurs.

5556 **NAME**

5557 carg, cargf, cargl — complex argument functions

5558 **SYNOPSIS**

5559 #include <complex.h>

5560 double carg(double complex *z*);5561 float cargf(float complex *z*);5562 long double cargl(long double complex *z*);5563 **DESCRIPTION**

5564 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5565 conflict between the requirements described here and the ISO C standard is unintentional. This
5566 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

5567 These functions shall compute the argument (also called phase angle) of *z*, with a branch cut
5568 along the negative real axis.

5569 **RETURN VALUE**5570 These functions shall return the value of the argument in the interval $[-\pi, +\pi]$.5571 **ERRORS**

5572 No errors are defined.

5573 **EXAMPLES**

5574 None.

5575 **APPLICATION USAGE**

5576 None.

5577 **RATIONALE**

5578 None.

5579 **FUTURE DIRECTIONS**

5580 None.

5581 **SEE ALSO**5582 *cimag()*, *conj()*, *cproj()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>5583 **CHANGE HISTORY**

5584 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5585 **NAME**

5586 casin, casinf, casinl — complex arc sine functions

5587 **SYNOPSIS**

5588 #include <complex.h>

5589 double complex casin(double complex *z*);5590 float complex casinf(float complex *z*);5591 long double complex casinl(long double complex *z*);5592 **DESCRIPTION**5593 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5594 conflict between the requirements described here and the ISO C standard is unintentional. This
5595 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.5596 These functions shall compute the complex arc sine of *z*, with branch cuts outside the interval
5597 $[-1, +1]$ along the real axis.5598 **RETURN VALUE**5599 These functions shall return the complex arc sine value, in the range of a strip mathematically
5600 unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the real axis.5601 **ERRORS**

5602 No errors are defined.

5603 **EXAMPLES**

5604 None.

5605 **APPLICATION USAGE**

5606 None.

5607 **RATIONALE**

5608 None.

5609 **FUTURE DIRECTIONS**

5610 None.

5611 **SEE ALSO**5612 *csin()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>5613 **CHANGE HISTORY**

5614 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5615 **NAME**

5616 casinh, casinhf, casinhl — complex arc hyperbolic sine functions

5617 **SYNOPSIS**

5618 #include <complex.h>

5619 double complex casinh(double complex *z*);5620 float complex casinhf(float complex *z*);5621 long double complex casinhl(long double complex *z*);5622 **DESCRIPTION**5623 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5624 conflict between the requirements described here and the ISO C standard is unintentional. This
5625 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.5626 These functions shall compute the complex arc hyperbolic sine of *z*, with branch cuts outside the
5627 interval $[-i, +i]$ along the imaginary axis.5628 **RETURN VALUE**5629 These functions shall return the complex arc hyperbolic sine value, in the range of a strip
5630 mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the
5631 imaginary axis.5632 **ERRORS**

5633 No errors are defined.

5634 **EXAMPLES**

5635 None.

5636 **APPLICATION USAGE**

5637 None.

5638 **RATIONALE**

5639 None.

5640 **FUTURE DIRECTIONS**

5641 None.

5642 **SEE ALSO**5643 `csinh()`, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>5644 **CHANGE HISTORY**

5645 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5646 **NAME**

5647 catan, catanf, catanl — complex arc tangent functions

5648 **SYNOPSIS**

5649 #include <complex.h>

5650 double complex catan(double complex *z*);5651 float complex catanf(float complex *z*);5652 long double complex catanl(long double complex *z*);5653 **DESCRIPTION**

5654 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5655 conflict between the requirements described here and the ISO C standard is unintentional. This
5656 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

5657 These functions shall compute the complex arc tangent of *z*, with branch cuts outside the
5658 interval $[-i, +i]$ along the imaginary axis.

5659 **RETURN VALUE**

5660 These functions shall return the complex arc tangent value, in the range of a strip
5661 mathematically unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the
5662 real axis.

5663 **ERRORS**

5664 No errors are defined.

5665 **EXAMPLES**

5666 None.

5667 **APPLICATION USAGE**

5668 None.

5669 **RATIONALE**

5670 None.

5671 **FUTURE DIRECTIONS**

5672 None.

5673 **SEE ALSO**5674 *ctan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>5675 **CHANGE HISTORY**

5676 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5677 **NAME**

5678 catanh, catanhf, catanhl — complex arc hyperbolic tangent functions

5679 **SYNOPSIS**

5680 #include <complex.h>

5681 double complex catanh(double complex *z*);5682 float complex catanhf(float complex *z*);5683 long double complex catanhl(long double complex *z*);5684 **DESCRIPTION**

5685 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5686 conflict between the requirements described here and the ISO C standard is unintentional. This
5687 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

5688 These functions shall compute the complex arc hyperbolic tangent of *z*, with branch cuts outside
5689 the interval $[-1, +1]$ along the real axis.

5690 **RETURN VALUE**

5691 These functions shall return the complex arc hyperbolic tangent value, in the range of a strip
5692 mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the
5693 imaginary axis.

5694 **ERRORS**

5695 No errors are defined.

5696 **EXAMPLES**

5697 None.

5698 **APPLICATION USAGE**

5699 None.

5700 **RATIONALE**

5701 None.

5702 **FUTURE DIRECTIONS**

5703 None.

5704 **SEE ALSO**5705 *ctanh()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>5706 **CHANGE HISTORY**

5707 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5708 **NAME**

5709 catclose — close a message catalog descriptor

5710 **SYNOPSIS**

5711 xSI #include <nl_types.h>

5712 int catclose(nl_catd catd);

5713

5714 **DESCRIPTION**5715 The *catclose()* function shall close the message catalog identified by *catd*. If a file descriptor is
5716 used to implement the type **nl_catd**, that file descriptor shall be closed.5717 **RETURN VALUE**5718 Upon successful completion, *catclose()* shall return 0; otherwise, -1 shall be returned, and *errno*
5719 set to indicate the error.5720 **ERRORS**5721 The *catclose()* function may fail if:

5722 [EBADF] The catalog descriptor is not valid. |

5723 [EINTR] The *catclose()* function was interrupted by a signal. |5724 **EXAMPLES**

5725 None.

5726 **APPLICATION USAGE**

5727 None.

5728 **RATIONALE**

5729 None.

5730 **FUTURE DIRECTIONS**

5731 None.

5732 **SEE ALSO**5733 *catgets()*, *catopen()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <nl_types.h> |5734 **CHANGE HISTORY**

5735 First released in Issue 2.

5736 **Issue 4**

5737 The [EBADF] and [EINTR] errors are added to the ERRORS section.

5738 **NAME**5739 `catgets` — read a program message5740 **SYNOPSIS**5741 XSI `#include <nl_types.h>`5742 `char *catgets(nl_catd catd, int set_id, int msg_id, const char *s);`

5743

5744 **DESCRIPTION**

5745 The `catgets()` function attempts to read message `msg_id`, in set `set_id`, from the message catalog
 5746 identified by `catd`. The `catd` argument is a message catalog descriptor returned from an earlier
 5747 call to `catopen()`. The `s` argument points to a default message string which shall be returned by
 5748 `catgets()` if it cannot retrieve the identified message.

5749 The `catgets()` function need not be reentrant. A function that is not required to be reentrant is not
 5750 required to be thread-safe.

5751 **RETURN VALUE**

5752 If the identified message is retrieved successfully, `catgets()` shall return a pointer to an internal
 5753 buffer area containing the null-terminated message string. If the call is unsuccessful for any
 5754 reason, `s` shall be returned and `errno` may be set to indicate the error.

5755 **ERRORS**5756 The `catgets()` function may fail if:

5757	[EBADF]	The <code>catd</code> argument is not a valid message catalog descriptor open for reading.	
------	---------	--	--

5758	[EINTR]	The read operation was terminated due to the receipt of a signal, and no data	
5759		was transferred.	

5760	[EINVAL]	The message catalog identified by <code>catd</code> is corrupted.	
------	----------	---	--

5761	[ENOMSG]	The message identified by <code>set_id</code> and <code>msg_id</code> is not in the message catalog.	
------	----------	--	--

5762 **EXAMPLES**

5763 None.

5764 **APPLICATION USAGE**

5765 None.

5766 **RATIONALE**

5767 None.

5768 **FUTURE DIRECTIONS**

5769 None.

5770 **SEE ALSO**5771 `catclose()`, `catopen()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<nl_types.h>`5772 **CHANGE HISTORY**

5773 First released in Issue 2.

5774 **Issue 4**5775 The type of argument `s` is changed from `char*` to `const char*`.

5776 The [EBADF] and [EINTR] errors are added to the ERRORS section.

5777 **Issue 4, Version 2**

5778 The following changes are incorporated for X/OPEN UNIX conformance:

- 5779 • The RETURN VALUE section notes that
- errno*
- may be set to indicate an error.
-
- 5780 • In the ERRORS section, [EINVAL] and [ENOMSG] are added as optional errors.

5781 **Issue 5**

5782 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

5783 **Issue 6**

5784 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

5785 **NAME**5786 `catopen` — open a message catalog5787 **SYNOPSIS**5788 xSI `#include <nl_types.h>`5789 `nl_catd catopen(const char *name, int oflag);`

5790

5791 **DESCRIPTION**

5792 The `catopen()` function shall open a message catalog and return a message catalog descriptor.
 5793 The `name` argument specifies the name of the message catalog to be opened. If `name` contains a
 5794 `'/'`, then `name` specifies a complete name for the message catalog. Otherwise, the environment
 5795 variable `NLSPATH` is used with `name` substituted for `%N` (see the Base Definitions volume of
 5796 IEEE Std. 1003.1-200x, Chapter 8, Environment Variables). If `NLSPATH` does not exist in the
 5797 environment, or if a message catalog cannot be found in any of the components specified by
 5798 `NLSPATH`, then an implementation-defined default path is used. This default may be affected by
 5799 the setting of `LC_MESSAGES` if the value of `oflag` is `NL_CAT_LOCALE`, or the `LANG`
 5800 environment variable if `oflag` is 0.

5801 A message catalog descriptor remains valid in a process until that process closes it, or a
 5802 successful call to one of the `exec` functions. A change in the setting of the `LC_MESSAGES`
 5803 category may invalidate existing open catalogs.

5804 If a file descriptor is used to implement message catalog descriptors, the `FD_CLOEXEC` flag
 5805 shall be set; see `<fcntl.h>`.

5806 If the value of the `oflag` argument is 0, the `LANG` environment variable is used to locate the
 5807 catalog without regard to the `LC_MESSAGES` category. If the `oflag` argument is
 5808 `NL_CAT_LOCALE`, the `LC_MESSAGES` category is used to locate the message catalog (see the
 5809 Base Definitions volume of IEEE Std. 1003.1-200x, Section 8.2, Internationalization Variables).

5810 **RETURN VALUE**

5811 Upon successful completion, `catopen()` shall return a message catalog descriptor for use on
 5812 subsequent calls to `catgets()` and `catclose()`. Otherwise, `catopen()` shall return `(nl_catd) -1` and set
 5813 `errno` to indicate the error.

5814 **ERRORS**5815 The `catopen()` function may fail if:

5816 [EACCES] Search permission is denied for the component of the path prefix of the
 5817 message catalog or read permission is denied for the message catalog.

5818 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

5819 [ENAMETOOLONG]

5820 The length of a path name of the message catalog exceeds {PATH_MAX} or a
 5821 path name component is longer than {NAME_MAX}.

5822 [ENAMETOOLONG]

5823 Path name resolution of a symbolic link produced an intermediate result
 5824 whose length exceeds {PATH_MAX}.

5825 [ENFILE] Too many files are currently open in the system.

5826 [ENOENT] The message catalog does not exist or the `name` argument points to an empty
 5827 string.

5828 [ENOMEM] Insufficient storage space is available.

- 5829 [ENOTDIR] A component of the path prefix of the message catalog is not a directory.
- 5830 **EXAMPLES**
- 5831 None.
- 5832 **APPLICATION USAGE**
- 5833 Some implementations of *catopen()* use *malloc()* to allocate space for internal buffer areas. The
- 5834 *catopen()* function may fail if there is insufficient storage space available to accommodate these
- 5835 buffers.
- 5836 Portable applications must assume that message catalog descriptors are not valid after a call to
- 5837 one of the *exec* functions.
- 5838 Application writers should be aware that guidelines for the location of message catalogs have
- 5839 not yet been developed. Therefore they should take care to avoid conflicting with catalogs used
- 5840 by other applications and the standard utilities.
- 5841 **RATIONALE**
- 5842 None.
- 5843 **FUTURE DIRECTIONS**
- 5844 None.
- 5845 **SEE ALSO**
- 5846 *catclose()*, *catgets()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<fcntl.h>`,
- 5847 `<nl_types.h>`, the Shell and Utilities volume of IEEE Std. 1003.1-200x
- 5848 **CHANGE HISTORY**
- 5849 First released in Issue 2.
- 5850 **Issue 4**
- 5851 The type of argument *name* is changed from **char*** to **const char***.
- 5852 The DESCRIPTION is updated:
- 5853
- To indicate the longevity of message catalog descriptors.
 - To specify values for the *oflag* argument and the effect of *LC_MESSAGES* and *NLSPATH*.
- 5854
- 5855 The [EACCES], [EMFILE], [ENAMETOOLONG], [ENFILE], [ENOENT], and [ENOTDIR] errors
- 5856 are added to the ERRORS section.
- 5857 The APPLICATION USAGE section is updated to indicate:
- 5858
- Portable applications should not assume the continued validity of message catalog
 - descriptors after a call to one of the *exec* functions.
- 5859
- 5860
- Message catalogs must be located with care.
- 5861 **Issue 4, Version 2**
- 5862 The following change is incorporated for X/OPEN UNIX conformance:
- 5863
- In the ERRORS section, an [ENAMETOOLONG] condition is defined that may report
 - excessive length of an intermediate result of path name resolution of a symbolic link.
- 5864

5865 **NAME**

5866 cbrt, cbrtf, cbrtl — cube root functions

5867 **SYNOPSIS**

5868 #include <math.h>

5869 double cbrt(double x);

5870 float cbrtf(float x);

5871 long double cbrtl(long double x);

5872 **DESCRIPTION**

5873 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 5874 conflict between the requirements described here and the ISO C standard is unintentional. This
 5875 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

5876 These functions shall compute the real cube root of *x*.

5877 An application wishing to check for error situations should set *errno* to 0 before calling these
 5878 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

5879 **RETURN VALUE**5880 Upon successful completion, these functions shall return the cube root of *x*.5881 If *x* is $\pm\text{Inf}$, these functions shall return *x*.5882 If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].5883 **ERRORS**

5884 These functions may fail if:

5885 [EDOM] The value of *x* is NaN.5886 **EXAMPLES**

5887 None.

5888 **APPLICATION USAGE**

5889 None.

5890 **RATIONALE**

5891 For some applications, a true cube root function, which returns negative results for negative
 5892 arguments, is more appropriate than *pow(x, 1.0/3.0)*, which returns a NaN for *x* less than 0.

5893 **FUTURE DIRECTIONS**

5894 None.

5895 **SEE ALSO**

5896 The Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

5897 **CHANGE HISTORY**

5898 First released in Issue 4, Version 2.

5899 **Issue 5**

5900 Moved from X/OPEN UNIX extension to BASE.

5901 **Issue 6**5902 The *cbrt()* function is no longer marked XSI.5903 The *cbrtf()* and *cbrtl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

5904 **NAME**

5905 ccos, ccosf, ccosl — complex cosine functions

5906 **SYNOPSIS**

5907 #include <complex.h>

5908 double complex ccos(double complex *z*);5909 float complex ccosf(float complex *z*);5910 long double complex ccosl(long double complex *z*);5911 **DESCRIPTION**5912 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5913 conflict between the requirements described here and the ISO C standard is unintentional. This
5914 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.5915 These functions shall compute the complex cosine of *z*.5916 **RETURN VALUE**

5917 These functions shall return the complex cosine value.

5918 **ERRORS**

5919 No errors are defined.

5920 **EXAMPLES**

5921 None.

5922 **APPLICATION USAGE**

5923 None.

5924 **RATIONALE**

5925 None.

5926 **FUTURE DIRECTIONS**

5927 None.

5928 **SEE ALSO**5929 *cacos()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>5930 **CHANGE HISTORY**

5931 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5932 **NAME**

5933 ccosh, ccoshf, ccoshl — complex hyperbolic cosine functions

5934 **SYNOPSIS**

5935 #include <complex.h>

5936 double complex ccosh(double complex *z*);5937 float complex ccoshf(float complex *z*);5938 long double complex ccoshl(long double complex *z*);5939 **DESCRIPTION**

5940 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5941 conflict between the requirements described here and the ISO C standard is unintentional. This
5942 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

5943 These functions shall compute the complex hyperbolic cosine of *z*.5944 **RETURN VALUE**

5945 These functions shall return the complex hyperbolic cosine value.

5946 **ERRORS**

5947 No errors are defined.

5948 **EXAMPLES**

5949 None.

5950 **APPLICATION USAGE**

5951 None.

5952 **RATIONALE**

5953 None.

5954 **FUTURE DIRECTIONS**

5955 None.

5956 **SEE ALSO**5957 *cacosh()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>5958 **CHANGE HISTORY**

5959 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5960 **NAME**

5961 ceil, ceilf, ceill — ceiling value function

5962 **SYNOPSIS**

5963 #include <math.h>

5964 double ceil(double x);

5965 float ceilf(float x);

5966 long double ceill(long double x);

5967 **DESCRIPTION**5968 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5969 conflict between the requirements described here and the ISO C standard is unintentional. This
5970 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.5971 The *ceil()*, *ceilf()*, and *ceill()* functions shall compute the smallest integral value not less than *x*.5972 An application wishing to check for error situations should set *errno* to 0 before calling *ceil()*. If
5973 *errno* is non-zero on return, or the return value is NaN, an error has occurred.5974 **RETURN VALUE**5975 Upon successful completion, the *ceil()*, *ceilf()*, and *ceill()* functions shall return the smallest
5976 integral value not less than *x*, expressed as a type **double**.5977 **XSI** If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].5978 If the correct value would cause overflow, HUGE_VAL shall be returned and *errno* set to
5979 **XSI** [ERANGE]. If *x* is $\pm\text{Inf}$ or ± 0 , the value of *x* shall be returned.5980 **ERRORS**5981 The *ceil()*, *ceilf()*, and *ceill()* functions shall fail if:

5982 [ERANGE] The result overflows.

5983 The *ceil()*, *ceilf()*, and *ceill()* functions may fail if:5984 **XSI** [EDOM] The value of *x* is NaN.5985 **XSI** No other errors shall occur.5986 **EXAMPLES**

5987 None.

5988 **APPLICATION USAGE**5989 The integral value returned by *ceil()* as a **double** need not be expressible as an **int** or **long**. The
5990 return value should be tested before assigning it to an integer type to avoid the undefined results
5991 of an integer overflow.5992 The *ceil()* function can only overflow when the floating point representation has
5993 DBL_MANT_DIG > DBL_MAX_EXP.5994 **RATIONALE**

5995 None.

5996 **FUTURE DIRECTIONS**

5997 None.

5998 **SEE ALSO**5999 *floor()*, *isnan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

6000 **CHANGE HISTORY**

6001 First released in Issue 1. Derived from Issue 1 of the SVID.

6002 **Issue 4**

6003 References to *matherr()* are removed.

6004 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the
6005 ISO C standard and to rationalize error handling in the mathematics functions.

6006 The return value specified for [EDOM] is marked as an extension.

6007 Support for x being $\pm\text{Inf}$ or ± 0 is added to the RETURN VALUE section and marked as an
6008 extension.

6009 **Issue 5**

6010 The DESCRIPTION is updated to indicate how an application should check for an error. This
6011 text was previously published in the APPLICATION USAGE section.

6012 **Issue 6**

6013 The *ceilf()* and *ceilll()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

6014 NAME

6015 `cexp`, `cexpf`, `cexpl` — complex exponential functions

6016 SYNOPSIS

6017 `#include <complex.h>`

6018 `double complex cexp(double complex z);`

6019 `float complex cexpf(float complex z);`

6020 `long double complex cexpl(long double complex z);`

6021 DESCRIPTION

6022 `CX` The functionality described on this reference page is aligned with the ISO C standard. Any
6023 conflict between the requirements described here and the ISO C standard is unintentional. This
6024 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

6025 These functions shall compute the complex exponent of z , defined as e^z .

6026 RETURN VALUE

6027 These functions shall return the complex exponential value of z .

6028 ERRORS

6029 No errors are defined.

6030 EXAMPLES

6031 None.

6032 APPLICATION USAGE

6033 None.

6034 RATIONALE

6035 None.

6036 FUTURE DIRECTIONS

6037 None.

6038 SEE ALSO

6039 `clog()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<complex.h>`

6040 CHANGE HISTORY

6041 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

6042 **NAME**

6043 cfgetispeed — get input baud rate

6044 **SYNOPSIS**

6045 #include <termios.h>

6046 speed_t cfgetispeed(const struct termios *termios_p);

6047 **DESCRIPTION**6048 The *cfgetispeed()* function shall extract the input baud rate from the **termios** structure to which
6049 the *termios_p* argument points.6050 This function shall return exactly the value in the **termios** data structure, without interpretation.6051 **RETURN VALUE**6052 Upon successful completion, *cfgetispeed()* shall return a value of type **speed_t** representing the
6053 input baud rate.6054 **ERRORS**

6055 No errors are defined.

6056 **EXAMPLES**

6057 None.

6058 **APPLICATION USAGE**

6059 None.

6060 **RATIONALE**6061 The term *baud* is used historically here, but is not technically correct. This is properly “bits per
6062 second”, which may not be the same as baud. However, the term is used because of the
6063 historical usage and understanding.6064 The *cfgetospeed()*, *cfgetispeed()*, *cfsetospeed()*, and *cfsetispeed()* functions do not take arguments as
6065 numbers, but rather as symbolic names. There are two reasons for this:

- 6066 1. Historically, numbers were not used because of the way the rate was stored in the data
-
- 6067 structure. This is retained even though a function is now used.
-
- 6068 2. More importantly, only a limited set of possible rates is at all portable, and this constrains
-
- 6069 the application to that set.

6070 There is nothing to prevent an implementation to accept, as an extension, a number (such as 126)
6071 if it wished, and because the encoding of the Bxxx symbols is not specified, this can be done so
6072 no ambiguity is introduced.6073 Setting the input baud rate to zero was a mechanism to allow for split baud rates. Clarifications
6074 in this volume of IEEE Std. 1003.1-200x have made it possible to determine whether split rates
6075 are supported and to support them without having to treat zero as a special case. Since this
6076 functionality is also confusing, it has been declared obsolescent. The 0 argument referred to is
6077 the literal constant 0, not the symbolic constant B0. This volume of IEEE Std. 1003.1-200x does
6078 not preclude B0 from being defined as the value 0; in fact, implementations would likely benefit
6079 from the two being equivalent. This volume of IEEE Std. 1003.1-200x does not fully specify
6080 whether the previous *cfsetispeed()* value is retained after a *tcgetattr()* as the actual value or as
6081 zero. Therefore, portable applications should always set both the input speed and output speed
6082 when setting either.6083 In historical implementations, the baud rate information is traditionally kept in **c_cflag**.
6084 Applications should be written to presume that this might be the case (and thus not blindly copy
6085 **c_cflag**), but not to rely on it in case it is in some other field of the structure. Setting the **c_cflag**
6086 field absolutely after setting a baud rate is a non-portable action because of this. In general, the

6087 unused parts of the flag fields might be used by the implementation and should not be blindly
6088 copied from the descriptions of one terminal device to another.

6089 **FUTURE DIRECTIONS**

6090 None.

6091 **SEE ALSO**

6092 *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*, the Base Definitions volume of
6093 IEEE Std. 1003.1-200x, `<termios.h>`, the Base Definitions volume of IEEE Std. 1003.1-200x,
6094 Chapter 11, General Terminal Interface

6095 **CHANGE HISTORY**

6096 First released in Issue 3.

6097 Entry included for alignment with the POSIX.1-1988 standard.

6098 **Issue 4**

6099 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 6100
- The type of the argument *termios_p* is changed from **struct termios*** to **const struct termios***.
 - The DESCRIPTION is changed to indicate that the function simply returns the value from *termios_p*, irrespective of how that structure was obtained. Issue 3 states that if *termios_p* was not obtained by a successful call to *tcgetattr()*, the behavior is undefined.
- 6101
6102
6103

6104 **NAME**

6105 cfgetospeed — get output baud rate

6106 **SYNOPSIS**

6107 #include <termios.h>

6108 speed_t cfgetospeed(const struct termios *termios_p);

6109 **DESCRIPTION**

6110 The *cfgetospeed()* function shall extract the output baud rate from the **termios** structure to which
6111 the *termios_p* argument points.

6112 This function shall return exactly the value in the **termios** data structure, without interpretation.

6113 **RETURN VALUE**

6114 Upon successful completion, *cfgetospeed()* shall return a value of type **speed_t** representing the
6115 output baud rate.

6116 **ERRORS**

6117 No errors are defined.

6118 **EXAMPLES**

6119 None.

6120 **APPLICATION USAGE**

6121 None.

6122 **RATIONALE**

6123 Refer to *cfgetispeed()*.

6124 **FUTURE DIRECTIONS**

6125 None.

6126 **SEE ALSO**

6127 *cfgetispeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*, the Base Definitions volume of
6128 IEEE Std. 1003.1-200x, <**termios.h**>, the Base Definitions volume of IEEE Std. 1003.1-200x,
6129 Chapter 11, General Terminal Interface

6130 **CHANGE HISTORY**

6131 First released in Issue 3.

6132 Entry included for alignment with the POSIX.1-1988 standard.

6133 **Issue 4**

6134 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 6135 • The type of the argument *termios_p* is changed from **struct termios*** to **const struct termios***.
- 6136 • The DESCRIPTION is changed to indicate that the function simply returns the value from
6137 *termios_p*, irrespective of how that structure was obtained. Issue 3 states that if *termios_p* was
6138 not obtained by a successful call to *tcgetattr()*, the behavior is undefined.

6139 **NAME**

6140 cfsetispeed — set input baud rate

6141 **SYNOPSIS**

6142 #include <termios.h>

6143 int cfsetispeed(struct termios *termios_p, speed_t speed);

6144 **DESCRIPTION**

6145 The *cfsetispeed()* function shall set the input baud rate stored in the structure pointed to by
6146 *termios_p* to *speed*.

6147 There shall be no effect on the baud rates set in the hardware until a subsequent successful call
6148 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set
6149 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*
6150 function is called.

6151 **RETURN VALUE**

6152 Upon successful completion, *cfsetispeed()* shall return 0; otherwise, -1 shall be returned, and
6153 *errno* may be set to indicate the error.

6154 **ERRORS**

6155 The *cfsetispeed()* function may fail if:

6156 [EINVAL] The *speed* value is not a valid baud rate.

6157 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in
6158 <termios.h>.

6159 **EXAMPLES**

6160 None.

6161 **APPLICATION USAGE**

6162 None.

6163 **RATIONALE**

6164 Refer to *cfgetispeed()*.

6165 **FUTURE DIRECTIONS**

6166 None.

6167 **SEE ALSO**

6168 *cfgetispeed()*, *cfgetospeed()*, *cfsetospeed()*, *tcsetattr()*, the Base Definitions volume of
6169 IEEE Std. 1003.1-200x, <termios.h>, the Base Definitions volume of IEEE Std. 1003.1-200x,
6170 Chapter 11, General Terminal Interface

6171 **CHANGE HISTORY**

6172 First released in Issue 3.

6173 Entry included for alignment with the POSIX.1-1988 standard.

6174 **Issue 4**

6175 The first description of the [EINVAL] error is added and is marked as an extension.

6176 **Issue 4, Version 2**

6177 The ERRORS section is changed to indicate that [EINVAL] may be returned if the specified
6178 speed is outside the range of possible speed values given in <termios.h>.

6179 **Issue 6**

6180 The following new requirements on POSIX implementations derive from alignment with the
6181 Single UNIX Specification:

- 6182 • The optional setting of *errno* and the [EINVAL] error conditions are added.

6183 **NAME**

6184 cfsetospeed — set output baud rate

6185 **SYNOPSIS**

6186 #include <termios.h>

6187 int cfsetospeed(struct termios *termios_p, speed_t speed);

6188 **DESCRIPTION**6189 The *cfsetospeed()* function shall set the output baud rate stored in the structure pointed to by
6190 *termios_p* to *speed*.6191 There shall be no effect on the baud rates set in the hardware until a subsequent successful call
6192 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set
6193 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*
6194 function is called.6195 **RETURN VALUE**6196 Upon successful completion, *cfsetospeed()* shall return 0; otherwise, it shall return -1 and *errno*
6197 may be set to indicate the error.6198 **ERRORS**6199 The *cfsetospeed()* function may fail if:6200 [EINVAL] The *speed* value is not a valid baud rate.6201 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in
6202 <termios.h>.6203 **EXAMPLES**

6204 None.

6205 **APPLICATION USAGE**

6206 None.

6207 **RATIONALE**6208 Refer to *cfgetispeed()*.6209 **FUTURE DIRECTIONS**

6210 None.

6211 **SEE ALSO**6212 *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *tcsetattr()*, the Base Definitions volume of
6213 IEEE Std. 1003.1-200x, <termios.h>, the Base Definitions volume of IEEE Std. 1003.1-200x,
6214 Chapter 11, General Terminal Interface6215 **CHANGE HISTORY**

6216 First released in Issue 3.

6217 Entry included for alignment with the POSIX.1-1988 standard.

6218 **Issue 4**

6219 The first description of the [EINVAL] error is added and is marked as an extension.

6220 **Issue 4, Version 2**6221 The ERRORS section is changed to indicate that [EINVAL] may be returned if the specified
6222 speed is outside the range of possible speed values given in <termios.h>.

6223 **Issue 6**

6224 The following new requirements on POSIX implementations derive from alignment with the
6225 Single UNIX Specification:

- 6226 • The optional setting of *errno* and the [EINVAL] error conditions are added.

6227 **NAME**

6228 chdir — change working directory

6229 **SYNOPSIS**

6230 #include <unistd.h>

6231 int chdir(const char *path);

6232 **DESCRIPTION**

6233 The *chdir()* function shall cause the directory named by the path name pointed to by the *path*
 6234 argument to become the current working directory; that is, the starting point for path searches
 6235 for path names not beginning with '/ '.

6236 **RETURN VALUE**

6237 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, the current
 6238 working directory shall remain unchanged, and *errno* shall be set to indicate the error.

6239 **ERRORS**6240 The *chdir()* function shall fail if:

6241 [EACCES] Search permission is denied for any component of the path name.

6242 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
6243 argument.

6244 [ENAMETOOLONG]

6245 The length of the *path* argument exceeds {PATH_MAX} or a path name
6246 component is longer than {NAME_MAX}.6247 [ENOENT] A component of *path* does not name an existing directory or *path* is an empty
6248 string.

6249 [ENOTDIR] A component of the path name is not a directory.

6250 The *chdir()* function may fail if:6251 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
6252 resolution of the *path* argument.

6253 [ENAMETOOLONG]

6254 As a result of encountering a symbolic link in resolution of the *path* argument,
6255 the length of the substituted path name string exceeded {PATH_MAX}.6256 **EXAMPLES**6257 **Changing the Current Working Directory**6258 The following example makes the value pointed to by **directory**, **/tmp**, the current working
6259 directory.

6260 #include <unistd.h>

6261 ...

6262 char *directory = "/tmp";

6263 int ret;

6264 ret = chdir (directory);

6265 **APPLICATION USAGE**

6266 The *chdir()* function only affects the working directory of the current process. The result if a
 6267 NULL argument is passed to *chdir()* is unspecified because some implementations dynamically
 6268 allocate space in that case.

6269 **RATIONALE**

6270 The *chdir()* function only affects the working directory of the current process.

6271 The result if a NULL argument is passed to *chdir()* is left implementation-defined because some
 6272 implementations dynamically allocate space in that case.

6273 **FUTURE DIRECTIONS**

6274 None.

6275 **SEE ALSO**

6276 *getcwd()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>

6277 **CHANGE HISTORY**

6278 First released in Issue 1. Derived from Issue 1 of the SVID.

6279 **Issue 4**

6280 The <**unistd.h**> header is added to the SYNOPSIS section.

6281 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 6282 • The type of argument *path* is changed from **char*** to **const char***.

6283 The following change is incorporated for alignment with the FIPS requirements:

- 6284 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path
 6285 name component is larger than {NAME_MAX} is now defined as mandatory and marked as
 6286 an extension.

6287 **Issue 4, Version 2**

6288 The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- 6289 • It states that [ELOOP] is returned if too many symbolic links are encountered during path
 6290 name resolution.
- 6291 • A second [ENAMETOOLONG] condition is defined that may report excessive length of an
 6292 intermediate result of path name resolution of a symbolic link.

6293 **Issue 6**

6294 The APPLICATION USAGE section is added.

6295 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 6296 • The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.
 6297 This is since behavior may vary from one file system to another.

6298 The following new requirements on POSIX implementations derive from alignment with the
 6299 Single UNIX Specification:

- 6300 • The [ELOOP] mandatory error condition is added.
- 6301 • A second [ENAMETOOLONG] is added as an optional error condition.

6302 The following changes were made to align with the IEEE P1003.1a draft standard:

- 6303 • The [ELOOP] optional error condition is added.

6304 **NAME**

6305 chmod — change mode of a file

6306 **SYNOPSIS**

6307 #include <sys/stat.h>

6308 int chmod(const char *path, mode_t mode);

6309 **DESCRIPTION**

6310 XSI The *chmod()* function shall change S_ISUID, S_ISGID, S_ISVTX, and the file permission bits of
 6311 the file named by the path name pointed to by the *path* argument to the corresponding bits in the
 6312 *mode* argument. The application shall ensure that the effective user ID of the process matches the
 6313 owner of the file or the process has appropriate privileges in order to do this.

6314 S_ISUID, S_ISGID, and the file permission bits are described in <sys/stat.h>.

6315 XSI If a directory is writable and the mode bit S_ISVTX is set on the directory, a process may remove
 6316 or rename files within that directory only if one or more of the following is true:

- 6317 • The effective user ID of the process is the same as that of the owner ID of the file.
- 6318 • The effective user ID of the process is the same as that of the owner ID of the directory.
- 6319 • The process has appropriate privileges.

6320

6321 XSI If the S_ISVTX bit is set on a non-directory file, the behavior is unspecified.

6322 If the calling process does not have appropriate privileges, and if the group ID of the file does
 6323 not match the effective group ID or one of the supplementary group IDs and if the file is a
 6324 regular file, bit S_ISGID (set-group-ID on execution) in the file's mode shall be cleared upon
 6325 successful return from *chmod()*.

6326 Additional implementation-defined restrictions may cause the S_ISUID and S_ISGID bits in
 6327 *mode* to be ignored.

6328 The effect on file descriptors for files open at the time of a call to *chmod()* is implementation-
 6329 defined.

6330 Upon successful completion, *chmod()* shall mark for update the *st_ctime* field of the file.6331 **RETURN VALUE**

6332 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 6333 indicate the error. If -1 is returned, no change to the file mode occurs.

6334 **ERRORS**6335 The *chmod()* function shall fail if:

- | | | |
|------|----------------|--|
| 6336 | [EACCES] | Search permission is denied on a component of the path prefix. |
| 6337 | [ELOOP] | A loop exists in symbolic links encountered during resolution of the <i>path</i> |
| 6338 | | argument. |
| 6339 | [ENAMETOOLONG] | |
| 6340 | | The length of the <i>path</i> argument exceeds {PATH_MAX} or a path name |
| 6341 | | component is longer than {NAME_MAX}. |
| 6342 | [ENOTDIR] | A component of the path prefix is not a directory. |
| 6343 | [ENOENT] | A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string. |
| 6344 | [EPERM] | The effective user ID does not match the owner of the file and the process |
| 6345 | | does not have appropriate privileges. |

6346	[EROFS]	The named file resides on a read-only file system.
6347		The <i>chmod()</i> function may fail if:
6348	[EINTR]	A signal was caught during execution of the function.
6349	[EINVAL]	The value of the <i>mode</i> argument is invalid.
6350	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
6351		resolution of the <i>path</i> argument.
6352	[ENAMETOOLONG]	
6353		As a result of encountering a symbolic link in resolution of the <i>path</i> argument,
6354		the length of the substituted path name strings exceeded {PATH_MAX}.

6355 EXAMPLES

6356 Setting Read Permissions for User, Group, and Others

6357 The following example sets read permissions for the owner, group, and others.

```
6358 #include <sys/stat.h>
6359 const char *path;
6360 ...
6361 chmod(path, S_IRUSR | S_IRGRP | S_IROTH);
```

6362 Setting Read, Write, and Execute Permissions for the Owner Only

6363 The following example sets read, write, and execute permissions for the owner, and no
6364 permissions for group and others.

```
6365 #include <sys/stat.h>
6366 const char *path;
6367 ...
6368 chmod(path, S_IRWXU);
```

6369 Setting Different Permissions for Owner, Group, and Other

6370 The following example sets owner permissions for *CHANGEFILE* to read, write, and execute,
6371 group permissions to read and execute, and other permissions to read.

```
6372 #include <sys/stat.h>
6373 #define CHANGEFILE "/etc/myfile"
6374 ...
6375 chmod(CHANGEFILE, S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH);
```

6376 Setting and Checking File Permissions

6377 The following example sets the file permission bits for a file named */home/cnd/mod1*, then calls
6378 the *stat()* function to verify the permissions.

```
6379 #include <sys/types.h>
6380 #include <sys/stat.h>
6381 int status;
6382 struct stat buffer
6383 ...
```

```
6384     chmod("home/cnd/mod1", S_IRWXU|S_IRWXG|S_IROTH|S_IWOTH);
6385     status = stat("home/cnd/mod1", &buffer);
```

6386 APPLICATION USAGE

6387 In order to ensure that the S_ISUID and S_ISGID bits are set, an application requiring this should
6388 use *stat()* after a successful *chmod()* to verify this.

6389 Any file descriptors currently open by any process on the file could possibly become invalid if
6390 the mode of the file is changed to a value which would deny access to that process. One
6391 situation where this could occur is on a stateless file system. This behavior will not occur in a
6392 conforming environment.

6393 RATIONALE

6394 This volume of IEEE Std. 1003.1-200x specifies that the S_ISGID bit is cleared by *chmod()* on a
6395 regular file under certain conditions. This is specified on the assumption that regular files may
6396 be executed, and the system should prevent users from making executable *setgid()* files perform
6397 with privileges that the caller does not have. On implementations that support execution of
6398 other file types, the S_ISGID bit should be cleared for those file types under the same
6399 circumstances.

6400 Implementations that use the S_ISUID bit to indicate some other function (for example,
6401 mandatory record locking) on non-executable files need not clear this bit on writing. They
6402 should clear the bit for executable files and any other cases where the bit grants special powers
6403 to processes that change the file contents. Similar comments apply to the S_ISGID bit.

6404 FUTURE DIRECTIONS

6405 None.

6406 SEE ALSO

6407 *chown()*, *mkdir()*, *mkfifo()*, *open()*, *stat()*, *statvfs()*, the Base Definitions volume of
6408 IEEE Std. 1003.1-200x, <sys/stat.h>, <sys/types.h>

6409 CHANGE HISTORY

6410 First released in Issue 1. Derived from Issue 1 of the SVID.

6411 Issue 4

6412 The <sys/types.h> header is now marked as optional (OH); this header need not be included on
6413 XSI-conformant systems.

6414 The [EINVAL] error is marked as an extension.

6415 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 6416 • The type of argument *path* is changed from **char*** to **const char***.

6417 The following change is incorporated for alignment with the FIPS requirements:

- 6418 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path
6419 name component is larger than {NAME_MAX} is now defined as mandatory and marked as
6420 an extension.

6421 Issue 4, Version 2

6422 The following changes are incorporated for X/OPEN UNIX conformance:

- 6423 • The DESCRIPTION is updated to describe X/OPEN UNIX functionality relating to
6424 permission checks applied when removing or renaming files in a directory having the
6425 S_ISVTX bit set.
- 6426 • In the ERRORS section, the condition whereby [ELOOP] is returned if too many symbolic
6427 links are encountered during path name resolution is defined as mandatory, and [EINTR] is

- 6428 added as an optional error.
- 6429 • In the ERRORS section, a second [ENAMETOOLONG] condition is defined that may report
6430 excessive length of an intermediate result of path name resolution of a symbolic link.
- 6431 **Issue 6**
- 6432 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:
- 6433 • The [ENAMETOOLONG] error is restored as an error dependent on _POSIX_NO_TRUNC.
6434 This is since behavior may vary from one file system to another.
- 6435 The following new requirements on POSIX implementations derive from alignment with the
6436 Single UNIX Specification:
- 6437 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
6438 required for conforming implementations of previous POSIX specifications, it was not
6439 required for UNIX applications.
- 6440 • The [EINVAL] and [EINTR] optional error conditions are added.
- 6441 • A second [ENAMETOOLONG] is added as an optional error condition.
- 6442 The following changes were made to align with the IEEE P1003.1a draft standard:
- 6443 • The [ELOOP] optional error condition is added.
- 6444 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

6445 **NAME**

6446 chown — change owner and group of a file

6447 **SYNOPSIS**

6448 #include <unistd.h>

6449 int chown(const char *path, uid_t owner, gid_t group);

6450 **DESCRIPTION**6451 The *path* argument points to a path name naming a file. The user ID and group ID of the named
6452 file are set to the numeric values contained in *owner* and *group*, respectively.6453 Only processes with an effective user ID equal to the user ID of the file or with appropriate
6454 privileges may change the ownership of a file. If `_POSIX_CHOWN_RESTRICTED` is in effect for
6455 *path*:

- 6456
- Changing the user ID is restricted to processes with appropriate privileges.
 - Changing the group ID is permitted to a process with an effective user ID equal to the user
6457 ID of the file, but without appropriate privileges, if and only if *owner* is equal to the file's user
6458 ID or `(uid_t)-1` and *group* is equal either to the calling process' effective group ID or to one of
6459 its supplementary group IDs.

6461 If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of
6462 the file mode are set, and the process does not have appropriate privileges, the set-user-ID
6463 (`S_ISUID`) and set-group-ID (`S_ISGID`) bits of the file mode shall be cleared upon successful
6464 return from *chown*(*path*). If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`,
6465 or `S_IXOTH` bits of the file mode are set, and the process has appropriate privileges, it is
6466 implementation-defined whether the set-user-ID and set-group-ID bits are altered. If the *chown*(*path*)
6467 function is successfully invoked on a file that is not a regular file and one or more of the
6468 `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of the file mode are set, the set-user-ID and set-group-ID
6469 bits may be cleared.6470 If *owner* or *group* is specified as `(uid_t)-1` or `(gid_t)-1`, respectively, the corresponding ID of the
6471 file is unchanged.6472 Upon successful completion, *chown*(*path*) shall mark for update the *st_ctime* field of the file.6473 **RETURN VALUE**6474 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
6475 indicate the error. If -1 is returned, no changes are made in the user ID and group ID of the file.6476 **ERRORS**6477 The *chown*(*path*) function shall fail if:

- 6478 [EACCES] Search permission is denied on a component of the path prefix.
-
- 6479 [ELOOP] A loop exists in symbolic links encountered during resolution of the
- path*
-
- 6480 argument.
-
- 6481 [ENAMETOOLONG] The length of the
- path*
- argument exceeds {PATH_MAX} or a path name
-
- 6482 component is longer than {NAME_MAX}.
-
- 6483 [ENOTDIR] A component of the path prefix is not a directory.
-
- 6484 [ENOENT] A component of
- path*
- does not name an existing file or
- path*
- is an empty string.
-
- 6485 [EPERM] The effective user ID does not match the owner of the file, or the calling
-
- 6486 process does not have appropriate privileges and
-
- 6487
- `_POSIX_CHOWN_RESTRICTED`
- indicates that such privilege is required.
-
- 6488

6489	[EROFS]	The named file resides on a read-only file system.
6490		The <i>chown()</i> function may fail if:
6491	[EIO]	An I/O error occurred while reading or writing to the file system.
6492	[EINTR]	The <i>chown()</i> function was interrupted by a signal which was caught.
6493	[EINVAL]	The owner or group ID supplied is not a value supported by the
6494		implementation.
6495	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
6496		resolution of the <i>path</i> argument.
6497	[ENAMETOOLONG]	
6498		As a result of encountering a symbolic link in resolution of the <i>path</i> argument,
6499		the length of the substituted path name string exceeded {PATH_MAX}.

6500 EXAMPLES

6501 None.

6502 APPLICATION USAGE

6503 Although *chown()* can be used on some systems by the file owner to change the owner and
 6504 group to any desired values, the only portable use of this function is to change the group of a file
 6505 to the effective GID of the calling process or to a member of its group set.

6506 RATIONALE

6507 System III and System V allow a user to give away files; that is, the owner of a file may change
 6508 its user ID to anything. This is a serious problem for implementations that are intended to meet
 6509 government security regulations. Version 7 and 4.3 BSD permit only the superuser to change the
 6510 user ID of a file. Some government agencies (usually not ones concerned directly with security)
 6511 find this limitation too confining. This volume of IEEE Std. 1003.1-200x uses *may* to permit
 6512 secure implementations while not disallowing System V.

6513 System III and System V allow the owner of a file to change the group ID to anything. Version 7
 6514 permits only the superuser to change the group ID of a file. 4.3 BSD permits the owner to
 6515 change the group ID of a file to its effective group ID or to any of the groups in the list of
 6516 supplementary group IDs, but to no others.

6517 The POSIX.1-1990 standard requires that the *chown()* function invoked by a non-appropriate
 6518 privileged process clear the *S_ISGID* and the *S_ISUID* bits for regular files, and permits them to
 6519 be cleared for other types of files. This is so that changes in accessibility do not accidentally
 6520 cause files to become security holes. Unfortunately, requiring these bits to be cleared on non-
 6521 executable data files also clears the mandatory file locking bit (shared with *S_ISGID*), which
 6522 is an extension on many implementations (it first appeared in System V). These bits should only
 6523 be required to be cleared on regular files that have one or more of their execute bits set.

6524 FUTURE DIRECTIONS

6525 None.

6526 SEE ALSO

6527 *chmod()*, *pathconf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<sys/types.h>`,
 6528 `<unistd.h>`

6529 CHANGE HISTORY

6530 First released in Issue 1. Derived from Issue 1 of the SVID.

6531 **Issue 4**

6532 The `<sys/types.h>` header is now marked as optional (OH); this header need not be included on
6533 XSI-conformant systems.

6534 The value for *owner* of `(uid_t)-1` is added to the DESCRIPTION to allow the use of `-1` by the
6535 owner of a file to change the group ID only.

6536 The APPLICATION USAGE section is added.

6537 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 6538 • The type of argument *path* is changed from `char*` to `const char*`.

6539 The following changes are incorporated for alignment with the FIPS requirements:

6540 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path
6541 name component is larger than {NAME_MAX} is now defined as mandatory and marked as
6542 an extension.

6543 • In the ERRORS section, the condition whereby [EPERM] is returned when an attempt is
6544 made to change the user ID of a file and the caller does not have appropriate privileges is
6545 now defined as mandatory and marked as an extension.

6546 **Issue 4, Version 2**

6547 The ERRORS section is updated for X/OPEN UNIX conformance as follows:

6548 • It states that [ELOOP] is returned if too many symbolic links are encountered during path
6549 name resolution.

6550 • The [EIO] and [EINTR] optional conditions are added.

6551 • A second [ENAMETOOLONG] condition is defined that may report excessive length of an
6552 intermediate result of path name resolution of a symbolic link.

6553 **Issue 6**

6554 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

6555 • The wording describing the optional dependency on `_POSIX_CHOWN_RESTRICTED` is
6556 restored.

6557 • The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.
6558 This is since behavior may vary from one file system to another.

6559 • The [EPERM] error is restored as an error dependent on `_POSIX_CHOWN_RESTRICTED`.
6560 This is since its operand is a path name and applications should be aware that the error may
6561 not occur for that path name if the file system does not support
6562 `_POSIX_CHOWN_RESTRICTED`.

6563 The following new requirements on POSIX implementations derive from alignment with the
6564 Single UNIX Specification:

6565 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
6566 required for conforming implementations of previous POSIX specifications, it was not
6567 required for UNIX applications.

6568 • The value for *owner* of `(uid_t)-1` allows the use of `-1` by the owner of a file to change the
6569 group ID only.

6570 • The [ELOOP] mandatory error condition is added.

6571 • The [EIO] and [EINTR] optional error conditions are added.

6572 • A second [ENAMETOOLONG] is added as an optional error condition.

6573 The following changes were made to align with the IEEE P1003.1a draft standard:

6574 • Clarification is added that the S_ISUID and S_ISGID bits do not need to be cleared when the
6575 process has appropriate privileges.

6576 • The [ELOOP] optional error condition is added.

6577 **NAME**

6578 cimag, cimagf, cimagl — complex imaginary functions

6579 **SYNOPSIS**

6580 #include <complex.h>

6581 double cimag(double complex z);

6582 float cimagf(float complex z);

6583 long double cimagl(long double complex z);

6584 **DESCRIPTION**

6585 cx The functionality described on this reference page is aligned with the ISO C standard. Any
6586 conflict between the requirements described here and the ISO C standard is unintentional. This
6587 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

6588 These functions shall compute the imaginary part of *z*.6589 **RETURN VALUE**

6590 These functions shall return the imaginary part value (as a real).

6591 **ERRORS**

6592 No errors are defined.

6593 **EXAMPLES**

6594 None.

6595 **APPLICATION USAGE**6596 For a variable *z* of complex type:6597 `z == creal(z) + cimag(z)*I`6598 **RATIONALE**

6599 None.

6600 **FUTURE DIRECTIONS**

6601 None.

6602 **SEE ALSO**6603 *carg()*, *conj()*, *cproj()*, *creal()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>6604 **CHANGE HISTORY**

6605 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

6606 **NAME**

6607 clearerr — clear indicators on a stream

6608 **SYNOPSIS**

6609 #include <stdio.h>

6610 void clearerr(FILE *stream);

6611 **DESCRIPTION**

6612 cx The functionality described on this reference page is aligned with the ISO C standard. Any
6613 conflict between the requirements described here and the ISO C standard is unintentional. This
6614 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

6615 The *clearerr()* function shall clear the end-of-file and error indicators for the stream to which
6616 *stream* points.

6617 **RETURN VALUE**6618 The *clearerr()* function shall return no value.6619 **ERRORS**

6620 No errors are defined.

6621 **EXAMPLES**

6622 None.

6623 **APPLICATION USAGE**

6624 None.

6625 **RATIONALE**

6626 None.

6627 **FUTURE DIRECTIONS**

6628 None.

6629 **SEE ALSO**

6630 The Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>

6631 **CHANGE HISTORY**

6632 First released in Issue 1. Derived from Issue 1 of the SVID.

6633 **NAME**

6634 clock — report CPU time used

6635 **SYNOPSIS**

6636 #include <time.h>

6637 clock_t clock(void);

6638 **DESCRIPTION**

6639 cx The functionality described on this reference page is aligned with the ISO C standard. Any
6640 conflict between the requirements described here and the ISO C standard is unintentional. This
6641 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

6642 The *clock()* function shall return the implementation's best approximation to the processor time
6643 used by the process since the beginning of an implementation-defined time related only to the
6644 process invocation.

6645 **RETURN VALUE**

6646 To determine the time in seconds, the value returned by *clock()* should be divided by the value
6647 xsi of the macro `CLOCKS_PER_SEC`. `CLOCKS_PER_SEC` is defined to be one million in `<time.h>`.
6648 If the processor time used is not available or its value cannot be represented, the function shall
6649 return the value `(clock_t)-1`.

6650 **ERRORS**

6651 No errors are defined.

6652 **EXAMPLES**

6653 None.

6654 **APPLICATION USAGE**

6655 In order to measure the time spent in a program, *clock()* should be called at the start of the
6656 program and its return value subtracted from the value returned by subsequent calls. The value
6657 returned by *clock()* is defined for compatibility across systems that have clocks with different
6658 resolutions. The resolution on any particular system need not be to microsecond accuracy.

6659 The value returned by *clock()* may wrap around on some systems. For example, on a machine
6660 with 32-bit values for `clock_t`, it wraps after 2 147 seconds or 36 minutes.

6661 **RATIONALE**

6662 None.

6663 **FUTURE DIRECTIONS**

6664 None.

6665 **SEE ALSO**

6666 *asctime()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,
6667 the Base Definitions volume of IEEE Std. 1003.1-200x, `<time.h>`

6668 **CHANGE HISTORY**

6669 First released in Issue 1. Derived from Issue 1 of the SVID.

6670 **Issue 4**6671 Reference to the resolution of `CLOCKS_PER_SEC` is marked as an extension.6672 The **ERRORS** section is added.

6673 Advice on how to calculate the time spent in a program is added to the **APPLICATION USAGE**
6674 section.

6675 The following changes are incorporated for alignment with the ISO C standard:

- 6676 • The `<time.h>` header is added to the SYNOPSIS section.
- 6677 • The DESCRIPTION and RETURN VALUE sections, though functionally equivalent to Issue
6678 3, are rewritten for clarity and consistency with the ISO C standard. This issue also defines
6679 under what circumstances `(clock_t)-1` can be returned by the function.
- 6680 • The function is no longer marked as an extension.

6681 **NAME**6682 clock_getcpuclockid — access a process CPU-time clock (**REALTIME**)6683 **SYNOPSIS**

6684 CPT #include <time.h>

6685 int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);

6686

6687 **DESCRIPTION**

6688 The *clock_getcpuclockid()* function shall return the clock ID of the CPU-time clock of the process
6689 specified by *pid*. If the process described by *pid* exists and the calling process has permission,
6690 the clock ID of this clock shall be returned in *clock_id*.

6691 If *pid* is zero, the *clock_getcpuclockid()* function shall return the clock ID of the CPU-time clock of
6692 the process making the call, in *clock_id*.

6693 The conditions under which one process has permission to obtain the CPU-time clock ID of
6694 other processes are implementation-defined.

6695 **RETURN VALUE**

6696 Upon successful completion, *clock_getcpuclockid()* shall return zero; otherwise, an error number
6697 shall be returned to indicate the error.

6698 **ERRORS**

6699 The *clock_getcpuclockid()* function shall fail if:

6700 [EPERM] The requesting process does not have permission to access the CPU-time
6701 clock for the process.

6702 The *clock_getcpuclockid()* function may fail if:

6703 [ESRCH] No process can be found corresponding to the process specified by *pid*.

6704 **EXAMPLES**

6705 None.

6706 **APPLICATION USAGE**

6707 The *clock_getcpuclockid()* function is part of the Process CPU-Time Clocks option and need not
6708 be provided on all implementations.

6709 **RATIONALE**

6710 None.

6711 **FUTURE DIRECTIONS**

6712 None.

6713 **SEE ALSO**

6714 *clock_getres()*, *timer_create()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**time.h**>

6715 **CHANGE HISTORY**

6716 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

6717 In the SYNOPSIS, the inclusion of <**sys/types.h**> is no longer required.

6718 NAME

6719 clock_getres, clock_gettime, clock_settime — clock and timer functions (**REALTIME**)

6720 SYNOPSIS

6721 TMR #include <time.h>

```
6722 int clock_getres(clockid_t clock_id, struct timespec *res);
6723 int clock_settime(clockid_t clock_id, const struct timespec *tp);
6724 int clock_gettime(clockid_t clock_id, struct timespec *tp);
6725
```

6726 DESCRIPTION

6727 The resolution of any clock can be obtained by calling *clock_getres()*. Clock resolutions are
 6728 implementation-defined and cannot be set by a process. If the argument *res* is not NULL, the
 6729 resolution of the specified clock shall be stored in the location pointed to by *res*. If *res* is NULL,
 6730 the clock resolution is not returned. If the *time* argument of *clock_settime()* is not a multiple of *res*,
 6731 then the value is truncated to a multiple of *res*.

6732 The *clock_gettime()* function shall return the current value *tp* for the specified clock, *clock_id*.

6733 The *clock_settime()* function shall set the specified clock, *clock_id*, to the value specified by *tp*.
 6734 Time values that are between two consecutive non-negative integer multiples of the resolution
 6735 of the specified clock are truncated down to the smaller multiple of the resolution.

6736 A clock may be system-wide (that is, visible to all processes) or per-process (measuring time that
 6737 is meaningful only within a process). All implementations shall support a *clock_id* of
 6738 CLOCK_REALTIME defined in <time.h>. This clock represents the realtime clock for the
 6739 system. For this clock, the values returned by *clock_gettime()* and specified by *clock_settime()*
 6740 represent the amount of time (in seconds and nanoseconds) since the Epoch. An implementation
 6741 may also support additional clocks. The interpretation of time values for these clocks is
 6742 unspecified.

6743 If the value of the CLOCK_REALTIME clock is set via *clock_settime()*, the new value of the clock
 6744 shall be used to determine the time of expiration for absolute time services based upon the
 6745 CLOCK_REALTIME clock. This applies to the time at which armed absolute timers expire. If the
 6746 absolute time requested at the invocation of such a time service is before the new value of the
 6747 clock, the time service shall expire immediately as if the clock had reached the requested time
 6748 normally.

6749 Setting the value of the CLOCK_REALTIME clock via *clock_settime()* shall have no effect on
 6750 threads that are blocked waiting for a relative time service based upon this clock, including the
 6751 *nanosleep()* function; nor on the expiration of relative timers based upon this clock.
 6752 Consequently, these time services shall expire when the requested relative interval elapses,
 6753 independently of the new or old value of the clock.

6754 MON If the Monotonic Clock option is supported, all implementations shall support a *clock_id* of
 6755 CLOCK_MONOTONIC defined in <time.h>. This clock represents the monotonic clock for the
 6756 system. For this clock, the value returned by *clock_gettime()* represents the amount of time (in
 6757 seconds and nanoseconds) since an unspecified point in the past (for example, system start-up
 6758 time, or the Epoch). This point does not change after system start-up time. The value of the
 6759 CLOCK_MONOTONIC clock cannot be set via *clock_settime()*. This function shall fail if it is
 6760 invoked with a *clock_id* argument of CLOCK_MONOTONIC.

6761 The effect of setting a clock via *clock_settime()* on armed per-process timers associated with a
 6762 clock other than CLOCK_REALTIME is implementation-defined.

6763 CS If the value of the CLOCK_REALTIME clock is set via *clock_settime()*, the new value of the clock
 6764 shall be used to determine the time at which the system shall awaken a thread blocked on an

- 6765 absolute *clock_nanosleep()* call based upon the CLOCK_REALTIME clock. If the absolute time
 6766 requested at the invocation of such a time service is before the new value of the clock, the call
 6767 shall return immediately as if the clock had reached the requested time normally.
- 6768 Setting the value of the CLOCK_REALTIME clock via *clock_settime()* shall have no effect on any
 6769 thread that is blocked on a relative *clock_nanosleep()* call. Consequently, the call shall return
 6770 when the requested relative interval elapses, independently of the new or old value of the clock.
- 6771 The appropriate privilege to set a particular clock is implementation-defined.
- 6772 CPT If `_POSIX_CPUTIME` is defined, implementations shall support clock ID values obtained by
 6773 invoking *clock_getcpuclockid()*, which represent the CPU-time clock of a given process.
 6774 Implementations shall also support the special `clockid_t` value
 6775 `CLOCK_PROCESS_CPUTIME_ID`, which represents the CPU-time clock of the calling process
 6776 when invoking one of the *clock_**() or *timer_**() functions. For these clock IDs, the values
 6777 returned by *clock_gettime()* and specified by *clock_settime()* represent the amount of execution
 6778 time of the process associated with the clock. Changing the value of a CPU-time clock via
 6779 *clock_settime()* shall have no effect on the behavior of the sporadic server scheduling policy (see
 6780 **Scheduling Policies** (on page 546)).
- 6781 TCT If `_POSIX_THREAD_CPUTIME` is defined, implementations shall support clock ID values
 6782 obtained by invoking *pthread_getcpuclockid()*, which represent the CPU-time clock of a given
 6783 thread. Implementations shall also support the special `clockid_t` value
 6784 `CLOCK_THREAD_CPUTIME_ID`, which represents the CPU-time clock of the calling thread
 6785 when invoking one of the *clock_**() or *timer_**() functions. For these clock IDs, the values
 6786 returned by *clock_gettime()* and specified by *clock_settime()* represent the amount of execution
 6787 time of the thread associated with the clock. Changing the value of a CPU-time clock via
 6788 *clock_settime()* shall have no effect on the behavior of the sporadic server scheduling policy (see
 6789 **Scheduling Policies** (on page 546)).
- 6790 **RETURN VALUE**
- 6791 A return value of 0 shall indicate that the call succeeded. A return value of -1 shall indicate that
 6792 an error occurred, and *errno* shall be set to indicate the error.
- 6793 **ERRORS**
- 6794 The *clock_getres()*, *clock_gettime()*, and *clock_settime()* functions shall fail if:
- 6795 [EINVAL] The *clock_id* argument does not specify a known clock.
- 6796 The *clock_settime()* function shall fail if:
- 6797 [EINVAL] The *tp* argument to *clock_settime()* is outside the range for the given clock ID.
- 6798 [EINVAL] The *tp* argument specified a nanosecond value less than zero or greater than
 6799 or equal to 1 000 million.
- 6800 MON [EINVAL] The value of the *clock_id* argument is `CLOCK_MONOTONIC`.
- 6801 The *clock_settime()* function may fail if:
- 6802 [EPERM] The requesting process does not have the appropriate privilege to set the
 6803 specified clock.

6804 **EXAMPLES**

6805 None.

6806 **APPLICATION USAGE**

6807 These functions are part of the Timers option and need not be available on all implementations.

6808 Note that the absolute value of the monotonic clock is meaningless (because its origin is
 6809 arbitrary), and thus there is no need to set it. Furthermore, realtime applications can rely on the
 6810 fact that the value of this clock is never set and, therefore, that time intervals measured with this
 6811 clock will not be affected by calls to *clock_settime()*.

6812 **RATIONALE**

6813 None.

6814 **FUTURE DIRECTIONS**

6815 None.

6816 **SEE ALSO**

6817 *clock_getcpuclockid()*, *clock_nanosleep()*, *ctime()*, *mq_timedreceive()*, *mq_timedsend()*, *nanosleep()*,
 6818 *pthread_mutex_timedlock()*, *sem_timedwait()*, *time()*, *timer_create()*, *timer_getoverrun()*, the Base
 6819 Definitions volume of IEEE Std. 1003.1-200x, <**time.h**>

6820 **CHANGE HISTORY**

6821 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

6822 **Issue 6**

6823 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 6824 implementation does not support the Timers option.

6825 The APPLICATION USAGE section is added.

6826 The following changes were made to align with the IEEE P1003.1a draft standard:

6827 • Clarification is added of the effect of resetting the clock resolution.

6828 CPU-time clocks and the *clock_getcpuclockid()* function are added for alignment with
 6829 IEEE Std. 1003.1d-1999.

6830 The following changes are added for alignment with IEEE Std. 1003.1j-2000:

6831 • The DESCRIPTION is updated as follows:

6832 — The value returned by *clock_gettime()* for CLOCK_MONOTONIC is specified.6833 — *clock_settime()* failing for CLOCK_MONOTONIC is specified.

6834 — The effects of *clock_settime()* on the *clock_nanosleep()* function with respect to
 6835 CLOCK_REALTIME is specified.

6836 • An [EINVAL] error is added to the ERRORS section, indicating that *clock_settime()* fails for
 6837 CLOCK_MONOTONIC.

6838 • The APPLICATION USAGE section notes that the CLOCK_MONOTONIC clock need not
 6839 and shall not be set by *clock_settime()* since the absolute value of the CLOCK_MONOTONIC
 6840 clock is meaningless.

6841 • The *clock_nanosleep()*, *mq_timedreceive()*, *mq_timedsend()*, *pthread_mutex_timedlock()*,
 6842 *sem_timedwait()*, *timer_create()*, and *timer_settime()* functions are added to the SEE ALSO
 6843 section.

6844 NAME

6845 clock_nanosleep — high resolution sleep with specifiable clock

6846 SYNOPSIS

6847 cs #include <time.h>

```
6848 int clock_nanosleep(clockid_t clock_id, int flags,  
6849     const struct timespec *rqtp, struct timespec *rmtp);  
6850
```

6851 DESCRIPTION

6852 If the flag `TIMER_ABSTIME` is not set in the *flags* argument, the *clock_nanosleep()* function shall
6853 cause the current thread to be suspended from execution until either the time interval specified
6854 by the *rqtp* argument has elapsed, or a signal is delivered to the calling thread and its action is to
6855 invoke a signal-catching function, or the process is terminated. The clock used to measure the
6856 time shall be the clock specified by *clock_id*.

6857 If the flag `TIMER_ABSTIME` is set in the *flags* argument, the *clock_nanosleep()* function shall
6858 cause the current thread to be suspended from execution until either the time value of the clock
6859 specified by *clock_id* reaches the absolute time specified by the *rqtp* argument, or a signal is
6860 delivered to the calling thread and its action is to invoke a signal-catching function, or the
6861 process is terminated. If, at the time of the call, the time value specified by *rqtp* is less than or
6862 equal to the time value of the specified clock, then *clock_nanosleep()* shall return immediately
6863 and the calling process shall not be suspended.

6864 The suspension time caused by this function may be longer than requested because the
6865 argument value is rounded up to an integer multiple of the sleep resolution, or because of the
6866 scheduling of other activity by the system. But, except for the case of being interrupted by a
6867 signal, the suspension time for the relative *clock_nanosleep()* function (that is, with the
6868 `TIMER_ABSTIME` flag not set) shall not be less than the time interval specified by *rqtp*, as
6869 measured by the corresponding clock. The suspension for the absolute *clock_nanosleep()* function
6870 (that is, with the `TIMER_ABSTIME` flag set) shall be in effect at least until the value of the
6871 corresponding clock reaches the absolute time specified by *rqtp*, except for the case of being
6872 interrupted by a signal.

6873 The use of the *clock_nanosleep()* function shall have no effect on the action or blockage of any
6874 signal.

6875 The *clock_nanosleep()* function shall fail if the *clock_id* argument refers to the CPU-time clock of
6876 the calling thread. It is unspecified if *clock_id* values of other CPU-time clocks are allowed.

6877 RETURN VALUE

6878 If the *clock_nanosleep()* function returns because the requested time has elapsed, its return value
6879 shall be zero.

6880 If the *clock_nanosleep()* function returns because it has been interrupted by a signal, it shall return
6881 the corresponding error value. For the relative *clock_nanosleep()* function, if the *rmtp* argument is
6882 non-NULL, the **timespec** structure referenced by it shall be updated to contain the amount of
6883 time remaining in the interval (the requested time minus the time actually slept). If the *rmtp*
6884 argument is NULL, the remaining time is not returned. The absolute *clock_nanosleep()* function
6885 has no effect on the structure referenced by *rmtp*.

6886 If *clock_nanosleep()* fails, it shall return the corresponding error value.

6887 **ERRORS**6888 The `clock_nanosleep()` function shall fail if:

- 6889 [EINTR] The `clock_nanosleep()` function was interrupted by a signal.
- 6890 [EINVAL] The `rtp` argument specified a nanosecond value less than zero or greater than or equal to 1 000 million; or the `TIMER_ABSTIME` flag was specified in `flags` and the `rtp` argument is outside the range for the clock specified by `clock_id`; or the `clock_id` argument does not specify a known clock, or specifies the CPU-time clock of the calling thread.
- 6895 [ENOTSUP] The `clock_id` argument specifies a clock for which `clock_nanosleep()` is not supported, such as a CPU-time clock.

6897 **EXAMPLES**

6898 None.

6899 **APPLICATION USAGE**

6900 Calling `clock_nanosleep()` with the value `TIMER_ABSTIME` not set in the `flags` argument and with a `clock_id` of `CLOCK_REALTIME` is equivalent to calling `nanosleep()` with the same `rtp` and `rmtp` arguments.

6903 **RATIONALE**

6904 The `nanosleep()` function specifies that the system-wide clock `CLOCK_REALTIME` is used to measure the elapsed time for this time service. However, with the introduction of the monotonic clock `CLOCK_MONOTONIC` a new relative sleep function is needed to allow an application to take advantage of the special characteristics of this clock.

6908 There are many applications in which a process needs to be suspended and then activated multiple times in a periodic way; for example, to poll the status of a non-interrupting device or to refresh a display device. For these cases, it is known that precise periodic activation cannot be achieved with a relative `sleep()` or `nanosleep()` function call. Suppose, for example, a periodic process that is activated at time T_0 , executes for a while, and then wants to suspend itself until time T_0+T , the period being T . If this process wants to use the `nanosleep()` function, it must first call `clock_gettime()` to get the current time, then calculate the difference between the current time and T_0+T and, finally, call `nanosleep()` using the computed interval. However, the process could be preempted by a different process between the two function calls, and in this case the interval computed would be wrong; the process would wake up later than desired. This problem would not occur with the absolute `clock_nanosleep()` function, since only one function call would be necessary to suspend the process until the desired time. In other cases, however, a relative sleep is needed, and that is why both functionalities are required.

6921 Although it is possible to implement periodic processes using the timers interface, this implementation would require the use of signals, and the reservation of some signal numbers. In this regard, the reasons for including an absolute version of the `clock_nanosleep()` function in IEEE Std. 1003.1-200x are the same as for the inclusion of the relative `nanosleep()`.

6925 It is also possible to implement precise periodic processes using `pthread_cond_timedwait()`, in which an absolute timeout is specified that takes effect if the condition variable involved is never signaled. However, the use of this interface is unnatural, and involves performing other operations on mutexes and condition variables that imply an unnecessary overhead. Furthermore, `pthread_cond_timedwait()` is not available in implementations that do not support threads.

6931 Although the interface of the relative and absolute versions of the new high resolution sleep service is the same `clock_nanosleep()` function, the `rmtp` argument is only used in the relative sleep. This argument is needed in the relative `clock_nanosleep()` function to reissue the function

6934 call if it is interrupted by a signal, but it is not needed in the absolute *clock_nanosleep()* function
6935 call; if the call is interrupted by a signal, the absolute *clock_nanosleep()* function can be invoked
6936 again with the same *rtp* argument used in the interrupted call.

6937 **FUTURE DIRECTIONS**

6938 None.

6939 **SEE ALSO**

6940 *clock_getres()*, *nanosleep()*, *pthread_cond_timedwait()*, *sleep()*, the Base Definitions volume of
6941 IEEE Std. 1003.1-200x, <**time.h**>

6942 **CHANGE HISTORY**

6943 First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.

6944 **NAME**

6945 clog, clogf, clogl — complex natural logarithm functions

6946 **SYNOPSIS**

6947 #include <complex.h>

6948 double complex clog(double complex *z*);6949 float complex clogf(float complex *z*);6950 long double complex clogl(long double complex *z*);6951 **DESCRIPTION**

6952 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
6953 conflict between the requirements described here and the ISO C standard is unintentional. This
6954 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

6955 These functions shall compute the complex natural (base *e*) logarithm of *z*, with a branch cut
6956 along the negative real axis.

6957 **RETURN VALUE**

6958 These functions shall return the complex natural logarithm value, in the range of a strip
6959 mathematically unbounded along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary
6960 axis.

6961 **ERRORS**

6962 No errors are defined.

6963 **EXAMPLES**

6964 None.

6965 **APPLICATION USAGE**

6966 None.

6967 **RATIONALE**

6968 None.

6969 **FUTURE DIRECTIONS**

6970 None.

6971 **SEE ALSO**6972 *cexp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>6973 **CHANGE HISTORY**

6974 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

6975 **NAME**

6976 close — close a file descriptor

6977 **SYNOPSIS**

6978 #include <unistd.h>

6979 int close(int *fildes*);6980 **DESCRIPTION**

6981 The *close()* function shall deallocate the file descriptor indicated by *fildes*. To deallocate means
 6982 to make the file descriptor available for return by subsequent calls to *open()* or other functions
 6983 that allocate file descriptors. All outstanding record locks owned by the process on the file
 6984 associated with the file descriptor shall be removed (that is, unlocked).

6985 If *close()* is interrupted by a signal that is to be caught, it shall return -1 with *errno* set to [EINTR]
 6986 and the state of *fildes* is unspecified. If an I/O error occurred while reading from or writing to the
 6987 file system during *close()*, it may return -1 with *errno* set to [EIO]; if this error is returned, the
 6988 state of *fildes* is unspecified.

6989 When all file descriptors associated with a pipe or FIFO special file are closed, any data
 6990 remaining in the pipe or FIFO shall be discarded.

6991 When all file descriptors associated with an open file description have been closed the open file
 6992 description shall be freed.

6993 If the link count of the file is 0, when all file descriptors associated with the file are closed, the
 6994 space occupied by the file shall be freed and the file shall no longer be accessible.

6995 **XSR** If a STREAMS-based *fildes* is closed and the calling process was previously registered to receive
 6996 a SIGPOLL signal for events associated with that STREAM, the calling process shall be
 6997 unregistered for events associated with the STREAM. The last *close()* for a STREAM causes the
 6998 STREAM associated with *fildes* to be dismantled. If O_NONBLOCK is not set and there have
 6999 been no signals posted for the STREAM, and if there is data on the module's write queue, *close()*
 7000 waits for an unspecified time (for each module and driver) for any output to drain before
 7001 dismantling the STREAM. The time delay can be changed via an I_SETCLTIME *ioctl()* request. If
 7002 the O_NONBLOCK flag is set, or if there are any pending signals, *close()* does not wait for
 7003 output to drain, and dismantles the STREAM immediately.

7004 If the implementation supports STREAMS-based pipes, and *fildes* is associated with one end of a
 7005 pipe, the last *close()* causes a hangup to occur on the other end of the pipe. In addition, if the
 7006 other end of the pipe has been named by *fattach()*, then the last *close()* forces the named end to
 7007 be detached by *fdetach()*. If the named end has no open file descriptors associated with it and
 7008 gets detached, the STREAM associated with that end is also dismantled.

7009 If *fildes* refers to the master side of a pseudo-terminal, and this is the last close, a SIGHUP signal
 7010 is sent to the process group, if any, for which the slave side of the pseudo-terminal is the
 7011 controlling terminal. It is unspecified whether closing the master side of the pseudo-terminal
 7012 flushes all queued input and output.

7013 If *fildes* refers to the slave side of a STREAMS-based pseudo-terminal, a zero-length message
 7014 may be sent to the master.

7015 **AIO** When there is an outstanding cancelable asynchronous I/O operation against *fildes* when *close()*
 7016 is called, that I/O operation may be canceled. An I/O operation that is not canceled completes
 7017 as if the *close()* operation had not yet occurred. All operations that are not canceled shall
 7018 complete as if the *close()* blocked until the operations completed. The *close()* operation itself
 7019 need not block awaiting such I/O completion. Whether any I/O operation is canceled, and
 7020 which I/O operation may be canceled upon *close()*, is implementation-defined.

7021 MF|SHM If a shared memory object or a memory mapped file remains referenced at the last close (that is,
 7022 a process has it mapped), then the entire contents of the memory object shall persist until the
 7023 memory object becomes unreferenced. If this is the last close of a shared memory object or a
 7024 memory mapped file and the close results in the memory object becoming unreferenced, and the
 7025 memory object has been unlinked, then the memory object shall be removed.

7026 If *fdes* refers to a socket, *close()* shall cause the socket to be destroyed. If the socket is in
 7027 connection-mode, and the `SO_LINGER` option is set for the socket with non-zero linger time,
 7028 and the socket has untransmitted data, then *close()* shall block for up to the current linger
 7029 interval until all data is transmitted.

7030 RETURN VALUE

7031 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 7032 indicate the error.

7033 ERRORS

7034 The *close()* function shall fail if:

7035 [EBADF] The *fdes* argument is not a valid file descriptor.

7036 [EINTR] The *close()* function was interrupted by a signal.

7037 The *close()* function may fail if:

7038 [EIO] An I/O error occurred while reading from or writing to the file system.

7039 EXAMPLES

7040 Reassigning a File Descriptor

7041 The following example closes the file descriptor associated with standard output for the current
 7042 process, re-assigns standard output to a new file descriptor, and closes the original file
 7043 descriptor to clean up. This example assumes that the file descriptor 0 (which is the descriptor
 7044 for standard input) is not closed.

```
7045 #include <unistd.h>
7046 ...
7047 int pfd;
7048 ...
7049 close(1);
7050 dup(pfd);
7051 close(pfd);
7052 ...
```

7053 Incidentally, this is exactly what could be achieved using:

```
7054 dup2(pfd, 1);
7055 close(pfd);
```

7056 Closing a File Descriptor

7057 In the following example, *close()* is used to close a file descriptor after an unsuccessful attempt is
 7058 made to associate that file descriptor with a stream.

```
7059 #include <stdio.h>
7060 #include <unistd.h>
7061 #include <stdlib.h>
```

```

7062     #define LOCKFILE "/etc/ptmp"
7063     ...
7064     int pfd;
7065     FILE *fpfd;
7066     ...
7067     if ((fpfd = fdopen (pfd, "w")) == NULL) {
7068         close(pfd);
7069         unlink(LOCKFILE);
7070         exit(1);
7071     }
7072     ...

```

7073 APPLICATION USAGE

7074 An application that had used the *stdio* routine *fopen()* to open a file should use the
 7075 corresponding *fclose()* routine rather than *close()*. Once a file is closed, the file descriptor no
 7076 longer exists, since the integer corresponding to it no longer refers to a file.

7077 RATIONALE

7078 The use of interruptible device close routines should be discouraged to avoid problems with the
 7079 implicit closes of file descriptors by *exec* and *exit()*. This volume of IEEE Std. 1003.1-200x only
 7080 intends to permit such behavior by specifying the [EINTR] error condition.

7081 FUTURE DIRECTIONS

7082 None.

7083 SEE ALSO

7084 *fattach()*, *fclose()*, *fdetach()*, *fopen()*, *ioctl()*, *open()*, the Base Definitions volume of
 7085 IEEE Std. 1003.1-200x, <**unistd.h**>, Section 2.6 (on page 539)

7086 CHANGE HISTORY

7087 First released in Issue 1. Derived from Issue 1 of the SVID.

7088 Issue 4

7089 The <**unistd.h**> header is added to the SYNOPSIS section.

7090 Issue 4, Version 2

7091 The following changes are incorporated for X/OPEN UNIX conformance:

- 7092 • The DESCRIPTION is updated to describe the actions of closing a file descriptor referring to
 7093 a STREAMS-based file or either side of a pseudo-terminal.
- 7094 • The ERRORS section describes a condition under which the [EIO] error may be returned.

7095 Issue 5

7096 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

7097 Issue 6

7098 The DESCRIPTION related to a STREAMS-based file or pseudo-terminal is marked as part of the
 7099 XSI STREAMS Option Group.

7100 The following new requirements on POSIX implementations derive from alignment with the
 7101 Single UNIX Specification:

- 7102 • The [EIO] error condition is added as an optional error.
- 7103 • The DESCRIPTION is updated to describe the state of the *fildev* file descriptor as unspecified
 7104 if an I/O error occurs and an [EIO] error condition is returned.

7105 Text referring to sockets is added to the DESCRIPTION.

7106
7107
7108

The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by specifying that shared memory objects and memory mapped files (and not typed memory objects) are the types of memory objects to which the paragraph on last closes applies.

7109 **NAME**

7110 closedir — close a directory stream

7111 **SYNOPSIS**

7112 #include <dirent.h>

7113 int closedir(DIR *dirp);

7114 **DESCRIPTION**

7115 The *closedir()* function shall close the directory stream referred to by the argument *dirp*. Upon
7116 return, the value of *dirp* may no longer point to an accessible object of the type **DIR**. If a file
7117 descriptor is used to implement type **DIR**, that file descriptor shall be closed.

7118 **RETURN VALUE**

7119 Upon successful completion, *closedir()* shall return 0; otherwise, -1 shall be returned and *errno*
7120 set to indicate the error.

7121 **ERRORS**7122 The *closedir()* function may fail if:7123 [EBADF] The *dirp* argument does not refer to an open directory stream. |7124 [EINTR] The *closedir()* function was interrupted by a signal. |7125 **EXAMPLES**7126 **Closing a Directory Stream**7127 The following program fragment demonstrates how the *closedir()* function is used.

```
7128 ...  
7129     DIR *dir;  
7130     struct dirent *dp;  
7131 ...  
7132     if ((dir = opendir(".")) == NULL) {  
7133 ...  
7134     }  
7135     while ((dp = readdir(dir)) != NULL) {  
7136 ...  
7137     }  
7138     closedir(dir);  
7139 ...
```

7140 **APPLICATION USAGE**

7141 None.

7142 **RATIONALE**

7143 None.

7144 **FUTURE DIRECTIONS**

7145 None.

7146 **SEE ALSO**7147 *opendir()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <dirent.h> |

7148 **CHANGE HISTORY**

7149 First released in Issue 2.

7150 **Issue 4**7151 The `<sys/types.h>` header is now marked as optional (OH); this header need not be included on
7152 XSI-conformant systems.

7153 The [EINTR] error is marked as an extension.

7154 **Issue 6**7155 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.7156 The following new requirements on POSIX implementations derive from alignment with the
7157 Single UNIX Specification:

- 7158 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
7159 required for conforming implementations of previous POSIX specifications, it was not
7160 required for UNIX applications.
- 7161 • The [EINTR] error condition is added as an optional error condition.

7162 **NAME**

7163 closelog, openlog, setlogmask, syslog — control system log

7164 **SYNOPSIS**

```

7165 xSI #include <syslog.h>
7166
7166 void closelog(void);
7167 void openlog(const char *ident, int logopt, int facility);
7168 int setlogmask(int maskpri);
7169 void syslog(int priority, const char *message, ... /* arguments */);
7170

```

7171 **DESCRIPTION**

7172 The *syslog()* function shall send a message to an implementation-defined logging facility, which
 7173 may log it in an implementation-defined system log, write it to the system console, forward it to
 7174 a list of users, or forward it to the logging facility on another host over the network. The logged
 7175 message shall include a message header and a message body. The message header contains at
 7176 least a timestamp and a tag string.

7177 The message body is generated from the *message* and following arguments in the same manner
 7178 as if these were arguments to *printf()*, except that occurrences of %m in the format string
 7179 pointed to by the *message* argument are replaced by the error message string associated with the
 7180 current value of *errno*. A trailing <newline> character is added if needed.

7181 Values of the *priority* argument are formed by OR'ing together a severity level value and an
 7182 optional facility value. If no facility value is specified, the current default facility value is used.

7183 Possible values of severity level include:

7184	LOG_EMERG	A panic condition.
7185	LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system database.
7186		
7187	LOG_CRIT	Critical conditions, such as hard device errors.
7188	LOG_ERR	Errors.
7189	LOG_WARNING	
7190		Warning messages.
7191	LOG_NOTICE	Conditions that are not error conditions, but that may require special handling.
7192		
7193	LOG_INFO	Informational messages.
7194	LOG_DEBUG	Messages that contain information normally of use only when debugging a program.
7195		

7196 The facility indicates the application or system component generating the message. Possible
 7197 facility values include:

7198	LOG_USER	Messages generated by arbitrary processes. This is the default facility identifier if none is specified.
7199		
7200	LOG_LOCAL0	Reserved for local use.
7201	LOG_LOCAL1	Reserved for local use.
7202	LOG_LOCAL2	Reserved for local use.

- 7203 LOG_LOCAL3 Reserved for local use.
- 7204 LOG_LOCAL4 Reserved for local use.
- 7205 LOG_LOCAL5 Reserved for local use.
- 7206 LOG_LOCAL6 Reserved for local use.
- 7207 LOG_LOCAL7 Reserved for local use.
- 7208 The *openlog()* function shall set process attributes that affect subsequent calls to *syslog()*. The
- 7209 *ident* argument is a string that is prepended to every message. The *logopt* argument indicates
- 7210 logging options. Values for *logopt* are constructed by a bitwise-inclusive OR of zero or more of
- 7211 the following:
- 7212 LOG_PID Log the process ID with each message. This is useful for identifying specific
- 7213 processes.
- 7214 LOG_CONS Write messages to the system console if they cannot be sent to the logging
- 7215 facility. The *syslog()* function ensures that the process does not acquire the
- 7216 console as a controlling terminal in the process of writing the message.
- 7217 LOG_NDELAY Open the connection to the logging facility immediately. Normally the open is
- 7218 delayed until the first message is logged. This is useful for programs that need
- 7219 to manage the order in which file descriptors are allocated.
- 7220 LOG_ODELAY Delay open until *syslog()* is called.
- 7221 LOG_NOWAIT Do not wait for child processes that may have been created during the course
- 7222 of logging the message. This option should be used by processes that enable
- 7223 notification of child termination using SIGCHLD, since *syslog()* may
- 7224 otherwise block waiting for a child whose exit status has already been
- 7225 collected.
- 7226 The *facility* argument encodes a default facility to be assigned to all messages that do not have
- 7227 an explicit facility already encoded. The initial default facility is LOG_USER.
- 7228 The *openlog()* and *syslog()* functions may allocate a file descriptor. It is not necessary to call
- 7229 *openlog()* prior to calling *syslog()*.
- 7230 The *closelog()* function shall close any open file descriptors allocated by previous calls to
- 7231 *openlog()* or *syslog()*.
- 7232 The *setlogmask()* function shall set the log priority mask for the current process to *maskpri* and
- 7233 return the previous mask. If the *maskpri* argument is 0, the current log mask is not modified.
- 7234 Calls by the current process to *syslog()* with a priority not set in *maskpri* shall be rejected. The
- 7235 default log mask allows all priorities to be logged. A call to *openlog()* is not required prior to
- 7236 calling *setlogmask()*.
- 7237 Symbolic constants for use as values of the *logopt*, *facility*, *priority*, and *maskpri* arguments are
- 7238 defined in the <syslog.h> header.
- 7239 **RETURN VALUE**
- 7240 The *setlogmask()* function shall return the previous log priority mask. The *closelog()*, *openlog()*,
- 7241 and *syslog()* functions shall return no value.
- 7242 **ERRORS**
- 7243 No errors are defined.

7244 **EXAMPLES**7245 **Using openlog()**

7246 The following example causes subsequent calls to *syslog()* to log the process ID with each
7247 message, and to write messages to the system console if they cannot be sent to the logging
7248 facility.

```
7249 #include <syslog.h>
7250 char *ident = "Process demo";
7251 int logopt = LOG_PID | LOG_CONS;
7252 int facility = LOG_USER;
7253 ...
7254 openlog(ident, logopt, facility);
```

7255 **Using setlogmask()**

7256 The following example causes subsequent calls to *syslog()* to accept error messages or messages
7257 generated by arbitrary processes, and to reject all other messages.

```
7258 #include <syslog.h>
7259 int result;
7260 int mask = LOG_MASK (LOG_ERR | LOG_USER);
7261 ...
7262 result = setlogmask(mask);
```

7263 **Using syslog**

7264 The following example sends the message "This is a message" to the default logging
7265 facility, marking the message as an error message generated by random processes.

```
7266 #include <syslog.h>
7267 char *message = "This is a message";
7268 int priority = LOG_ERR | LOG_USER;
7269 ...
7270 syslog(priority, message);
```

7271 **APPLICATION USAGE**

7272 None.

7273 **RATIONALE**

7274 None.

7275 **FUTURE DIRECTIONS**

7276 None.

7277 **SEE ALSO**

7278 *printf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**syslog.h**>

7279 **CHANGE HISTORY**

7280 First released in Issue 4, Version 2.

7281 **Issue 5**

7282 Moved from X/OPEN UNIX extension to BASE.

7283 NAME

7284 confstr — get configurable variables

7285 SYNOPSIS

7286 #include <unistd.h>

7287 size_t confstr(int name, char *buf, size_t len);

7288 DESCRIPTION

7289 The *confstr()* function provides a method for applications to get configuration-defined string
 7290 values. Its use and purpose are similar to *sysconf()*, but it is used where string values rather than
 7291 numeric values are returned.

7292 The *name* argument represents the system variable to be queried. The implementation shall
 7293 support the following name values, defined in <unistd.h>. It may support others:

7294 _CS_PATH
 7295 _CS_POSIX_V6_ILP32_OFF32_CFLAGS
 7296 _CS_POSIX_V6_ILP32_OFF32_LDFLAGS
 7297 _CS_POSIX_V6_ILP32_OFF32_LIBS
 7298 _CS_POSIX_V6_ILP32_OFF32_LINTFLAGS
 7299 _CS_POSIX_V6_ILP32_OFFBIG_CFLAGS
 7300 _CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS
 7301 _CS_POSIX_V6_ILP32_OFFBIG_LIBS
 7302 _CS_POSIX_V6_ILP32_OFFBIG_LINTFLAGS
 7303 _CS_POSIX_V6_LP64_OFF64_CFLAGS
 7304 _CS_POSIX_V6_LP64_OFF64_LDFLAGS
 7305 _CS_POSIX_V6_LP64_OFF64_LIBS
 7306 _CS_POSIX_V6_LP64_OFF64_LINTFLAGS
 7307 _CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS
 7308 _CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS
 7309 _CS_POSIX_V6_LPBIG_OFFBIG_LIBS
 7310 _CS_POSIX_V6_LPBIG_OFFBIG_LINTFLAGS
 7311 XSI CS_XBS5_ILP32_OFF32_CFLAGS (LEGACY)
 7312 CS_XBS5_ILP32_OFF32_LDFLAGS (LEGACY)
 7313 CS_XBS5_ILP32_OFF32_LIBS (LEGACY)
 7314 CS_XBS5_ILP32_OFF32_LINTFLAGS (LEGACY)
 7315 CS_XBS5_ILP32_OFFBIG_CFLAGS (LEGACY)
 7316 CS_XBS5_ILP32_OFFBIG_LDFLAGS (LEGACY)
 7317 CS_XBS5_ILP32_OFFBIG_LIBS (LEGACY)
 7318 CS_XBS5_ILP32_OFFBIG_LINTFLAGS (LEGACY)
 7319 CS_XBS5_LP64_OFF64_CFLAGS (LEGACY)
 7320 CS_XBS5_LP64_OFF64_LDFLAGS (LEGACY)
 7321 CS_XBS5_LP64_OFF64_LIBS (LEGACY)
 7322 CS_XBS5_LP64_OFF64_LINTFLAGS (LEGACY)
 7323 CS_XBS5_LPBIG_OFFBIG_CFLAGS (LEGACY)
 7324 CS_XBS5_LPBIG_OFFBIG_LDFLAGS (LEGACY)
 7325 CS_XBS5_LPBIG_OFFBIG_LIBS (LEGACY)
 7326 CS_XBS5_LPBIG_OFFBIG_LINTFLAGS (LEGACY)

7327

7328 If *len* is not 0, and if *name* has a configuration-defined value, *confstr()* shall copy that value into
 7329 the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes,
 7330 including the terminating null, then *confstr()* shall truncate the string to *len*-1 bytes and null-
 7331 terminate the result. The application can detect that the string was truncated by comparing the

7332 value returned by *confstr()* with *len*.

7333 If *len* is 0 and *buf* is a null pointer, then *confstr()* shall still return the integer value as defined
7334 below, but shall not return a string. If *len* is 0 but *buf* is not a null pointer, the result is
7335 unspecified.

7336 If the implementation supports the Shell option, the string stored in *buf* after a call to:

```
7337 confstr(_CS_PATH, buf, sizeof(buf))
```

7338 can be used as a value of the *PATH* environment variable that accesses all of the standard
7339 utilities of IEEE Std. 1003.1-200x, if the return value is less than or equal to *sizeof(buf)*.

7340 RETURN VALUE

7341 If *name* has a configuration-defined value, *confstr()* shall return the size of buffer that would be
7342 needed to hold the entire configuration-defined value including the terminating null. If this
7343 return value is greater than *len*, the string returned in *buf* is truncated.

7344 If *name* is invalid, *confstr()* shall return 0 and set *errno* to indicate the error.

7345 If *name* does not have a configuration-defined value, *confstr()* shall return 0 and leave *errno*
7346 unchanged.

7347 ERRORS

7348 The *confstr()* function shall fail if:

7349 [EINVAL] The value of the *name* argument is invalid.

7350 EXAMPLES

7351 None.

7352 APPLICATION USAGE

7353 An application can distinguish between an invalid *name* parameter value and one that
7354 corresponds to a configurable variable that has no configuration-defined value by checking if
7355 *errno* is modified. This mirrors the behavior of *sysconf()*.

7356 The original need for this function was to provide a way of finding the configuration-defined
7357 default value for the environment variable *PATH*. Since *PATH* can be modified by the user to
7358 include directories that could contain utilities replacing the standard utilities in the Shell and
7359 Utilities volume of IEEE Std. 1003.1-200x, applications need a way to determine the system-
7360 supplied *PATH* environment variable value that contains the correct search path for the standard
7361 utilities.

7362 An application could use:

```
7363 confstr(name, (char *)NULL, (size_t)0)
```

7364 to find out how big a buffer is needed for the string value; use *malloc()* to allocate a buffer to
7365 hold the string; and call *confstr()* again to get the string. Alternately, it could allocate a fixed,
7366 static buffer that is big enough to hold most answers (perhaps 512 or 1024 bytes), but then use
7367 *malloc()* to allocate a larger buffer if it finds that this is too small.

7368 RATIONALE

7369 Application developers can normally determine any configuration variable by means of reading
7370 from the stream opened by a call to:

```
7371 popen("command -p getconf variable", "r");
```

7372 The *confstr()* function with a *name* argument of *_CS_PATH* returns a string that can be used as a
7373 *PATH* environment variable setting that will reference the standard shell and utilities as
7374 described in the Shell and Utilities volume of IEEE Std. 1003.1-200x.

7375 The *confstr()* function copies the returned string into a buffer supplied by the application instead
7376 of returning a pointer to a string. This allows a cleaner function in some implementations (such
7377 as those with lightweight threads) and resolves questions about when the application must copy
7378 the string returned.

7379 **FUTURE DIRECTIONS**

7380 None.

7381 **SEE ALSO**

7382 *pathconf()*, *sysconf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>, the Shell
7383 and Utilities volume of IEEE Std. 1003.1-200x, *c99*

7384 **CHANGE HISTORY**

7385 First released in Issue 4. Derived from the ISO POSIX-2 standard.

7386 **Issue 5**

7387 A table indicating the permissible values of *name* are added to the DESCRIPTION. All those
7388 marked EX are new in this issue.

7389 **Issue 6**

7390 The Open Group corrigenda item U033/7 has been applied. The return value for the case
7391 returning the size of the buffer now explicitly states that this includes the terminating null.

7392 The following new requirements on POSIX implementations derive from alignment with the
7393 Single UNIX Specification:

- 7394 • The DESCRIPTION is updated with new arguments which can be used to determine
7395 configuration strings for C compiler flags, linker/loader flags, and libraries for each different
7396 supported programming environment. This is a change to support data size neutrality.

7397 The following changes were made to align with the IEEE P1003.1a draft standard:

- 7398 • The DESCRIPTION is updated to include text describing how `_CS_PATH` can be used to
7399 obtain a *PATH* to access the standard utilities.

7400 The macros associated with the *c89* programming models are marked LEGACY and new
7401 equivalent macros associated with *c99* are introduced.

7402 **NAME**

7403 conj, conjf, conjl — complex conjugate functions

7404 **SYNOPSIS**

7405 #include <complex.h>

7406 double complex conj(double complex *z*);7407 float complex conjf(float complex *z*);7408 long double complex conjl(long double complex *z*);7409 **DESCRIPTION**

7410 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7411 conflict between the requirements described here and the ISO C standard is unintentional. This
7412 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

7413 These functions shall compute the complex conjugate of *z*, by reversing the sign of its imaginary
7414 part.

7415 **RETURN VALUE**

7416 These functions return the complex conjugate value.

7417 **ERRORS**

7418 No errors are defined.

7419 **EXAMPLES**

7420 None.

7421 **APPLICATION USAGE**

7422 None.

7423 **RATIONALE**

7424 None.

7425 **FUTURE DIRECTIONS**

7426 None.

7427 **SEE ALSO**

7428 *carg()*, *cimag()*, *cproj()*, *creal()*, the Base Definitions volume of IEEE Std. 1003.1-200x,
7429 <complex.h>

7430 **CHANGE HISTORY**

7431 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7432 **NAME**

7433 connect — connect a socket

7434 **SYNOPSIS**

7435 #include <sys/socket.h>

7436 int connect(int *socket*, const struct sockaddr **address*,
7437 socklen_t *address_len*);7438 **DESCRIPTION**7439 The *connect()* function requests a connection to be made on a socket. The function takes the
7440 following arguments:

7441 *socket* Specifies the file descriptor associated with the socket.

7442 *address* Points to a **sockaddr** structure containing the peer address. The length and
7443 format of the address depend on the address family of the socket.

7444 *address_len* Specifies the length of the **sockaddr** structure pointed to by the *address*
7445 argument.

7446 If the socket has not already been bound to a local address, *connect()* shall bind it to an address
7447 which, unless the socket's address family is AF_UNIX, is an unused local address.7448 If the initiating socket is not connection-mode, then *connect()* shall set the socket's peer address,
7449 and no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all
7450 datagrams are sent on subsequent *send()* functions, and limits the remote sender for subsequent
7451 *recv()* functions. If *address* is a null address for the protocol, the socket's peer address shall be
7452 reset.7453 If the initiating socket is connection-mode, then *connect()* attempts to establish a connection to
7454 the address specified by the *address* argument.7455 If the connection cannot be established immediately and O_NONBLOCK is not set for the file
7456 descriptor for the socket, *connect()* shall block for up to an unspecified timeout interval until the
7457 connection is established. If the timeout interval expires before the connection is established,
7458 *connect()* shall fail and the connection attempt shall be aborted. If *connect()* is interrupted by a
7459 signal that is caught while blocked waiting to establish a connection, *connect()* shall fail and set
7460 *errno* to [EINTR], but the connection request shall not be aborted, and the connection shall be
7461 established asynchronously.7462 If the connection cannot be established immediately and O_NONBLOCK is set for the file
7463 descriptor for the socket, *connect()* shall fail and set *errno* to [EINPROGRESS], but the connection
7464 request shall not be aborted, and the connection shall be established asynchronously.
7465 Subsequent calls to *connect()* for the same socket, before the connection is established, shall fail
7466 and set *errno* to [EALREADY].7467 When the connection has been established asynchronously, *select()* and *poll()* shall indicate that
7468 the file descriptor for the socket is ready for writing.7469 The socket in use may require the process to have appropriate privileges to use the *connect()*
7470 function.7471 **RETURN VALUE**7472 Upon successful completion, *connect()* shall return 0; otherwise, -1 shall be returned and *errno*
7473 set to indicate the error.

7474 **ERRORS**

- 7475 The *connect()* function shall fail if:
- 7476 [EADDRNOTAVAIL]
7477 The specified address is not available from the local machine.
- 7478 [EAFNOSUPPORT]
7479 The specified address is not a valid address for the address family of the
7480 specified socket.
- 7481 [EALREADY] A connection request is already in progress for the specified socket.
- 7482 [EBADF] The *socket* argument is not a valid file descriptor.
- 7483 [ECONNREFUSED]
7484 The target address was not listening for connections or refused the connection
7485 request.
- 7486 [EINPROGRESS] O_NONBLOCK is set for the file descriptor for the socket and the connection
7487 cannot be immediately established; the connection shall be established
7488 asynchronously.
- 7489 [EINTR] The attempt to establish a connection was interrupted by delivery of a signal
7490 that was caught; the connection shall be established asynchronously.
- 7491 [EISCONN] The specified socket is connection-mode and is already connected.
- 7492 [ENETUNREACH]
7493 No route to the network is present.
- 7494 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 7495 [EPROTOTYPE] The specified address has a different type than the socket bound to the
7496 specified peer address.
- 7497 [ETIMEDOUT] The attempt to connect timed out before a connection was made.
- 7498 If the address family of the socket is AF_UNIX, then *connect()* shall fail if:
- 7499 [EIO] An I/O error occurred while reading from or writing to the file system.
- 7500 [ELOOP] A loop exists in symbolic links encountered during resolution of the path
7501 name in *address*.
- 7502 [ENAMETOOLONG]
7503 A component of a path name exceeded {NAME_MAX} characters, or an entire
7504 path name exceeded {PATH_MAX} characters.
- 7505 [ENOENT] A component of the path name does not name an existing file or the path
7506 name is an empty string.
- 7507 [ENOTDIR] A component of the path prefix of the path name in *address* is not a directory.
- 7508 The *connect()* function may fail if:
- 7509 [EACCES] Search permission is denied for a component of the path prefix; or write
7510 access to the named socket is denied.
- 7511 [EADDRINUSE] Attempt to establish a connection that uses addresses that are already in use.
- 7512 [ECONNRESET] Remote host reset the connection request.
- 7513 [EHOSTUNREACH]
7514 The destination host cannot be reached (probably because the host is down or

7515 a remote router cannot reach it).

7516 [EINVAL] The *address_len* argument is not a valid length for the address family; or
7517 invalid address family in the **sockaddr** structure.

7518 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
7519 resolution of the path name in *address*.

7520 [ENAMETOOLONG]
7521 Path name resolution of a symbolic link produced an intermediate result
7522 whose length exceeds {PATH_MAX}.

7523 [ENETDOWN] The local network interface used to reach the destination is down.

7524 [ENOBUFS] No buffer space is available.

7525 [EOPNOTSUPP] The socket is listening and cannot be connected.

7526 **EXAMPLES**

7527 None.

7528 **APPLICATION USAGE**

7529 If *connect()* fails, the state of the socket is unspecified. Portable applications should close the file
7530 descriptor and create a new socket before attempting to reconnect.

7531 **RATIONALE**

7532 None.

7533 **FUTURE DIRECTIONS**

7534 None.

7535 **SEE ALSO**

7536 *accept()*, *bind()*, *close()*, *getsockname()*, *poll()*, *select()*, *send()*, *shutdown()*, *socket()*, the Base
7537 Definitions volume of IEEE Std. 1003.1-200x, <sys/socket.h>

7538 **CHANGE HISTORY**

7539 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7540 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
7541 [ELOOP] error condition is added.

7542 **NAME**

7543 copysign, copysignf, copysignl — number manipulation function

7544 **SYNOPSIS**

7545 #include <math.h>

7546 double copysign(double x, double y);

7547 float copysignf(float x, float y);

7548 long double copysignl(long double x, long double y);

7549 **DESCRIPTION**

7550 cx The functionality described on this reference page is aligned with the ISO C standard. Any
7551 conflict between the requirements described here and the ISO C standard is unintentional. This
7552 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

7553 These functions shall produce a value with the magnitude of *x* and the sign of *y*. They produce a
7554 NaN (with the sign of *y*) if *x* is a NaN. On implementations that represent a signed zero but do
7555 not treat negative zero consistently in arithmetic operations, these functions regard the sign of
7556 zero as positive.

7557 An application wishing to check for error situations should set *errno* to 0 before calling these
7558 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

7559 **RETURN VALUE**

7560 Upon successful completion, these functions shall return a value with the magnitude of *x* and
7561 the sign of *y*.

7562 If *x* is $\pm\text{Inf}$, these functions shall return *x*.

7563 If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

7564 **ERRORS**

7565 These functions may fail if:

7566 [EDOM] The value of *x* is NaN.

7567 **EXAMPLES**

7568 None.

7569 **APPLICATION USAGE**

7570 None.

7571 **RATIONALE**

7572 *copysign()* and *signbit()* need not be consistent with each other if the arithmetic is not consistent
7573 in its treatment of zeros. For example, the IBM S/370 has instructions to flip the sign bit making
7574 it possible to create a negative zero, but $\pm 0.0 \times \pm 1.0$ is always $+0.0$. In this case, *copysign()* will
7575 treat 0.0 as positive, while *signbit()* will treat it as negative.

7576 **FUTURE DIRECTIONS**

7577 None.

7578 **SEE ALSO**

7579 *signbit()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

7580 **CHANGE HISTORY**

7581 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7582 **NAME**

7583 cos, cosf, cosl — cosine function

7584 **SYNOPSIS**

7585 #include <math.h>

7586 double cos(double x);

7587 float cosf(float x);

7588 long double cosl(long double x);

7589 **DESCRIPTION**

7590 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 7591 conflict between the requirements described here and the ISO C standard is unintentional. This
 7592 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

7593 These functions shall compute the cosine of x , measured in radians.

7594 An application wishing to check for error situations should set *errno* to 0 before calling *cos()*. If
 7595 *errno* is non-zero on return, or the returned value is NaN, an error has occurred.

7596 **RETURN VALUE**7597 Upon successful completion, these functions shall return the cosine of x .7598 **XSI** If x is NaN, NaN shall be returned and *errno* may be set to [EDOM].

7599 **XSI** If x is $\pm\text{Inf}$, either 0 shall be returned and *errno* set to [EDOM], or NaN shall be returned and *errno*
 7600 may be set to [EDOM].

7601 If the result underflows, 0 shall be returned and *errno* may be set to [ERANGE].7602 **ERRORS**

7603 These functions may fail if:

7604 **XSI** [EDOM] The value of x is NaN or x is $\pm\text{Inf}$.

7605 [ERANGE] The result underflows

7606 **XSI** No other errors shall occur.7607 **EXAMPLES**7608 **Taking the Cosine of a 45-Degree Angle**

7609 #include <math.h>

7610 ...

7611 double radians = 45 * M_PI / 180;

7612 double result;

7613 ...

7614 result = cos(radians);

7615 **APPLICATION USAGE**7616 The *cos()* function may lose accuracy when its argument is far from 0.7617 **RATIONALE**

7618 None.

7619 **FUTURE DIRECTIONS**

7620 None.

7621 **SEE ALSO**

7622 *acos()*, *isnan()*, *sin()*, *tan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

7623 **CHANGE HISTORY**

7624 First released in Issue 1. Derived from Issue 1 of the SVID.

7625 **Issue 4**

7626 References to *matherr()* are removed.

7627 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalize error handling in the mathematics functions.

7629 The return value specified for [EDOM] is marked as an extension.

7630 **Issue 5**

7631 The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

7633 **Issue 6**

7634 The *cosf()* and *cosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

7635 **NAME**

7636 cosh, coshf, coshl — hyperbolic cosine function

7637 **SYNOPSIS**

7638 #include <math.h>

7639 double cosh(double x);

7640 float coshf(float x);

7641 long double coshl(long double x);

7642 **DESCRIPTION**

7643 cx The functionality described on this reference page is aligned with the ISO C standard. Any
7644 conflict between the requirements described here and the ISO C standard is unintentional. This
7645 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

7646 These functions shall compute the hyperbolic cosine of x .

7647 An application wishing to check for error situations should set *errno* to 0 before calling *cosh()*. If
7648 *errno* is non-zero on return, or the returned value is NaN, an error has occurred.

7649 **RETURN VALUE**7650 Upon successful completion, these functions shall return the hyperbolic cosine of x .7651 If the result would cause an overflow, HUGE_VAL shall be returned and *errno* set to [ERANGE].7652 xSI If x is NaN, NaN shall be returned and *errno* may be set to [EDOM].7653 **ERRORS**

7654 These functions shall fail if:

7655 [ERANGE] The result would cause an overflow.

7656 These functions may fail if:

7657 xSI [EDOM] The value of x is NaN.

7658 xSI No other errors shall occur.

7659 **EXAMPLES**

7660 None.

7661 **APPLICATION USAGE**

7662 None.

7663 **RATIONALE**

7664 None.

7665 **FUTURE DIRECTIONS**

7666 None.

7667 **SEE ALSO**7668 *acosh()*, *isnan()*, *sinh()*, *tanh()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>7669 **CHANGE HISTORY**

7670 First released in Issue 1. Derived from Issue 1 of the SVID.

7671 **Issue 4**7672 References to *matherr()* are removed.

7673 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the
7674 ISO C standard and to rationalize error handling in the mathematics functions.

- 7675 The return value specified for [EDOM] is marked as an extension.
- 7676 **Issue 5**
- 7677 The DESCRIPTION is updated to indicate how an application should check for an error. This
- 7678 text was previously published in the APPLICATION USAGE section.
- 7679 **Issue 6**
- 7680 The *coshf()* and *coshl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

7681 **NAME**

7682 cpow, cpowf, cpowl — complex power functions

7683 **SYNOPSIS**

7684 #include <complex.h>

7685 double complex cpow(double complex *x*, double complex *y*);7686 float complex cpowf(float complex *x*, float complex *y*);7687 long double complex cpowl(long double complex *x*,7688 long double complex *y*);7689 **DESCRIPTION**7690 *CX* The functionality described on this reference page is aligned with the ISO C standard. Any
7691 conflict between the requirements described here and the ISO C standard is unintentional. This
7692 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.7693 These functions shall compute the complex power function x^y , with a branch cut for the first
7694 parameter along the negative real axis.7695 **RETURN VALUE**

7696 These functions shall return the complex power function value.

7697 **ERRORS**

7698 No errors are defined.

7699 **EXAMPLES**

7700 None.

7701 **APPLICATION USAGE**

7702 None.

7703 **RATIONALE**

7704 None.

7705 **FUTURE DIRECTIONS**

7706 None.

7707 **SEE ALSO**7708 *cabs()*, *csqrt()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>7709 **CHANGE HISTORY**

7710 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7711 **NAME**

7712 cproj, cprojf, cprojl — complex projection functions

7713 **SYNOPSIS**

7714 #include <complex.h>

7715 double complex cproj(double complex z);

7716 float complex cprojf(float complex z);

7717 long double complex cprojl(long double complex z);

7718 **DESCRIPTION**

7719 cx The functionality described on this reference page is aligned with the ISO C standard. Any
7720 conflict between the requirements described here and the ISO C standard is unintentional. This
7721 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

7722 These functions shall compute a projection of z onto the Riemann sphere: z projects to z , except
7723 that all complex infinities (even those with one infinite part and one NaN part) project to
7724 positive infinity on the real axis. If z has an infinite part, then $cproj(z)$ is equivalent to:

7725 $\text{INFINITY} + \text{I} * \text{copysign}(0.0, \text{cimag}(z))$ 7726 **RETURN VALUE**

7727 These functions shall return the value of the projection onto the Riemann sphere.

7728 **ERRORS**

7729 No errors are defined.

7730 **EXAMPLES**

7731 None.

7732 **APPLICATION USAGE**

7733 None.

7734 **RATIONALE**

7735 Two topologies are commonly used in complex mathematics: the complex plane with its
7736 continuum of infinities, and the Riemann sphere with its single infinity. The complex plane is
7737 better suited for transcendental functions, the Riemann sphere for algebraic functions. The
7738 complex types with their multiplicity of infinities provide a useful (though imperfect) model for
7739 the complex plane. The $cproj()$ function helps model the Riemann sphere by mapping all
7740 infinities to one, and should be used just before any operation, especially comparisons, that
7741 might give spurious results for any of the other infinities. Note that a complex value with one
7742 infinite part and one NaN part is regarded as an infinity, not a NaN, because if one part is
7743 infinite, the complex value is infinite independent of the value of the other part. For the same
7744 reason, $cabs()$ returns an infinity if its argument has an infinite part and a NaN part.

7745 **FUTURE DIRECTIONS**

7746 None.

7747 **SEE ALSO**

7748 $carg()$, $cimag()$, $conj()$, $creal()$, the Base Definitions volume of IEEE Std. 1003.1-200x,
7749 <complex.h>

7750 **CHANGE HISTORY**

7751 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7752 **NAME**

7753 creal, crealf, creall — complex real functions

7754 **SYNOPSIS**

7755 #include <complex.h>

7756 double creal(double complex *z*);7757 float crealf(float complex *z*);7758 long double creall(long double complex *z*);7759 **DESCRIPTION**

7760 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7761 conflict between the requirements described here and the ISO C standard is unintentional. This
7762 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

7763 These functions shall compute the real part of *z*.7764 **RETURN VALUE**

7765 These functions shall return the real part value.

7766 **ERRORS**

7767 No errors are defined.

7768 **EXAMPLES**

7769 None.

7770 **APPLICATION USAGE**7771 For a variable *z* of complex type:7772 `z == creal(z) + cimag(z)*I`7773 **RATIONALE**

7774 None.

7775 **FUTURE DIRECTIONS**

7776 None.

7777 **SEE ALSO**

7778 `carg()`, `cimag()`, `conj()`, `cproj()`, the Base Definitions volume of IEEE Std. 1003.1-200x,
7779 `<complex.h>`

7780 **CHANGE HISTORY**

7781 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7782 **NAME**

7783 creat — create a new file or rewrite an existing one

7784 **SYNOPSIS**

7785 OH #include <sys/stat.h>

7786 #include <fcntl.h>

7787 int creat(const char *path, mode_t mode);

7788 **DESCRIPTION**

7789 The function call:

7790 creat(path, mode)

7791 is equivalent to:

7792 open(path, O_WRONLY|O_CREAT|O_TRUNC, mode)

7793 **RETURN VALUE**7794 Refer to *open()*.7795 **ERRORS**7796 Refer to *open()*.7797 **EXAMPLES**7798 **Creating a File**7799 The following example creates the file **/tmp/file** with read and write permissions for the file
7800 owner and read permission for group and others. The resulting file descriptor is assigned to the
7801 *fd* variable.

7802 #include <fcntl.h>

7803 ...

7804 int fd;

7805 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;

7806 char *filename = "/tmp/file";

7807 ...

7808 fd = creat(filename, mode);

7809 ...

7810 **APPLICATION USAGE**

7811 None.

7812 **RATIONALE**7813 The *creat()* function is redundant. Its services are also provided by the *open()* function. It has
7814 been included primarily for historical purposes since many existing applications depend on it. It
7815 is best considered a part of the C binding rather than a function that should be provided in other
7816 languages.7817 **FUTURE DIRECTIONS**

7818 None.

7819 **SEE ALSO**7820 *open()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <fcntl.h>, <sys/stat.h>, |
7821 <sys/types.h>

7822 **CHANGE HISTORY**

7823 First released in Issue 1. Derived from Issue 1 of the SVID.

7824 **Issue 4**7825 The `<sys/types.h>` and `<sys/stat.h>` headers are now marked as optional (OH); these headers
7826 need not be included on XSI-conformant systems.

7827 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 7828
- The type of argument *path* is changed from `char*` to `const char*`.

7829 **Issue 6**7830 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.7831 The following new requirements on POSIX implementations derive from alignment with the
7832 Single UNIX Specification:

- 7833
- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
7834 required for conforming implementations of previous POSIX specifications, it was not
7835 required for UNIX applications.

7836 **NAME**7837 crypt — string encoding function (**CRYPT**)7838 **SYNOPSIS**7839 XSI `#include <unistd.h>`7840 `char *crypt(const char *key, const char *salt);`

7841

7842 **DESCRIPTION**7843 The *crypt()* function is a string encoding function. The algorithm is implementation-defined.7844 The *key* argument points to a string to be encoded. The *salt* argument is a string chosen from the
7845 set:

7846 a b c d e f g h i j k l m n o p q r s t u v w x y z

7847 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

7848 0 1 2 3 4 5 6 7 8 9 . /

7849 The first two characters of this string may be used to perturb the encoding algorithm.

7850 The return value of *crypt()* points to static data that is overwritten by each call.7851 The *crypt()* function need not be reentrant. A function that is not required to be reentrant is not
7852 required to be thread-safe.7853 **RETURN VALUE**7854 Upon successful completion, *crypt()* shall return a pointer to the encoded string. The first two
7855 characters of the returned value are those of the *salt* argument. Otherwise, it shall return a null
7856 pointer and set *errno* to indicate the error.7857 **ERRORS**7858 The *crypt()* function shall fail if:

7859 [ENOSYS] The functionality is not supported on this implementation.

7860 **EXAMPLES**7861 **Encoding Passwords**7862 The following example finds a user database entry matching a particular user name and changes
7863 the current password to a new password. The *crypt()* function is used to generate an encoded
7864 version of each password. The first call to *crypt()* produces an encoded version of the old
7865 password; that encoded password is then compared to the password stored in the user database.
7866 The second call to *crypt()* encodes the new password before it is stored.7867 The *putpwent()* function, used in the following example, is not part of IEEE Std. 1003.1-200x.7868 `#include <unistd.h>`7869 `#include <pwd.h>`7870 `#include <string.h>`7871 `#include <stdio.h>`7872 `...`7873 `int valid_change;`7874 `int pfd; /* Integer for file descriptor returned by open(). */`7875 `FILE *fpfd; /* File pointer for use in putpwent(). */`7876 `struct passwd *p;`7877 `char user[100];`7878 `char oldpasswd[100];`7879 `char newpasswd[100];`

```

7880     char savepasswd[100];
7881     ...
7882     valid_change = 0;
7883     while ((p = getpwent()) != NULL) {
7884         /* Change entry if found. */
7885         if (strcmp(p->pw_name, user) == 0) {
7886             if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
7887                 strcpy(savepasswd, crypt(newpasswd, user));
7888                 p->pw_passwd = savepasswd;
7889                 valid_change = 1;
7890             }
7891             else {
7892                 fprintf(stderr, "Old password is not valid\n");
7893             }
7894         }
7895         /* Put passwd entry into ptmp. */
7896         putpwent(p, fpfd);
7897     }

```

7898 APPLICATION USAGE

7899 The values returned by this function need not be portable among XSI-conformant systems.

7900 RATIONALE

7901 None.

7902 FUTURE DIRECTIONS

7903 None.

7904 SEE ALSO

7905 *encrypt()*, *setkey()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>

7906 CHANGE HISTORY

7907 First released in Issue 1. Derived from Issue 1 of the SVID.

7908 Issue 4

7909 The <**unistd.h**> header is added to the SYNOPSIS section.

7910 The type of arguments *key* and *salt* are changed from **char*** to **const char***.

7911 The DESCRIPTION now explicitly defines the characters that can appear in the *salt* argument.

7912 Issue 5

7913 Normative text previously in the APPLICATION USAGE section is moved to the
7914 DESCRIPTION.

7915 **NAME**

7916 csin, csinf, csinl — complex sine functions

7917 **SYNOPSIS**

7918 #include <complex.h>

7919 double complex csin(double complex *z*);7920 float complex csinf(float complex *z*);7921 long double complex csinl(long double complex *z*);7922 **DESCRIPTION**7923 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7924 conflict between the requirements described here and the ISO C standard is unintentional. This
7925 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.7926 These functions shall compute the complex sine of *z*.7927 **RETURN VALUE**

7928 These functions shall return the complex sine value.

7929 **ERRORS**

7930 No errors are defined.

7931 **EXAMPLES**

7932 None.

7933 **APPLICATION USAGE**

7934 None.

7935 **RATIONALE**

7936 None.

7937 **FUTURE DIRECTIONS**

7938 None.

7939 **SEE ALSO**7940 *casin()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>7941 **CHANGE HISTORY**

7942 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7943 **NAME**

7944 csinh, csinhf, csinhl — complex hyperbolic sine functions

7945 **SYNOPSIS**

7946 #include <complex.h>

7947 double complex csinh(double complex *z*);7948 float complex csinhf(float complex *z*);7949 long double complex csinhl(long double complex *z*);7950 **DESCRIPTION**7951 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7952 conflict between the requirements described here and the ISO C standard is unintentional. This
7953 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.7954 These functions shall compute the complex hyperbolic sine of *z*.7955 **RETURN VALUE**

7956 These functions shall return the complex hyperbolic sine value.

7957 **ERRORS**

7958 No errors are defined.

7959 **EXAMPLES**

7960 None.

7961 **APPLICATION USAGE**

7962 None.

7963 **RATIONALE**

7964 None.

7965 **FUTURE DIRECTIONS**

7966 None.

7967 **SEE ALSO**7968 *casinh()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>7969 **CHANGE HISTORY**

7970 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7971 **NAME**

7972 csqrt, csqrtf, csqrtl — complex square root functions

7973 **SYNOPSIS**

7974 #include <complex.h>

7975 double complex csqrt(double complex z);

7976 float complex csqrtf(float complex z);

7977 long double complex csqrtl(long double complex z);

7978 **DESCRIPTION**

7979 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7980 conflict between the requirements described here and the ISO C standard is unintentional. This
7981 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

7982 These functions shall compute the complex square root of z , with a branch cut along the
7983 negative real axis.

7984 **RETURN VALUE**

7985 These functions shall return the complex square root value, in the range of the right half-plane
7986 (including the imaginary axis).

7987 **ERRORS**

7988 No errors are defined.

7989 **EXAMPLES**

7990 None.

7991 **APPLICATION USAGE**

7992 None.

7993 **RATIONALE**

7994 None.

7995 **FUTURE DIRECTIONS**

7996 None.

7997 **SEE ALSO**7998 *cabs()*, *cpow()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>7999 **CHANGE HISTORY**

8000 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

8001 **NAME**

8002 ctan, ctanf, ctanl — complex tangent functions

8003 **SYNOPSIS**

8004 #include <complex.h>

8005 double complex ctan(double complex *z*);8006 float complex ctanf(float complex *z*);8007 long double complex ctanl(long double complex *z*);8008 **DESCRIPTION**

8009 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
8010 conflict between the requirements described here and the ISO C standard is unintentional. This
8011 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

8012 These functions shall compute the complex tangent of *z*.8013 **RETURN VALUE**

8014 These functions shall return the complex tangent value.

8015 **ERRORS**

8016 No errors are defined.

8017 **EXAMPLES**

8018 None.

8019 **APPLICATION USAGE**

8020 None.

8021 **RATIONALE**

8022 None.

8023 **FUTURE DIRECTIONS**

8024 None.

8025 **SEE ALSO**8026 *catan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**complex.h**>8027 **CHANGE HISTORY**

8028 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

8029 **NAME**

8030 ctanh, ctanhf, ctanhl — complex hyperbolic tangent functions

8031 **SYNOPSIS**

8032 #include <complex.h>

8033 double complex ctanh(double complex *z*);8034 float complex ctanhf(float complex *z*);8035 long double complex ctanhl(long double complex *z*);8036 **DESCRIPTION**8037 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
8038 conflict between the requirements described here and the ISO C standard is unintentional. This
8039 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.8040 These functions shall compute the complex hyperbolic tangent of *z*.8041 **RETURN VALUE**

8042 These functions shall return the complex hyperbolic tangent value.

8043 **ERRORS**

8044 No errors are defined.

8045 **EXAMPLES**

8046 None.

8047 **APPLICATION USAGE**

8048 None.

8049 **RATIONALE**

8050 None.

8051 **FUTURE DIRECTIONS**

8052 None.

8053 **SEE ALSO**8054 *catanh()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <complex.h>8055 **CHANGE HISTORY**

8056 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

8057 **NAME**

8058 ctermid — generate a path name for controlling terminal

8059 **SYNOPSIS**

8060 #include <stdio.h>

8061 char *ctermid(char *s);

8062 **DESCRIPTION**

8063 The *ctermid()* function shall generate a string that, when used as a path name, refers to the
8064 current controlling terminal for the current process. If *ctermid()* returns a path name, access to
8065 the file is not guaranteed.

8066 If the application uses any of the `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS`
8067 functions, it shall ensure that the *ctermid()* function is called with a non-NULL parameter.

8068 **RETURN VALUE**

8069 If *s* is a null pointer, the string is generated in an area that may be static (and therefore may be
8070 overwritten by each call), the address of which shall be returned. Otherwise, *s* is assumed to
8071 point to a character array of at least `{L_ctermid}` bytes; the string is placed in this array and the
8072 value of *s* shall be returned. The symbolic constant `{L_ctermid}` is defined in `<stdio.h>`, and shall
8073 have a value greater than 0.

8074 The *ctermid()* function shall return an empty string if the path name that would refer to the
8075 controlling terminal cannot be determined, or if the function is unsuccessful.

8076 **ERRORS**

8077 No errors are defined.

8078 **EXAMPLES**8079 **Determining the Controlling Terminal for the Current Process**

8080 The following example returns a pointer to a string that identifies the controlling terminal for the
8081 current process. The path name for the terminal is stored in the array pointed to by the *ptr*
8082 argument, which has a size of `{L_ctermid}` bytes, as indicated by the *term* argument.

8083 #include <stdio.h>

8084 ...

8085 char term[L_ctermid];

8086 char *ptr;

8087 ptr = ctermid(term);

8088 **APPLICATION USAGE**

8089 The difference between *ctermid()* and *ttyname()* is that *ttyname()* must be handed a file
8090 descriptor and return a path of the terminal associated with that file descriptor, while *ctermid()*
8091 returns a string (such as `"/dev/tty"`) that refers to the current controlling terminal if used as a
8092 path name.

8093 **RATIONALE**

8094 `{L_ctermid}` must be defined appropriately for a given implementation and must be greater than
8095 zero so that array declarations using it are accepted by the compiler. The value includes the
8096 terminating null byte.

8097 Portable applications that use threads cannot call *ctermid()* with NULL as the parameter if either
8098 `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS` is defined. If *s* is not NULL, the
8099 *ctermid()* function generates a string that, when used as a path name, refers to the current
8100 controlling terminal for the current process. If *s* is NULL, the return value of *ctermid()* is

8101 undefined.

8102 If the *ctermid()* function returns a path name, access to the file is not guaranteed.

8103 There is no additional burden on the programmer—changing to use a hypothetical thread-safe
8104 version of *ctermid()* along with allocating a buffer is more of a burden than merely allocating a
8105 buffer. Application code should not assume that the returned string is short, as some
8106 implementations have more than two path name components before reaching a logical device
8107 name.

8108 **FUTURE DIRECTIONS**

8109 None.

8110 **SEE ALSO**

8111 *ttyname()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>

8112 **CHANGE HISTORY**

8113 First released in Issue 1. Derived from Issue 1 of the SVID.

8114 **Issue 4**

8115 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 8116 • The DESCRIPTION and RETURN VALUE sections, though functionally identical to Issue 3,
8117 are rewritten.

8118 **Issue 5**

8119 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8120 **Issue 6**

8121 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

8122 **NAME**

8123 ctime, ctime_r — convert a time value to date and time string

8124 **SYNOPSIS**

8125 #include <time.h>

8126 char *ctime(const time_t *clock);

8127 TSF char *ctime_r(const time_t *clock, char *buf);

8128

8129 **DESCRIPTION**8130 CX The functionality described on this reference page is aligned with the ISO C standard. Any
8131 conflict between the requirements described here and the ISO C standard is unintentional. This
8132 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.8133 The *ctime()* function shall convert the time pointed to by *clock*, representing time in seconds
8134 since the Epoch, to local time in the form of a string. It is equivalent to:

8135 asctime(localtime(clock))

8136 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions return values in one of two static
8137 objects: a broken-down time structure and an array of **char**. Execution of any of the functions
8138 may overwrite the information returned in either of these objects by any of the other functions.8139 The *ctime()* function need not be reentrant. A function that is not required to be reentrant is not
8140 required to be thread-safe.8141 TSF The *ctime_r()* function shall convert the calendar time pointed to by *clock* to local time in exactly
8142 the same form as *ctime()* and puts the string into the array pointed to by *buf* (which contains at
8143 least 26 bytes) and return *buf*.8144 Unlike *ctime()*, the thread-safe version *ctime_r()* is not required to set *tzname*.8145 **RETURN VALUE**8146 The *ctime()* function shall return the pointer returned by *asctime()* with that broken-down time
8147 as an argument.8148 TSF Upon successful completion, *ctime_r()* shall return a pointer to the string pointed to by *buf*.
8149 When an error is encountered, a null pointer shall be returned.8150 **ERRORS**

8151 No errors are defined.

8152 **EXAMPLES**

8153 None.

8154 **APPLICATION USAGE**8155 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.
8156 The *ctime()* function is included for compatibility with older implementations, and does not
8157 support localized date and time formats. Applications should use the *strptime()* function to
8158 achieve maximum portability.8159 The *ctime_r()* function is thread-safe and shall return values in a user-supplied buffer instead of
8160 possibly using a static data area that may be overwritten by each call.8161 **RATIONALE**

8162 None.

8163 **FUTURE DIRECTIONS**

8164 None.

8165 **SEE ALSO**8166 *asctime()*, *clock()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,
8167 the Base Definitions volume of IEEE Std. 1003.1-200x, <**time.h**>8168 **CHANGE HISTORY**

8169 First released in Issue 1. Derived from Issue 1 of the SVID.

8170 **Issue 4**8171 The APPLICATION USAGE section is expanded to describe the time-handling functions
8172 generally and to refer users to *strftime()*, which is a locale-dependent time-handling function.

8173 The following change is incorporated for alignment with the ISO C standard:

- 8174
- The type of argument *clock* is changed from **time_t*** to **const time_t***.

8175 **Issue 5**8176 Normative text previously in the APPLICATION USAGE section is moved to the
8177 DESCRIPTION.8178 The *ctime_r()* function is included for alignment with the POSIX Threads Extension.8179 A note indicating that the *ctime()* function need not be reentrant is added to the DESCRIPTION.8180 **Issue 6**

8181 Extensions beyond the ISO C standard are now marked.

8182 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8183 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
8184 its avoidance of possibly using a static data area.

8185 **NAME**

8186 daylight — daylight savings time flag

8187 **SYNOPSIS**

8188 xSI #include <time.h>

8189 extern int daylight;

8190

8191 **DESCRIPTION**

8192 Refer to *tzset()*.

8193 NAME

8194 dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey,
8195 dbm_open, dbm_store — database functions

8196 SYNOPSIS

```
8197 xSI #include <ndbm.h>

8198 int dbm_clearerr(DBM *db);
8199 void dbm_close(DBM *db);
8200 int dbm_delete(DBM *db, datum key);
8201 int dbm_error(DBM *db);
8202 datum dbm_fetch(DBM *db, datum key);
8203 datum dbm_firstkey(DBM *db);
8204 datum dbm_nextkey(DBM *db);
8205 DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
8206 int dbm_store(DBM *db, datum key, datum content, int store_mode);
8207
```

8208 DESCRIPTION

8209 These functions create, access, and modify a database.

8210 A **datum** consists of at least two members, *dptr* and *dsize*. The *dptr* member points to an object
8211 that is *dsize* bytes in length. Arbitrary binary data, as well as character strings, may be stored in
8212 the object pointed to by *dptr*.

8213 The database is stored in two files. One file is a directory containing a bit map of keys and has
8214 **.dir** as its suffix. The second file contains all data and has **.pag** as its suffix.

8215 The *dbm_open()* function shall open a database. The *file* argument to the function is the path
8216 name of the database. The function opens two files named *file.dir* and *file.pag*. The *open_flags*
8217 argument has the same meaning as the *flags* argument of *open()* except that a database opened
8218 for write-only access opens the files for read and write access and the behavior of the
8219 O_APPEND flag is unspecified. The *file_mode* argument has the same meaning as the third
8220 argument of *open()*.

8221 The *dbm_close()* function shall close a database. The application shall ensure that argument *db* is
8222 a pointer to a **dbm** structure that has been returned from a call to *dbm_open()*.

8223 The *dbm_fetch()* function shall read a record from a database. The argument *db* is a pointer to a
8224 database structure that has been returned from a call to *dbm_open()*. The argument *key* is a
8225 **datum** that has been initialized by the application to the value of the key that matches the key of
8226 the record the program is fetching.

8227 The *dbm_store()* function shall write a record to a database. The argument *db* is a pointer to a
8228 database structure that has been returned from a call to *dbm_open()*. The argument *key* is a
8229 **datum** that has been initialized by the application to the value of the key that identifies (for
8230 subsequent reading, writing, or deleting) the record the application is writing. The argument
8231 *content* is a **datum** that has been initialized by the application to the value of the record the
8232 program is writing. The argument *store_mode* controls whether *dbm_store()* replaces any pre-
8233 existing record that has the same key that is specified by the *key* argument. The application shall
8234 set *store_mode* to either DBM_INSERT or DBM_REPLACE. If the database contains a record that
8235 matches the *key* argument and *store_mode* is DBM_REPLACE, the existing record is replaced with
8236 the new record. If the database contains a record that matches the *key* argument and *store_mode*
8237 is DBM_INSERT, the existing record is left unchanged and the new record ignored. If the
8238 database does not contain a record that matches the *key* argument and *store_mode* is either
8239 DBM_INSERT or DBM_REPLACE, the new record is inserted in the database.

8240 The application shall ensure that the sum of the sizes of a key/content pair does not exceed the
8241 internal block size. Moreover, the application shall ensure that all key/content pairs that hash
8242 together fit on a single block. The *dbm_store()* function shall return an error in the event that a
8243 disk block fills with inseparable data.

8244 The *dbm_delete()* function shall delete a record and its key from the database. The argument *db* is
8245 a pointer to a database structure that has been returned from a call to *dbm_open()*. The argument
8246 *key* is a **datum** that has been initialized by the application to the value of the key that identifies
8247 the record the program is deleting.

8248 The *dbm_firstkey()* function shall return the first key in the database. The argument *db* is a
8249 pointer to a database structure that has been returned from a call to *dbm_open()*.

8250 The *dbm_nextkey()* function shall return the next key in the database. The argument *db* is a
8251 pointer to a database structure that has been returned from a call to *dbm_open()*. The application
8252 shall ensure that the *dbm_firstkey()* function is called before calling *dbm_nextkey()*. Subsequent
8253 calls to *dbm_nextkey()* return the next key until all of the keys in the database have been
8254 returned.

8255 The *dbm_error()* function shall return the error condition of the database. The argument *db* is a
8256 pointer to a database structure that has been returned from a call to *dbm_open()*.

8257 The *dbm_clearerr()* function shall clear the error condition of the database. The argument *db* is a
8258 pointer to a database structure that has been returned from a call to *dbm_open()*.

8259 These database functions shall support an internal block size large enough to support
8260 key/content pairs of at least 1 023 bytes.

8261 The *dptr* pointers returned by these functions may point into static storage that may be changed
8262 by subsequent calls.

8263 These functions need not be reentrant. A function that is not required to be reentrant is not
8264 required to be thread-safe.

8265 RETURN VALUE

8266 The *dbm_store()* and *dbm_delete()* functions shall return 0 when they succeed and a negative
8267 value when they fail.

8268 The *dbm_store()* function shall return 1 if it is called with a *flags* value of DBM_INSERT and the
8269 function finds an existing record with the same key.

8270 The *dbm_error()* function shall return 0 if the error condition is not set and return a non-zero
8271 value if the error condition is set.

8272 The return value of *dbm_clearerr()* is unspecified.

8273 The *dbm_firstkey()* and *dbm_nextkey()* functions shall return a key **datum**. When the end of the
8274 database is reached, the *dptr* member of the key is a null pointer. If an error is detected, the *dptr*
8275 member of the key shall be a null pointer and the error condition of the database shall be set.

8276 The *dbm_fetch()* function shall return a content **datum**. If no record in the database matches the
8277 key or if an error condition has been detected in the database, the *dptr* member of the content
8278 shall be a null pointer.

8279 The *dbm_open()* function shall return a pointer to a database structure. If an error is detected
8280 during the operation, *dbm_open()* shall return a **(DBM*)0**.

8281 ERRORS

8282 No errors are defined.

8283 EXAMPLES

8284 None.

8285 APPLICATION USAGE

8286 The following code can be used to traverse the database:

```
8287 for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

8288 The *dbm_* functions provided in this library should not be confused in any way with those of a
8289 general-purpose database management system. These functions do not provide for multiple
8290 search keys per entry, they do not protect against multi-user access (in other words they do not
8291 lock records or files), and they do not provide the many other useful database functions that are
8292 found in more robust database management systems. Creating and updating databases by use of
8293 these functions is relatively slow because of data copies that occur upon hash collisions. These
8294 functions are useful for applications requiring fast lookup of relatively static information that is
8295 to be indexed by a single key.

8296 The *dbm_delete()* function need not physically reclaim file space, although it does make it
8297 available for reuse by the database.

8298 After calling *dbm_store()* or *dbm_delete()* during a pass through the keys by *dbm_firstkey()* and
8299 *dbm_nextkey()*, the application should reset the database by calling *dbm_firstkey()* before again
8300 calling *dbm_nextkey()*. The contents of these files are unspecified and may not be portable.

8301 RATIONALE

8302 None.

8303 FUTURE DIRECTIONS

8304 None.

8305 SEE ALSO

8306 *open()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**ndbm.h**>

8307 CHANGE HISTORY

8308 First released in Issue 4, Version 2.

8309 Issue 5

8310 Moved from X/OPEN UNIX extension to BASE.

8311 Normative text previously in the APPLICATION USAGE section is moved to the
8312 DESCRIPTION.

8313 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

8314 Issue 6

8315 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

8316 **NAME**

8317 difftime — compute the difference between two calendar time values

8318 **SYNOPSIS**

8319 #include <time.h>

8320 double difftime(time_t *time1*, time_t *time0*);

8321 **DESCRIPTION**

8322 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
8323 conflict between the requirements described here and the ISO C standard is unintentional. This
8324 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

8325 The *difftime()* function shall compute the difference between two calendar times (as returned by
8326 *time()*): *time1*– *time0*.

8327 **RETURN VALUE**

8328 The *difftime()* function shall return the difference expressed in seconds as a type **double**.

8329 **ERRORS**

8330 No errors are defined.

8331 **EXAMPLES**

8332 None.

8333 **APPLICATION USAGE**

8334 None.

8335 **RATIONALE**

8336 None.

8337 **FUTURE DIRECTIONS**

8338 None.

8339 **SEE ALSO**

8340 *asctime()*, *clock()*, *ctime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,
8341 the Base Definitions volume of IEEE Std. 1003.1-200x, <**time.h**>

8342 **CHANGE HISTORY**

8343 First released in Issue 4. Derived from the ISO C standard.

8344 **NAME**

8345 dirname — report the parent directory name of a file path name

8346 **SYNOPSIS**

8347 XSI #include <libgen.h>

8348 char *dirname(char *path);

8349

8350 **DESCRIPTION**8351 The *dirname()* function shall take a pointer to a character string that contains a path name, and
8352 return a pointer to a string that is a path name of the parent directory of that file. Trailing '/'
8353 characters in the path are not counted as part of the path.8354 If *path* does not contain a '/', then *dirname()* shall return a pointer to the string ".". If *path* is a
8355 null pointer or points to an empty string, *dirname()* shall return a pointer to the string ".".8356 The *dirname()* function need not be reentrant. A function that is not required to be reentrant is
8357 not required to be thread-safe.8358 **RETURN VALUE**8359 The *dirname()* function shall return a pointer to a string that is the parent directory of *path*. If
8360 *path* is a null pointer or points to an empty string, a pointer to a string "." is returned.8361 The *dirname()* function may modify the string pointed to by *path*, and may return a pointer to
8362 static storage that may then be overwritten by subsequent calls to *dirname()*.8363 **ERRORS**

8364 No errors are defined.

8365 **EXAMPLES**8366 The following code fragment reads a path name, changes the current working directory to the
8367 parent directory, and opens the file.8368 char path[MAXPATHLEN], *pathcopy;
8369 int fd;
8370 fgets(path, MAXPATHLEN, stdin);
8371 pathcopy = strdup(path);
8372 chdir(dirname(pathcopy));
8373 fd = open(basename(path), O_RDONLY);8374 **Sample Input and Output Strings for dirname()**8375 In the following table, the input string is the value pointed to by *path*, and the output string is
8376 the return value of the *dirname()* function.

8377

8378

8379

8380

8381

8382

8383

Input String	Output String
"/usr/lib"	"/usr"
"/usr/"	"/"
"usr"	."
"/"	"/"
."	."
".."	."

8384 **Changing the Current Directory to the Parent Directory**

8385 The following program fragment reads a path name, changes the current working directory to
8386 the parent directory, and opens the file.

```
8387 #include <unistd.h>
8388 #include <limits.h>
8389 #include <stdio.h>
8390 #include <fcntl.h>
8391 #include <string.h>
8392 #include <libgen.h>
8393 ...
8394 char path[PATH_MAX], *pathcopy;
8395 int fd;
8396 ...
8397 fgets(path, PATH_MAX, stdin);
8398 pathcopy = strdup(path);
8399 chdir(dirname(pathcopy));
8400 fd = open(basename(path), O_RDONLY);
```

8401 **APPLICATION USAGE**

8402 The *dirname()* and *basename()* functions together yield a complete path name. The expression
8403 *dirname(path)* obtains the path name of the directory where *basename(path)* is found.

8404 Since the meaning of the leading *"/"* is implementation-defined, *dirname("/foo)* may return
8405 either *"/"* or *'/'* (but nothing else).

8406 **RATIONALE**

8407 None.

8408 **FUTURE DIRECTIONS**

8409 None.

8410 **SEE ALSO**

8411 *basename()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**libgen.h**>

8412 **CHANGE HISTORY**

8413 First released in Issue 4, Version 2.

8414 **Issue 5**

8415 Moved from X/OPEN UNIX extension to BASE.

8416 Normative text previously in the APPLICATION USAGE section is moved to the
8417 DESCRIPTION.

8418 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

8419 **NAME**

8420 `div` — compute the quotient and remainder of an integer division

8421 **SYNOPSIS**

8422 `#include <stdlib.h>`

8423 `div_t div(int numer, int denom);`

8424 **DESCRIPTION**

8425 `CX` The functionality described on this reference page is aligned with the ISO C standard. Any
8426 conflict between the requirements described here and the ISO C standard is unintentional. This
8427 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

8428 The `div()` function shall compute the quotient and remainder of the division of the numerator
8429 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the integer
8430 of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be
8431 represented, the behavior is undefined; otherwise, *quot*denom+rem* shall equal *numer*.

8432 **RETURN VALUE**

8433 The `div()` function shall return a structure of type `div_t`, comprising both the quotient and the
8434 remainder. The structure includes the following members, in any order:

8435 `int quot; /* quotient */`

8436 `int rem; /* remainder */`

8437 **ERRORS**

8438 No errors are defined.

8439 **EXAMPLES**

8440 None.

8441 **APPLICATION USAGE**

8442 None.

8443 **RATIONALE**

8444 None.

8445 **FUTURE DIRECTIONS**

8446 None.

8447 **SEE ALSO**

8448 `ldiv()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<stdlib.h>`

8449 **CHANGE HISTORY**

8450 First released in Issue 4. Derived from the ISO C standard.

8451 **NAME**8452 dldclose — close a *dlopen()* object8453 **SYNOPSIS**

```
8454 xSI       #include <dldfcn.h>
8455           int dldclose(void *handle);
8456
```

8457 **DESCRIPTION**

8458 The *dldclose()* function is used to inform the system that the object referenced by a *handle* returned
8459 from a previous *dlopen()* invocation is no longer needed by the application.

8460 The use of *dldclose()* reflects a statement of intent on the part of the process, but does not create
8461 any requirement upon the implementation, such as removal of the code or symbols referenced
8462 by *handle*. Once an object has been closed using *dldclose()* an application should assume that its
8463 symbols are no longer available to *dlsym()*. All objects loaded automatically as a result of
8464 invoking *dlopen()* on the referenced object are also closed if this is the last reference to it.

8465 Although a *dldclose()* operation is not required to remove structures from an address space,
8466 neither is an implementation prohibited from doing so. The only restriction on such a removal is
8467 that no object shall be removed to which references have been relocated, until or unless all such
8468 references are removed. For instance, an object that had been loaded with a *dlopen()* operation
8469 specifying the `RTLD_GLOBAL` flag might provide a target for dynamic relocations performed in
8470 the processing of other objects—in such environments, an application may assume that no
8471 relocation, once made, shall be undone or remade unless the object requiring the relocation has
8472 itself been removed.

8473 **RETURN VALUE**

8474 If the referenced object was successfully closed, *dldclose()* shall return 0. If the object could not be
8475 closed, or if *handle* does not refer to an open object, *dldclose()* shall return a non-zero value. More
8476 detailed diagnostic information shall be available through *dlderror()*.

8477 **ERRORS**

8478 No errors are defined.

8479 **EXAMPLES**8480 The following example illustrates use of *dlopen()* and *dldclose()*:

```
8481           ...
8482           /* Open a dynamic library and then close it ... */
8483           #include <dldfcn.h>
8484           void *mylib;
8485           int  eret;
8486           mylib = dlopen("mylib.so.1", RTLD_LAZY);
8487           ...
8488           eret = dldclose(mylib);
8489           ...
```

8490 **APPLICATION USAGE**

8491 A portable application should employ a *handle* returned from a *dlopen()* invocation only within a
8492 given scope bracketed by the *dlopen()* and *dldclose()* operations. Implementations are free to use
8493 reference counting or other techniques such that multiple calls to *dlopen()* referencing the same
8494 object may return the same object for *handle*. Implementations are also free to reuse a *handle*.
8495 For these reasons, the value of a *handle* must be treated as an opaque object by the application,
8496 used only in calls to *dlsym()* and *dldclose()*.

8497 **RATIONALE**

8498 None.

8499 **FUTURE DIRECTIONS**

8500 None.

8501 **SEE ALSO**8502 *derror()*, *dlopen()*, *dlsym()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**dlfcn.h**>8503 **CHANGE HISTORY**

8504 First released in Issue 5.

8505 **Issue 6**8506 The DESCRIPTION is updated to say that the referenced object is closed “if this is the last
8507 reference to it”.

8508 **NAME**8509 `dlderror` — get diagnostic information8510 **SYNOPSIS**8511 XSI `#include <dldfcn.h>`8512 `char *dlderror(void);`

8513

8514 **DESCRIPTION**

8515 The `dlderror()` function shall return a null-terminated character string (with no trailing <newline>)
8516 that describes the last error that occurred during dynamic linking processing. If no dynamic
8517 linking errors have occurred since the last invocation of `dlderror()`, `dlderror()` shall return NULL.
8518 Thus, invoking `dlderror()` a second time, immediately following a prior invocation, shall result in
8519 NULL being returned.

8520 The `dlderror()` function need not be reentrant. A function that is not required to be reentrant is not
8521 required to be thread-safe.

8522 **RETURN VALUE**

8523 If successful, `dlderror()` shall return a null-terminated character string; otherwise, NULL shall be
8524 returned.

8525 **ERRORS**

8526 No errors are defined.

8527 **EXAMPLES**

8528 The following example prints out the last dynamic linking error:

```
8529 ...  
8530 #include <dldfcn.h>  
8531 char *errstr;  
8532 errstr = dlderror();  
8533 if (errstr != NULL)  
8534 printf ("A dynamic linking error occurred: (%s)\n", errstr);  
8535 ...
```

8536 **APPLICATION USAGE**

8537 The messages returned by `dlderror()` may reside in a static buffer that is overwritten on each call
8538 to `dlderror()`. Application code should not write to this buffer. Programs wishing to preserve an
8539 error message should make their own copies of that message. Depending on the application
8540 environment with respect to asynchronous execution events, such as signals or other
8541 asynchronous computation sharing the address space, portable applications should use a critical
8542 section to retrieve the error pointer and buffer.

8543 **RATIONALE**

8544 None.

8545 **FUTURE DIRECTIONS**

8546 None.

8547 **SEE ALSO**8548 `dldclose()`, `dldopen()`, `dldsym()`, the Base Definitions volume of IEEE Std. 1003.1-200x, <dldfcn.h>

8549 **CHANGE HISTORY**

8550 First released in Issue 5.

8551 **Issue 6**

8552 In the DESCRIPTION the note about reentrancy and thread-safety is added.

8553 NAME

8554 dlopen — gain access to an executable object file

8555 SYNOPSIS

8556 XSI `#include <dlfcn.h>`8557 `void *dlopen(const char *file, int mode);`

8558

8559 DESCRIPTION

8560 The *dlopen()* function shall make an executable object file specified by *file* available to the calling
 8561 program. The class of files eligible for this operation and the manner of their construction are
 8562 specified by the implementation, though typically such files are executable objects such as
 8563 shared libraries, relocatable files, or programs. Note that some implementations permit the
 8564 construction of dependencies between such objects that are embedded within files. In such
 8565 cases, a *dlopen()* operation shall load such dependencies in addition to the object referenced by
 8566 *file*. Implementations may also impose specific constraints on the construction of programs that
 8567 can employ *dlopen()* and its related services.

8568 A successful *dlopen()* shall return a *handle* which the caller may use on subsequent calls to
 8569 *dlsym()* and *dlclose()*. The value of this *handle* should not be interpreted in any way by the caller.

8570 *file* is used to construct a path name to the object file. If *file* contains a slash character, the *file*
 8571 argument is used as the path name for the file. Otherwise, *file* is used in an implementation-
 8572 defined manner to yield a path name.

8573 If the value of *file* is 0, *dlopen()* shall provide a *handle* on a global symbol object. This object
 8574 provides access to the symbols from an ordered set of objects consisting of the original program
 8575 image file, together with any objects loaded at program start-up as specified by that process
 8576 image file (for example, shared libraries), and the set of objects loaded using a *dlopen()* operation
 8577 together with the RTLD_GLOBAL flag. As the latter set of objects can change during execution,
 8578 the set identified by *handle* can also change dynamically.

8579 Only a single copy of an object file is brought into the address space, even if *dlopen()* is invoked
 8580 multiple times in reference to the file, and even if different path names are used to reference the
 8581 file.

8582 The *mode* parameter describes how *dlopen()* shall operate upon *file* with respect to the processing
 8583 of relocations and the scope of visibility of the symbols provided within *file*. When an object is
 8584 brought into the address space of a process, it may contain references to symbols whose
 8585 addresses are not known until the object is loaded. These references shall be relocated before the
 8586 symbols can be accessed. The *mode* parameter governs when these relocations take place and
 8587 may have the following values:

8588 RTLD_LAZY Relocations shall be performed at an implementation-defined time,
 8589 ranging from the time of the *dlopen()* call until the first reference to a
 8590 given symbol occurs. Specifying RTLD_LAZY should improve
 8591 performance on implementations supporting dynamic symbol binding as
 8592 a process may not reference all of the functions in any given object. And,
 8593 for systems supporting dynamic symbol resolution for normal process
 8594 execution, this behavior mimics the normal handling of process
 8595 execution.

8596 RTLD_NOW All necessary relocations shall be performed when the object is first
 8597 loaded. This may waste some processing if relocations are performed for
 8598 functions that are never referenced. This behavior may be useful for
 8599 applications that need to know as soon as an object is loaded that all

8600 symbols referenced during execution are available.

8601 Any object loaded by *dlopen()* that requires relocations against global symbols can reference the
8602 symbols in the original process image file, any objects loaded at program start-up, from the
8603 object itself as well as any other object included in the same *dlopen()* invocation, and any objects
8604 that were loaded in any *dlopen()* invocation and which specified the RTLD_GLOBAL flag. To
8605 determine the scope of visibility for the symbols loaded with a *dlopen()* invocation, the *mode*
8606 parameter should be a bitwise-inclusive OR with one of the following values:

8607 RTLD_GLOBAL The object's symbols shall be made available for the relocation processing
8608 of any other object. In addition, symbol lookup using *dlopen(0, mode)* and
8609 an associated *dlsym()* allows objects loaded with this *mode* to be searched.

8610 RTLD_LOCAL The object's symbols shall not be made available for the relocation
8611 processing of any other object.

8612 If neither RTLD_GLOBAL nor RTLD_LOCAL are specified, then an implementation-defined
8613 default behavior shall be applied.

8614 If a *file* is specified in multiple *dlopen()* invocations, *mode* is interpreted at each invocation. Note,
8615 however, that once RTLD_NOW has been specified all relocations shall have been completed
8616 rendering further RTLD_NOW operations redundant and any further RTLD_LAZY operations
8617 irrelevant. Similarly, note that once RTLD_GLOBAL has been specified the object shall maintain
8618 the RTLD_GLOBAL status regardless of any previous or future specification of RTLD_LOCAL,
8619 as long as the object remains in the address space (see *dlclose()*).

8620 Symbols introduced into a program through calls to *dlopen()* may be used in relocation
8621 activities. Symbols so introduced may duplicate symbols already defined by the program or
8622 previous *dlopen()* operations. To resolve the ambiguities such a situation might present, the
8623 resolution of a symbol reference to symbol definition is based on a symbol resolution order. Two
8624 such resolution orders are defined: *load* or *dependency* ordering. Load order establishes an
8625 ordering among symbol definitions, such that the definition first loaded (including definitions
8626 from the image file and any dependent objects loaded with it) has priority over objects added
8627 later (via *dlopen()*). Load ordering is used in relocation processing. Dependency ordering uses a
8628 breadth-first order starting with a given object, then all of its dependencies, then any dependents
8629 of those, iterating until all dependencies are satisfied. With the exception of the global symbol
8630 object obtained via a *dlopen()* operation on a *file* of 0, dependency ordering is used by the
8631 *dlsym()* function. Load ordering is used in *dlsym()* operations upon the global symbol object.

8632 When an object is first made accessible via *dlopen()* it and its dependent objects are added in
8633 dependency order. Once all the objects are added, relocations are performed using load order.
8634 Note that if an object or its dependencies had been previously loaded, the load and dependency
8635 orders may yield different resolutions.

8636 The symbols introduced by *dlopen()* operations, and available through *dlsym()* are at a
8637 minimum those which are exported as symbols of global scope by the object. Typically such
8638 symbols shall be those that were specified in (for example) C source code as having *extern*
8639 linkage. The precise manner in which an implementation constructs the set of exported symbols
8640 for a *dlopen()* object is specified by that implementation.

8641 **RETURN VALUE**

8642 If *file* cannot be found, cannot be opened for reading, is not of an appropriate object format for
8643 processing by *dlopen()*, or if an error occurs during the process of loading *file* or relocating its
8644 symbolic references, *dlopen()* shall return NULL. More detailed diagnostic information shall be
8645 available through *dlerror()*.

8646 **ERRORS**

8647 No errors are defined.

8648 **EXAMPLES**

8649 None.

8650 **APPLICATION USAGE**

8651 None.

8652 **RATIONALE**

8653 None.

8654 **FUTURE DIRECTIONS**

8655 None.

8656 **SEE ALSO**

8657 *dlclose()*, *dLError()*, *dlsym()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**dlfcn.h**>

8658 **CHANGE HISTORY**

8659 First released in Issue 5.

8660 **NAME**8661 dlsym — obtain the address of a symbol from a *dlopen()* object8662 **SYNOPSIS**

8663 XSI #include <dlfcn.h>

8664 void *dlsym(void *restrict *handle*, const char *restrict *name*);

8665

8666 **DESCRIPTION**8667 The *dlsym()* function allows a process to obtain the address of a symbol defined within an object
8668 made accessible through a *dlopen()* call. *handle* is the value returned from a call to *dlopen()* (and
8669 which has not since been released via a call to *dlclose()*), and *name* is the symbol's name as a
8670 character string.8671 The *dlsym()* function shall search for the named symbol in all objects loaded automatically as a
8672 result of loading the object referenced by *handle* (see *dlopen()*). Load ordering is used in *dlsym()*
8673 operations upon the global symbol object. The symbol resolution algorithm used shall be
8674 dependency order as described in *dlopen()*.

8675 The RTLD_NEXT flag is reserved for future use.

8676 **RETURN VALUE**8677 If *handle* does not refer to a valid object opened by *dlopen()*, or if the named symbol cannot be
8678 found within any of the objects associated with *handle*, *dlsym()* shall return NULL. More
8679 detailed diagnostic information shall be available through *dLError()*.8680 **ERRORS**

8681 No errors are defined.

8682 **EXAMPLES**8683 The following example shows how *dlopen()* and *dlsym()* can be used to access either function or
8684 data objects. For simplicity, error checking has been omitted.8685 void *handle;
8686 int *iptr, (*fptr)(int);

8687 /* open the needed object */
8688 handle = dlopen("/usr/home/me/libfoo.so.1", RTLD_LAZY);

8689 /* find the address of function and data objects */
8690 fptr = (int (*)(int))dlsym(handle, "my_function");
8691 iptr = (int *)dlsym(handle, "my_object");

8692 /* invoke function, passing value of integer as a parameter */
8693 (*fptr)(*iptr);8694 **APPLICATION USAGE**8695 Special purpose values for *handle* are reserved for future use. These values and their meanings
8696 are:8697 RTLD_NEXT Specifies the next object after this one that defines *name*. *This one* refers to the
8698 object containing the invocation of *dlsym()*. The *next* object is the one found
8699 upon the application of a load order symbol resolution algorithm (see
8700 *dlopen()*). The next object is either one of global scope (because it was
8701 introduced as part of the original process image or because it was added with
8702 a *dlopen()* operation including the RTLD_GLOBAL flag), or is an object that
8703 was included in the same *dlopen()* operation that loaded this one.

8704 The RTLD_NEXT flag is useful to navigate an intentionally created hierarchy
8705 of multiply-defined symbols created through *interposition*. For example, if a
8706 program wished to create an implementation of *malloc()* that embedded some
8707 statistics gathering about memory allocations, such an implementation could
8708 use the real *malloc()* definition to perform the memory allocation—and itself
8709 only embed the necessary logic to implement the statistics gathering function.

8710 **RATIONALE**

8711 None.

8712 **FUTURE DIRECTIONS**

8713 None.

8714 **SEE ALSO**

8715 *dlclose()*, *derror()*, *dlopen()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <dlfcn.h>

8716 **CHANGE HISTORY**

8717 First released in Issue 5.

8718 **Issue 6**

8719 The **restrict** keyword is added to the *dlsym()* prototype for alignment with the
8720 ISO/IEC 9899:1999 standard.

8721 NAME

8722 drand48, erand48, jrand48, lcong48, lrand48, mrand48, nrand48, seed48, srand48 — generate
8723 uniformly distributed pseudo-random numbers

8724 SYNOPSIS

```
8725 xSI #include <stdlib.h>

8726 double drand48(void);
8727 double erand48(unsigned short xsubi[3]);
8728 long jrand48(unsigned short xsubi[3]);
8729 void lcong48(unsigned short param[7]);
8730 long lrand48(void);
8731 long mrand48(void);
8732 long nrand48(unsigned short xsubi[3]);
8733 unsigned short *seed48(unsigned short seed16v[3]);
8734 void srand48(long seedval);
8735
```

8736 DESCRIPTION

8737 This family of functions generates pseudo-random numbers using a linear congruential
8738 algorithm and 48-bit integer arithmetic.

8739 The *drand48()* and *erand48()* functions shall return non-negative, double-precision, floating-
8740 point values, uniformly distributed over the interval [0.0,1.0).

8741 The *lrnd48()* and *nrnd48()* functions shall return non-negative, long integers, uniformly
8742 distributed over the interval $[0, 2^{31})$.

8743 The *mrnd48()* and *jrnd48()* functions shall return signed long integers uniformly distributed
8744 over the interval $[-2^{31}, 2^{31})$.

8745 The *srand48()*, *seed48()*, and *lcong48()* are initialization entry points, one of which should be
8746 invoked before either *drand48()*, *lrnd48()*, or *mrnd48()* is called. (Although it is not
8747 recommended practice, constant default initializer values shall be supplied automatically if
8748 *drand48()*, *lrnd48()*, or *mrnd48()* is called without a prior call to an initialization entry point.)
8749 The *erand48()*, *nrnd48()*, and *jrnd48()* functions do not require an initialization entry point to
8750 be called first.

8751 All the routines work by generating a sequence of 48-bit integer values, X_i , according to the
8752 linear congruential formula:

$$8753 \quad X_{n+1} = (aX_n + c)_{\text{mod } m} \quad n \geq 0$$

8754 The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless *lcong48()* is invoked,
8755 the multiplier value a and the addend value c are given by:

$$8756 \quad a = 5\text{DEECE66D}_{16} = 273673163155_8$$

$$8757 \quad c = \text{B}_{16} = 13_8$$

8758 The value returned by any of the *drand48()*, *erand48()*, *jrnd48()*, *lrnd48()*, *mrnd48()*, or
8759 *nrnd48()* functions is computed by first generating the next 48-bit X_i in the sequence. Then the
8760 appropriate number of bits, according to the type of data item to be returned, are copied from
8761 the high-order (leftmost) bits of X_i and transformed into the returned value.

8762 The *drand48()*, *lrnd48()*, and *mrnd48()* functions store the last 48-bit X_i generated in an
8763 internal buffer; that is why the application shall ensure that these are initialized prior to being
8764 invoked. The *erand48()*, *nrnd48()*, and *jrnd48()* functions require the calling program to
8765 provide storage for the successive X_i values in the array specified as an argument when the

8766 functions are invoked. That is why these routines do not have to be initialized; the calling
 8767 program merely has to place the desired initial value of X_i into the array and pass it as an
 8768 argument. By using different arguments, *erand48()*, *rand48()*, and *jrand48()* allow separate
 8769 modules of a large program to generate several *independent* streams of pseudo-random numbers;
 8770 that is, the sequence of numbers in each stream shall *not* depend upon how many times the
 8771 routines are called to generate numbers for the other streams.

8772 The initializer function *srand48()* sets the high-order 32 bits of X_i to the low-order 32 bits
 8773 contained in its argument. The low-order 16 bits of X_i are set to the arbitrary value $330E_{16}$.

8774 The initializer function *seed48()* sets the value of X_i to the 48-bit value specified in the argument
 8775 array. The low-order 16 bits of X_i are set to the low-order 16 bits of *seed16v*[0]. The mid-order 16
 8776 bits of X_i are set to the low-order 16 bits of *seed16v*[1]. The high-order 16 bits of X_i are set to the
 8777 low-order 16 bits of *seed16v*[2]. In addition, the previous value of X_i is copied into a 48-bit
 8778 internal buffer, used only by *seed48()*, and a pointer to this buffer is the value returned by
 8779 *seed48()*. This returned pointer, which can just be ignored if not needed, is useful if a program is
 8780 to be restarted from a given point at some future time—use the pointer to get at and store the
 8781 last X_i value, and then use this value to re-initialize via *seed48()* when the program is restarted.

8782 The initializer function *lcg48()* allows the user to specify the initial X_i , the multiplier value a ,
 8783 and the addend value c . Argument array elements *param*[0-2] specify X_i , *param*[3-5] specify the
 8784 multiplier a , and *param*[6] specifies the 16-bit addend c . After *lcg48()* is called, a subsequent
 8785 call to either *srand48()* or *seed48()* shall restore the standard multiplier and addend values, a and
 8786 c , specified above.

8787 The *drand48()*, *lrnd48()*, and *mrnd48()* functions need not be reentrant. A function that is not
 8788 required to be reentrant is not required to be thread-safe.

8789 RETURN VALUE

8790 As described in the DESCRIPTION above.

8791 ERRORS

8792 No errors are defined.

8793 EXAMPLES

8794 None.

8795 APPLICATION USAGE

8796 None.

8797 RATIONALE

8798 None.

8799 FUTURE DIRECTIONS

8800 None.

8801 SEE ALSO

8802 *rand()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>

8803 CHANGE HISTORY

8804 First released in Issue 1. Derived from Issue 1 of the SVID.

8805 Issue 4

8806 The type **long** is replaced by **long** and the type **unsigned short** is replaced by **unsigned short** in
 8807 the SYNOPSIS section.

8808 In the DESCRIPTION, the description of *srand48()* is amended to fix a limitation in Issue 3,
 8809 which indicates that the high-order 32 bits of X_i are set to the {LONG_BIT} bits in the argument.
 8810 Though unintentional, the implication of this statement is that {LONG_BIT} would be 32 on all

- 8811 systems compliant with Issue 3, when in fact Issue 3 imposes no such restriction.
- 8812 The `<stdlib.h>` header is added to the SYNOPSIS section.
- 8813 The argument list for the `lrnd48()` and `mrnd48()` functions now contains **void**.
- 8814 **Issue 5**
- 8815 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.
- 8816 **Issue 6**
- 8817 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

8818 **NAME**

8819 dup, dup2 — duplicate an open file descriptor

8820 **SYNOPSIS**

8821 #include <unistd.h>

8822 int dup(int *fil-des*);8823 int dup2(int *fil-des*, int *fil-des2*);8824 **DESCRIPTION**8825 The *dup()* and *dup2()* functions provide an alternative interface to the service provided by
8826 *fcntl()* using the F_DUPFD command. The call:8827 *fid* = dup(*fil-des*);

8828 is equivalent to:

8829 *fid* = fcntl(*fil-des*, F_DUPFD, 0);

8830 The call:

8831 *fid* = dup2(*fil-des*, *fil-des2*);

8832 is equivalent to:

8833 close(*fil-des2*);8834 *fid* = fcntl(*fil-des*, F_DUPFD, *fil-des2*);

8835 except for the following:

- 8836 • If *fil-des2* is less than 0 or greater than or equal to {OPEN_MAX}, *dup2()* shall return -1 with
8837 *errno* set to [EBADF].
- 8838 • If *fil-des* is a valid file descriptor and is equal to *fil-des2*, *dup2()* shall return *fil-des2* without
8839 closing it.
- 8840 • If *fil-des* is not a valid file descriptor, *dup2()* shall return -1 and shall not close *fil-des2*.
- 8841 • The value returned shall be equal to the value of *fil-des2* upon successful completion, or -1
8842 upon failure.

8843 **RETURN VALUE**8844 Upon successful completion a non-negative integer, namely the file descriptor, shall be returned;
8845 otherwise, -1 shall be returned and *errno* set to indicate the error.8846 **ERRORS**8847 The *dup()* function shall fail if:8848 [EBADF] The *fil-des* argument is not a valid open file descriptor.8849 [EMFILE] The number of file descriptors in use by this process would exceed
8850 {OPEN_MAX}.8851 The *dup2()* function shall fail if:8852 [EBADF] The *fil-des* argument is not a valid open file descriptor or the argument *fil-des2* is
8853 negative or greater than or equal to {OPEN_MAX}.8854 [EINTR] The *dup2()* function was interrupted by a signal.

8855 **EXAMPLES**8856 **Redirecting Standard Output to a File**

8857 The following example closes standard output for the current processes, re-assigns standard
8858 output to go to the file referenced by *pfid*, and closes the original file descriptor to clean up.

```
8859 #include <unistd.h>
8860 ...
8861 int pfid;
8862 ...
8863 close(1);
8864 dup(pfid);
8865 close(pfid);
8866 ...
```

8867 **Redirecting Error Messages**

8868 The following example redirects messages from *stderr* to *stdout*.

```
8869 #include <unistd.h>
8870 ...
8871 dup2(1, 2);
8872 ...
```

8873 **APPLICATION USAGE**

8874 None.

8875 **RATIONALE**

8876 The *dup()* and *dup2()* functions are redundant. Their services are also provided by the *fcntl()*
8877 function. They have been included in this volume of IEEE Std. 1003.1-200x primarily for
8878 historical reasons, since many existing applications use them.

8879 While the brief code segment shown is very similar in behavior to *dup2()*, a conforming
8880 implementation based on other functions defined in this volume of IEEE Std. 1003.1-200x is
8881 significantly more complex. Least obvious is the possible effect of a signal-catching function that
8882 could be invoked between steps and allocate or deallocate file descriptors. This could be avoided
8883 by blocking signals.

8884 The *dup2()* function is not marked obsolescent because it presents a type-safe version of
8885 functionality provided in a type-unsafe version by *fcntl()*. It is used in the POSIX Ada binding.

8886 The *dup2()* function is not intended for use in critical regions as a synchronization mechanism.

8887 In the description of [EBADF], the case of *fildes* being out of range is covered by the given case of
8888 *fildes* not being valid. The descriptions for *fildes* and *fildes2* are different because the only kind of
8889 invalidity that is relevant for *fildes2* is whether it is out of range; that is, it does not matter
8890 whether *fildes2* refers to an open file when the *dup2()* call is made.

8891 **FUTURE DIRECTIONS**

8892 None.

8893 **SEE ALSO**

8894 *close()*, *fcntl()*, *open()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<unistd.h>**

8895 **CHANGE HISTORY**

8896 First released in Issue 1. Derived from Issue 1 of the SVID.

8897 **Issue 4**

8898 The <unistd.h> header is added to the SYNOPSIS section.

8899 [EINTR] is no longer required for *dup()* because *fcntl()* does not return [EINTR] for F_DUPFD.

8900 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

8901 • In the DESCRIPTION, the fourth bullet item describing differences between *dup()* and
8902 *dup2()* is added.

8903 • In the ERRORS section, error values returned by *dup()* and *dup2()* are now described
8904 separately.

8905 **Issue 6**

8906 The RATIONALE section is added.

8907 **NAME**8908 ecvt, fcvt, gcvt — convert a floating-point number to a string (**LEGACY**)8909 **SYNOPSIS**

8910 xSI #include <stdlib.h>

8911 char *ecvt(double value, int ndigit, int *restrict decpt,
8912 int *restrict sign);8913 char *fcvt(double value, int ndigit, int *restrict decpt,
8914 int *restrict sign);

8915 char *gcvt(double value, int ndigit, char *buf);

8916

8917 **DESCRIPTION**8918 The *ecvt()*, *fcvt()*, and *gcvt()* functions shall convert floating-point numbers to null-terminated
8919 strings.8920 The *ecvt()* function shall convert *value* to a null-terminated string of *ndigit* digits (where *ndigit* is
8921 reduced to an unspecified limit determined by the precision of a **double**) and return a pointer to
8922 the string. The high-order digit is non-zero, unless the value is 0. The low-order digit is rounded.
8923 The position of the radix character relative to the beginning of the string is stored in the integer
8924 pointed to by *decpt* (negative means to the left of the returned digits). If *value* is zero, it is
8925 unspecified whether the integer pointed to by *decpt* would be 0 or 1. The radix character is not
8926 included in the returned string. If the sign of the result is negative, the integer pointed to by *sign*
8927 is non-zero; otherwise, it is 0.8928 If the converted value is out of range or is not representable, the contents of the returned string
8929 are unspecified.8930 The *fcvt()* function is identical to *ecvt()* except that *ndigit* specifies the number of digits desired
8931 after the radix character. The total number of digits in the result string is restricted to an
8932 unspecified limit as determined by the precision of a **double**.8933 The *gcvt()* function shall convert *value* to a null-terminated string (similar to that of the *%g*
8934 format of *printf()*) in the array pointed to by *buf* and return *buf*. It produces *ndigit* significant
8935 digits (limited to an unspecified value determined by the precision of a **double**) in *%f* if possible,
8936 or *%e* (scientific notation) otherwise. A minus sign is included in the returned string if *value* is
8937 less than 0. A radix character is included in the returned string if *value* is not a whole number.
8938 Trailing zeros are suppressed where *value* is not a whole number. The radix character is
8939 determined by the current locale. If *setlocale()* has not been called successfully, the default locale,
8940 POSIX, is used. The default locale specifies a period ('.') as the radix character. The
8941 *LC_NUMERIC* category determines the value of the radix character within the current locale.8942 These functions need not be reentrant. A function that is not required to be reentrant is not
8943 required to be thread-safe.8944 **RETURN VALUE**8945 The *ecvt()* and *fcvt()* functions shall return a pointer to a null-terminated string of digits.8946 The *gcvt()* function shall return *buf*.8947 The return values from *ecvt()* and *fcvt()* may point to static data which may be overwritten by
8948 subsequent calls to these functions.8949 **ERRORS**

8950 No errors are defined.

8951 **EXAMPLES**

8952 None.

8953 **APPLICATION USAGE**8954 *sprintf()* is preferred over this function.8955 **RATIONALE**

8956 None.

8957 **FUTURE DIRECTIONS**

8958 These functions may be withdrawn in a future version.

8959 **SEE ALSO**8960 *printf()*, *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>8961 **CHANGE HISTORY**

8962 First released in Issue 4, Version 2.

8963 **Issue 5**

8964 Moved from X/OPEN UNIX extension to BASE.

8965 Normative text previously in the APPLICATION USAGE section is moved to the
8966 DESCRIPTION.

8967 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

8968 **Issue 6**

8969 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8970 This function is marked LEGACY.

8971 The **restrict** keyword is added to the *ecvt()* and *fcvt()* prototypes for alignment with the
8972 ISO/IEC 9899:1999 standard.

8973 **NAME**8974 encrypt — encoding function (**CRYPT**)8975 **SYNOPSIS**8976 XSI `#include <unistd.h>`8977 `void encrypt(char block[64], int edflag);`

8978

8979 **DESCRIPTION**

8980 The `encrypt()` function provides (rather primitive) access to an implementation-defined
8981 encoding algorithm. The key generated by `setkey()` is used to encrypt the string `block` with
8982 `encrypt()`.

8983 The `block` argument to `encrypt()` is an array of length 64 bytes containing only the bytes with
8984 numerical value of 0 and 1. The array is modified in place to a similar array using the key set by
8985 `setkey()`. If `edflag` is 0, the argument is encoded. If `edflag` is 1, the argument may be decoded (see
8986 the APPLICATION USAGE section); if the argument is not decoded, `errno` shall be set to
8987 [ENOSYS].

8988 The `encrypt()` function shall not change the setting of `errno` if successful. An application wishing
8989 to check for error situations should set `errno` to 0 before calling `encrypt()`. If `errno` is non-zero on
8990 return, an error has occurred.

8991 The `encrypt()` function need not be reentrant. A function that is not required to be reentrant is
8992 not required to be thread-safe.

8993 **RETURN VALUE**8994 The `encrypt()` function shall return no value.8995 **ERRORS**8996 The `encrypt()` function shall fail if:

8997 [ENOSYS] The functionality is not supported on this implementation.

8998 **EXAMPLES**

8999 None.

9000 **APPLICATION USAGE**

9001 In some environments, decoding might not be implemented. This is related to U.S. Government
9002 restrictions on encryption and decryption routines: the DES decryption algorithm cannot be
9003 exported outside the U.S. Historical practice has been to ship a different version of the
9004 encryption library without the decryption feature in the routines supplied. Thus the exported
9005 version of `encrypt()` does encoding but not decoding.

9006 **RATIONALE**

9007 None.

9008 **FUTURE DIRECTIONS**

9009 None.

9010 **SEE ALSO**9011 `crypt()`, `setkey()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<unistd.h>`9012 **CHANGE HISTORY**

9013 First released in Issue 1. Derived from Issue 1 of the SVID.

9014 **Issue 4**

9015 The <**unistd.h**> header is added to the SYNOPSIS section.

9016 The DESCRIPTION is amended:

- 9017 • To specify the encoding algorithm as implementation-defined
- 9018 • To change entry to function
- 9019 • To make decoding optional

9020 The APPLICATION USAGE section is expanded to explain the restrictions on the availability of
9021 the DES decryption algorithm.

9022 **Issue 5**

9023 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

9024 **Issue 6**

9025 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

9026 **NAME**

9027 endgrent, getgrent, setgrent — group database entry functions

9028 **SYNOPSIS**

```
9029 xSI      #include <grp.h>
          void endgrent(void);
          struct group *getgrent(void);
          void setgrent(void);
9033
```

9034 **DESCRIPTION**9035 The *endgrent()* function may be called to close the group database when processing is complete.

9036 An implementation that provides extended security controls may impose further
 9037 implementation-defined restrictions on accessing the group database. In particular, the system
 9038 may deny the existence of some or all of the group database entries associated with groups other
 9039 than those groups associated with the caller and may omit users other than the caller from the
 9040 list of members of groups in database entries that are returned.

9041 The *getgrent()* function shall return a pointer to a structure containing the broken-out fields of an
 9042 entry in the group database. When first called, *getgrent()* shall return a pointer to a **group**
 9043 structure containing the first entry in the group database. Thereafter, it shall return a pointer to a
 9044 **group** structure containing the next group structure in the group database, so successive calls
 9045 may be used to search the entire database.

9046 The *setgrent()* function effectively rewinds the group database to allow repeated searches.9047 These functions need not be reentrant. A function that is not required to be reentrant is not
9048 required to be thread-safe.9049 **RETURN VALUE**

9050 When first called, *getgrent()* shall return a pointer to the first group structure in the group
 9051 database. Upon subsequent calls it shall return the next group structure in the group database.
 9052 The *getgrent()* function shall return a null pointer on end-of-file or an error and *errno* may be set
 9053 to indicate the error.

9054 The return value may point to a static area which is overwritten by a subsequent call to
9055 *getgrgid()*, *getgrnam()*, or *getgrent()*.9056 **ERRORS**9057 The *getgrent()* function may fail if:

- | | | |
|------|----------|--|
| 9058 | [EINTR] | A signal was caught during the operation. |
| 9059 | [EIO] | An I/O error has occurred. |
| 9060 | [EMFILE] | {OPEN_MAX} file descriptors are currently open in the calling process. |
| 9061 | [ENFILE] | The maximum allowable number of files is currently open in the system. |

9062 **EXAMPLES**

9063 None.

9064 **APPLICATION USAGE**

9065 These functions are provided due to their historical usage. Applications should avoid
9066 dependencies on fields in the group database, whether the database is a single file, or where in
9067 the file system name space the database resides. Applications should use *getgrnam()* and
9068 *getgrgid()* whenever possible because it avoids these dependencies.

9069 **RATIONALE**

9070 None.

9071 **FUTURE DIRECTIONS**

9072 None.

9073 **SEE ALSO**

9074 *getgrgid()*, *getgrnam()*, *getlogin()*, *getpwent()*, the Base Definitions volume of
9075 IEEE Std. 1003.1-200x, <grp.h>

9076 **CHANGE HISTORY**

9077 First released in Issue 4, Version 2.

9078 **Issue 5**

9079 Moved from X/OPEN UNIX extension to BASE.

9080 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
9081 VALUE section.

9082 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

9083 **Issue 6**

9084 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

9085 **NAME**

9086 endhostent, freehostent, gethostent, sethostent — network host database functions

9087 **SYNOPSIS**

9088 #include <netdb.h>

9089 void endhostent(void);

9090 void freehostent(struct hostent *ptr);

9091 struct hostent *gethostent(void);

9092 void sethostent(int stayopen);

9093 **DESCRIPTION**

9094 These functions enable applications to retrieve information about hosts. This information is
 9095 considered to be stored in a database that can be accessed sequentially or randomly.
 9096 Implementation of this database is unspecified.

9097 **Note:** In many cases it is implemented by the Domain Name System, as documented in
 9098 RFC 1034, RFC 1035, and RFC 1886.

9099 Entries are returned in **hostent** structures. Refer to *gethostbyaddr()* for a definition of the **hostent**
 9100 structure.

9101 The *endhostent()* function shall close the connection to the database, releasing any open file
 9102 descriptor.

9103 The *freehostent()* function shall free the memory occupied by the **hostent** structure pointed to by
 9104 *ptr* and any structures pointed to from that structure, provided that **hostent** was obtained by a
 9105 call to *getipnodebyaddr()* or *getipnodebyname()*. Applications shall not call *freehostent()* except to
 9106 pass it a pointer that was obtained from *getipnodebyaddr()* or *getipnodebyname()*.

9107 The *gethostent()* function shall read the next entry in the database, opening and closing a
 9108 connection to the database as necessary.

9109 The *sethostent()* function shall open a connection to the database and set the next entry for
 9110 retrieval to the first entry in the database. If the *stayopen* argument is non-zero, the connection
 9111 shall not be closed by a call to *gethostent()*, *getipnodebyname()*, *gethostbyname()*, *getipnodebyaddr()*,
 9112 or *gethostbyaddr()*, and the implementation may maintain an open file descriptor.

9113 These functions need not be reentrant. A function that is not required to be reentrant is not
 9114 required to be thread-safe.

9115 **RETURN VALUE**

9116 Upon successful completion, the *gethostent()* function shall return a pointer to a **hostent**
 9117 structure if the requested entry was found, and a null pointer if the end of the database was
 9118 reached or the requested entry was not found.

9119 **ERRORS**9120 No errors are defined for *endhostent()*, *gethostent()*, and *sethostent()*.9121 **EXAMPLES**

9122 None.

9123 **APPLICATION USAGE**

9124 The **hostent** structure returned by *getipnodebyaddr()* and *getipnodebyname()*, and any structures
 9125 pointed to from those structures, are dynamically allocated. Applications should call
 9126 *freehostent()* to free the memory used by these structures.

9127 The *gethostent()* function may return pointers to static data, which may be overwritten by
 9128 subsequent calls to any of these functions. Applications shall not call *freehostent()* for this area.

9129 **RATIONALE**

9130 None.

9131 **FUTURE DIRECTIONS**

9132 None.

9133 **SEE ALSO**9134 *endservent()*, *gethostbyaddr()*, *gethostbyname()*, *getipnodebyaddr()*, *getipnodebyname()*, the Base |9135 Definitions volume of IEEE Std. 1003.1-200x, <**netdb.h**> |9136 **CHANGE HISTORY**

9137 First released in Issue 6. Derived from the XNS, Issue 5.2 specification. |

9138 **NAME**

9139 endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent — network database functions

9140 **SYNOPSIS**

```
9141     #include <netdb.h>

9142     void endnetent(void);
9143     struct netent *getnetbyaddr(uint32_t net, int type);
9144     struct netent *getnetbyname(const char *name);
9145     struct netent *getnetent(void);
9146     void setnetent(int stayopen);
```

9147 **DESCRIPTION**

9148 These functions enable applications to retrieve information about networks. This information is
 9149 considered to be stored in a database that can be accessed sequentially or randomly.
 9150 Implementation of this database is unspecified.

9151 The *endnetent()* function shall close the database, releasing any open file descriptor.

9152 The *getnetbyaddr()* function shall search the database from the beginning, and find the first entry
 9153 for which the address family specified by *type* matches the *n_addrtype* member and the network
 9154 number *net* matches the *n_net* member, opening a connection to the database if necessary. The
 9155 *net* argument shall be the network number in host byte order.

9156 The *getnetbyname()* function shall search the database from the beginning and find the first entry
 9157 for which the network name specified by *name* matches the *n_name* member, opening and
 9158 closing a connection to the database as necessary.

9159 The *getnetent()* function shall read the next entry of the database, opening a connection to the
 9160 database if necessary.

9161 The *setnetent()* function shall open and rewind the database. If the *stayopen* argument is non-
 9162 zero, the connection to the *net* database shall not be closed after each call to *getnetent()* (either
 9163 directly, or indirectly through one of the other *getnet**(*)* functions), and the implementation may
 9164 maintain an open file descriptor to the database.

9165 The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()*, functions shall each return a pointer to a
 9166 **netent** structure, the members of which shall contain the fields of an entry in the network
 9167 database.

9168 These functions need not be reentrant. A function that is not required to be reentrant is not
 9169 required to be thread-safe.

9170 **RETURN VALUE**

9171 Upon successful completion, *getnetbyaddr()*, *getnetbyname()*, and *getnetent()*, shall return a
 9172 pointer to a **netent** structure if the requested entry was found, and a null pointer if the end of the
 9173 database was reached or the requested entry was not found. Otherwise, a null pointer shall be
 9174 returned.

9175 **ERRORS**

9176 No errors are defined.

9177 **EXAMPLES**

9178 None.

9179 **APPLICATION USAGE**9180 The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()*, functions may return pointers to static data,
9181 which may be overwritten by subsequent calls to any of these functions.9182 **RATIONALE**

9183 None.

9184 **FUTURE DIRECTIONS**

9185 None.

9186 **SEE ALSO**9187 The Base Definitions volume of IEEE Std. 1003.1-200x, <**netdb.h**>9188 **CHANGE HISTORY**

9189 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9190 **NAME**

9191 endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent — network protocol
 9192 database functions

9193 **SYNOPSIS**

9194 #include <netdb.h>

```
9195 void endprotoent(void);
9196 struct protoent *getprotobyname(const char *name);
9197 struct protoent *getprotobynumber(int proto);
9198 struct protoent *getprotoent(void);
9199 void setprotoent(int stayopen);
```

9200 **DESCRIPTION**

9201 These functions enable applications to retrieve information about protocols. This information is
 9202 considered to be stored in a database that can be accessed sequentially or randomly.
 9203 Implementation of this database is unspecified.

9204 The *endprotoent()* function shall close the connection to the database, releasing any open file
 9205 descriptor.

9206 The *getprotobyname()* function shall search the database from the beginning and find the first
 9207 entry for which the protocol name specified by *name* matches the *p_name* member, opening a
 9208 connection to the database if necessary.

9209 The *getprotobynumber()* function shall search the database from the beginning and find the first
 9210 entry for which the protocol number specified by *proto* matches the *p_proto* member, opening a
 9211 connection to the database if necessary.

9212 The *getprotoent()* function shall read the next entry of the database, opening and closing a
 9213 connection to the database as necessary.

9214 The *setprotoent()* function shall open a connection to the database, and set the next entry to the
 9215 first entry. If the *stayopen* argument is non-zero, the connection to the network protocol database
 9216 shall not be closed after each call to *getprotoent()* (either directly, or indirectly through one of the
 9217 other *getproto**() functions), and the implementation may maintain an open file descriptor for
 9218 the database.

9219 The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()*, functions shall each return a pointer
 9220 to a **protoent** structure, the members of which shall contain the fields of an entry in the network
 9221 protocol database.

9222 These functions need not be reentrant. A function that is not required to be reentrant is not
 9223 required to be thread-safe.

9224 **RETURN VALUE**

9225 Upon successful completion, *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* return a
 9226 pointer to a **protoent** structure if the requested entry was found, and a null pointer if the end of
 9227 the database was reached or the requested entry was not found. Otherwise, a null pointer is
 9228 returned.

9229 **ERRORS**

9230 No errors are defined.

9231 **EXAMPLES**

9232 None.

9233 **APPLICATION USAGE**

9234 The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions may return pointers to
9235 static data, which may be overwritten by subsequent calls to any of these functions.

9236 **RATIONALE**

9237 None.

9238 **FUTURE DIRECTIONS**

9239 None.

9240 **SEE ALSO**

9241 The Base Definitions volume of IEEE Std. 1003.1-200x, <**netdb.h**>

9242 **CHANGE HISTORY**

9243 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9244 **NAME**

9245 endpwent, getpwent, setpwent — user database functions

9246 **SYNOPSIS**

```
9247 xSI #include <pwd.h>
9248 void endpwent(void);
9249 struct passwd *getpwent(void);
9250 void setpwent(void);
9251
```

9252 **DESCRIPTION**9253 The *endpwent()* function may be called to close the user database when processing is complete.

9254 An implementation that provides extended security controls may impose further
 9255 implementation-defined restrictions on accessing the user database. In particular, the system
 9256 may deny the existence of some or all of the user database entries associated with users other
 9257 than the caller.

9258 The *getpwent()* function shall return a pointer to a structure containing the broken-out fields of
 9259 an entry in the user database. Each entry in the user database contains a **passwd** structure. When
 9260 first called, *getpwent()* shall return a pointer to a **passwd** structure containing the first entry in
 9261 the user database. Thereafter, it shall return a pointer to a **passwd** structure containing the next
 9262 entry in the user database. Successive calls can be used to search the entire user database.

9263 If an end-of-file or an error is encountered on reading, *getpwent()* shall return a null pointer.9264 The *setpwent()* function effectively rewinds the user database to allow repeated searches.

9265 These functions need not be reentrant. A function that is not required to be reentrant is not
 9266 required to be thread-safe.

9267 **RETURN VALUE**9268 The *getpwent()* function shall return a null pointer on end-of-file or error.9269 **ERRORS**9270 The *getpwent()*, *setpwent()*, and *endpwent()* functions may fail if:

9271 [EIO] An I/O error has occurred.

9272 In addition, *getpwent()* and *setpwent()* may fail if:

9273 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

9274 [ENFILE] The maximum allowable number of files is currently open in the system.

9275 The return value may point to a static area which is overwritten by a subsequent call to
 9276 *getpwuid()*, *getpwnam()*, or *getpwent()*.

9277 **EXAMPLES**9278 **Searching the User Database**

9279 The following example uses the *getpwent()* function to get successive entries in the user
 9280 database, returning a pointer to a **passwd** structure that contains information about each user.
 9281 The call to *endpwent()* closes the user database and cleans up.

```
9282 #include <pwd.h>
9283 ...
9284 struct passwd *p;
9285 ...
```

```
9286     while ((p = getpwent ()) != NULL) {
9287         ...
9288     }
9289     endpwent();
9290     ...
```

9291 APPLICATION USAGE

9292 These functions are provided due to their historical usage. Applications should avoid
9293 dependencies on fields in the password database, whether the database is a single file, or where
9294 in the file system name space the database resides. Applications should use *getpwuid()*
9295 whenever possible because it avoids these dependencies.

9296 RATIONALE

9297 None.

9298 FUTURE DIRECTIONS

9299 None.

9300 SEE ALSO

9301 *endgrent()*, *getlogin()*, *getpwnam()*, *getpwuid()*, the Base Definitions volume of
9302 IEEE Std. 1003.1-200x, <pwd.h>

9303 CHANGE HISTORY

9304 First released in Issue 4, Version 2.

9305 Issue 5

9306 Moved from X/OPEN UNIX extension to BASE.

9307 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
9308 VALUE section.

9309 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

9310 Issue 6

9311 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

9312 **NAME**

9313 endservent, getservbyname, getservbyport, getservent, setservent — network services database
 9314 functions

9315 **SYNOPSIS**

```
9316     #include <netdb.h>

9317     void endservent(void);
9318     struct servent *getservbyname(const char *name, const char *proto);
9319     struct servent *getservbyport(int port, const char *proto);
9320     struct servent *getservent(void);
9321     void setservent(int stayopen);
```

9322 **DESCRIPTION**

9323 These functions enable applications to retrieve information about network services. This
 9324 information is considered to be stored in a database that can be accessed sequentially or
 9325 randomly. Implementation of this database is unspecified.

9326 The *endservent()* function shall close the database, releasing any open file descriptor.

9327 The *getservbyname()* function shall search the database from the beginning and find the first
 9328 entry for which the service name specified by *name* matches the *s_name* member and the protocol
 9329 name specified by *proto* matches the *s_proto* member, opening a connection to the database if
 9330 necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be matched.

9331 The *getservbyport()* function shall search the database from the beginning and find the first entry
 9332 for which the port specified by *port* matches the *s_port* member and the protocol name specified
 9333 by *proto* matches the *s_proto* member, opening a connection to the database if necessary. If *proto*
 9334 is a null pointer, any value of the *s_proto* member shall be matched. The *port* argument shall be in
 9335 network byte order.

9336 The *getservent()* function shall read the next entry of the database, opening and closing a
 9337 connection to the database as necessary.

9338 The *setservent()* function shall open a connection to the database, and set the next entry to the
 9339 first entry. If the *stayopen* argument is non-zero, the *net* database shall not be closed after each
 9340 call to the *getservent()* function (either directly, or indirectly through one of the other *getserv*()*
 9341 functions), and the implementation may maintain an open file descriptor for the database.

9342 The *getservbyname()*, *getservbyport()*, and *getservent()* functions shall each return a pointer to a
 9343 **servent** structure, the members of which shall contain the fields of an entry in the network
 9344 services database.

9345 These functions need not be reentrant. A function that is not required to be reentrant is not
 9346 required to be thread-safe.

9347 **RETURN VALUE**

9348 Upon successful completion, *getservbyname()*, *getservbyport()*, and *getservent()* return a pointer to
 9349 a **servent** structure if the requested entry was found, and a null pointer if the end of the database
 9350 was reached or the requested entry was not found. Otherwise, a null pointer is returned.

9351 **ERRORS**

9352 No errors are defined.

9353 **EXAMPLES**

9354 None.

9355 **APPLICATION USAGE**

9356 The *port* argument of *getservbyport()* need not be compatible with the port values of all address
9357 families.

9358 The *getservbyname()*, *getservbyport()*, and *getservent()* functions may return pointers to static
9359 data, which may be overwritten by subsequent calls to any of these functions.

9360 **RATIONALE**

9361 None.

9362 **FUTURE DIRECTIONS**

9363 None.

9364 **SEE ALSO**

9365 *endhostent()*, *endprotoent()*, *htonl()*, *inet_addr()*, the Base Definitions volume of
9366 IEEE Std. 1003.1-200x, <netdb.h>

9367 **CHANGE HISTORY**

9368 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9369 **NAME**

9370 endutxent, getutxent, getutxid, getutxline, pututxline, setutxent — user accounting database
 9371 functions

9372 **SYNOPSIS**

```
9373 xSI #include <utmpx.h>
9374
9374 void endutxent(void);
9375 struct utmpx *getutxent(void);
9376 struct utmpx *getutxid(const struct utmpx *id);
9377 struct utmpx *getutxline(const struct utmpx *line);
9378 struct utmpx *pututxline(const struct utmpx *utmpx);
9379 void setutxent(void);
9380
```

9381 **DESCRIPTION**

9382 These functions provide access to the user accounting database.

9383 The *endutxent()* function closes the user accounting database.

9384 An implementation that provides extended security controls may impose further
 9385 implementation-defined restrictions on accessing the user accounting database. In particular, the
 9386 system may deny the existence of some or all of the user accounting database entries associated
 9387 with users other than the caller.

9388 The *getutxent()* function reads in the next entry from the user accounting database. If the
 9389 database is not already open, it opens it. If it reaches the end of the database, it fails.

9390 The *getutxid()* function searches forward from the current point in the database. If the *ut_type*
 9391 value of the **utmpx** structure pointed to by *id* is *BOOT_TIME*, *OLD_TIME*, or *NEW_TIME*, then
 9392 it stops when it finds an entry with a matching *ut_type* value. If the *ut_type* value is
 9393 *INIT_PROCESS*, *LOGIN_PROCESS*, *USER_PROCESS*, or *DEAD_PROCESS*, then it stops when
 9394 it finds an entry whose type is one of these four and whose *ut_id* member matches the *ut_id*
 9395 member of the **utmpx** structure pointed to by *id*. If the end of the database is reached without a
 9396 match, *getutxid()* fails.

9397 The *getutxid()* or *getutxline()* function may cache data. For this reason, to use *getutxline()* to
 9398 search for multiple occurrences, it is necessary to zero out the static data after each success, or
 9399 *getutxline()* could just return a pointer to the same **utmpx** structure over and over again.

9400 There is one exception to the rule about removing the structure before further reads are done.
 9401 The implicit read done by *pututxline()* (if it finds that it is not already at the correct place in the
 9402 user accounting database) shall not modify the static structure returned by *getutxent()*,
 9403 *getutxid()*, or *getutxline()*, if the application has just modified this structure and passed the
 9404 pointer back to *pututxline()*.

9405 For all entries that match a request, the *ut_type* member indicates the type of the entry. Other
 9406 members of the entry shall contain meaningful data based on the value of the *ut_type* member as
 9407 follows:

9408
9409
9410
9411
9412
9413
9414
9415
9416
9417
9418

ut_type Member	Other Members with Meaningful Data
EMPTY	No others
BOOT_TIME	<i>ut_tv</i>
OLD_TIME	<i>ut_tv</i>
NEW_TIME	<i>ut_tv</i>
USER_PROCESS	<i>ut_id</i> , <i>ut_user</i> (login name of the user), <i>ut_line</i> , <i>ut_pid</i> , <i>ut_tv</i>
INIT_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>
LOGIN_PROCESS	<i>ut_id</i> , <i>ut_user</i> (implementation-defined name of the login process), <i>ut_pid</i> , <i>ut_tv</i>
DEAD_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>

9419
9420
9421
9422

The *getutxline()* function searches forward from the current point in the database until it finds an entry of the type LOGIN_PROCESS or USER_PROCESS which also has a *ut_line* value matching that in the **utmpx** structure pointed to by *line*. If the end of the database is reached without a match, *getutxline()* fails.

9423
9424
9425
9426

If the process has appropriate privileges, the *pututxline()* function writes out the structure into the user accounting database. It uses *getutxid()* to search for a record that satisfies the request. If this search succeeds, then the entry is replaced. Otherwise, a new entry is made at the end of the user accounting database.

9427
9428

The *setutxent()* function resets the input to the beginning of the database. This should be done before each search for a new entry if it is desired that the entire database be examined.

9429
9430

These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

9431 RETURN VALUE

9432
9433
9434

Upon successful completion, *getutxent()*, *getutxid()*, and *getutxline()* shall return a pointer to a **utmpx** structure containing a copy of the requested entry in the user accounting database. Otherwise, a null pointer shall be returned.

9435
9436

The return value may point to a static area which is overwritten by a subsequent call to *getutxid()* or *getutxline()*.

9437
9438
9439

Upon successful completion, *pututxline()* shall return a pointer to a **utmpx** structure containing a copy of the entry added to the user accounting database. Otherwise, a null pointer shall be returned.

9440

The *endutxent()* and *setutxent()* functions shall return no value.

9441 ERRORS

9442
9443

No errors are defined for the *endutxent()*, *getutxent()*, *getutxid()*, *getutxline()*, and *setutxent()* functions.

9444

The *pututxline()* function may fail if:

9445

[EPERM] The process does not have appropriate privileges.

9446 **EXAMPLES**

9447 None.

9448 **APPLICATION USAGE**9449 The sizes of the arrays in the structure can be found using the *sizeof* operator.9450 **RATIONALE**

9451 None.

9452 **FUTURE DIRECTIONS**

9453 None.

9454 **SEE ALSO**

9455 The Base Definitions volume of IEEE Std. 1003.1-200x, <utmpx.h>

9456 **CHANGE HISTORY**

9457 First released in Issue 4, Version 2.

9458 **Issue 5**

9459 Moved from X/OPEN UNIX extension to BASE.

9460 Normative text previously in the APPLICATION USAGE section is moved to the
9461 DESCRIPTION.

9462 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

9463 **Issue 6**

9464 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

9465 **NAME**9466 **environ** — array of character pointers to the environment strings9467 **SYNOPSIS**

9468 extern char **environ;

9469 **DESCRIPTION**9470 Refer to the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables
9471 and *exec*. |

9472 **NAME**

9473 erand48 — generate uniformly distributed pseudo-random numbers

9474 **SYNOPSIS**

9475 xSI #include <stdlib.h>

9476 double erand48(unsigned short xsubi[3]);

9477

9478 **DESCRIPTION**9479 Refer to *drand48()*.

9480 **NAME**

9481 erf, erfc, erfcf, erfcl, erff, erfl — error and complementary error functions

9482 **SYNOPSIS**

```

9483 xSI #include <math.h>
9484     double erf(double x);
9485     double erfc(double x);
9486     float erfcf(float x);
9487     long double erfcl(long double x);
9488     float erff(float x);
9489     long double erfl(long double x);
9490

```

9491 **DESCRIPTION**9492 The *erf()*, *erff()* and *erfl()* functions shall compute the error function of *x*, defined as:

$$\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

9494 The *erfc()*, *erfcf()*, and *erfcl()* functions shall compute $1.0 - erf(x)$.9495 An application wishing to check for error situations should set *errno* to 0 before calling *erf()*. If
9496 *errno* is non-zero on return, or the return value is NaN, an error has occurred.9497 **RETURN VALUE**9498 Upon successful completion, these functions shall return the value of the error function and
9499 complementary error function, respectively.9500 If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].9501 If the correct value would cause underflow, 0 shall be returned and *errno* may be set to
9502 [ERANGE].9503 **ERRORS**9504 The *erfc()*, *erfcf()*, and *erfcl()* functions shall fail if:9505 [ERANGE] The value of *x* is too large.9506 The *erf()* and *erfc()* functions may fail if:9507 [EDOM] The value of *x* is NaN.

9508 [ERANGE] The result underflows

9509 No other errors shall occur.

9510 **EXAMPLES**

9511 None.

9512 **APPLICATION USAGE**9513 The *erfc()* function is provided because of the extreme loss of relative accuracy if *erf(x)* is called
9514 for large *x* and the result subtracted from 1.0.9515 **RATIONALE**

9516 None.

9517 **FUTURE DIRECTIONS**

9518 None.

9519 **SEE ALSO**

9520 *isnan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

9521 **CHANGE HISTORY**

9522 First released in Issue 1. Derived from Issue 1 of the SVID.

9523 **Issue 4**

9524 References to *matherr()* are removed.

9525 The RETURN VALUE and ERRORS sections are substantially rewritten to rationalize error
9526 handling in the mathematics functions.

9527 **Issue 5**

9528 The DESCRIPTION is updated to indicate how an application should check for an error. This
9529 text was previously published in the APPLICATION USAGE section.

9530 **Issue 6**

9531 The *erfcf()*, *erfcl()*, *erff()*, and *erfl()* functions are added for alignment with the
9532 ISO/IEC 9899:1999 standard.

9533 **NAME**9534 `errno` — error return value9535 **SYNOPSIS**9536 `#include <errno.h>`9537 **DESCRIPTION**9538 *errno* is used by many functions to return error values.

9539 Many functions provide an error number in *errno* which has type **int** and is defined in `<errno.h>`.
9540 The value of *errno* shall be defined only after a call to a function for which it is explicitly stated to
9541 be set and until it is changed by the next function call. The value of *errno* should only be
9542 examined when it is indicated to be valid by a function's return value. Programs should obtain
9543 the definition of *errno* by the inclusion of `<errno.h>`. No function in this volume of
9544 IEEE Std. 1003.1-200x shall set *errno* to 0.

9545 It is unspecified whether *errno* is a macro or an identifier declared with external linkage. If a
9546 macro definition is suppressed in order to access an actual object, or a program defines an
9547 identifier with the name *errno*, the behavior is undefined.

9548 The symbolic values stored in *errno* are documented in the ERRORS sections on all relevant
9549 pages.

9550 **RETURN VALUE**

9551 None.

9552 **ERRORS**

9553 None.

9554 **EXAMPLES**

9555 None.

9556 **APPLICATION USAGE**

9557 Previously both POSIX and X/Open documents were more restrictive than the ISO C standard
9558 in that they required *errno* to be defined as an external variable, whereas the ISO C standard
9559 required only that *errno* be defined as a modifiable **lvalue** with type **int**.

9560 A program that uses *errno* for error checking should set it to 0 before a function call, then inspect
9561 it before a subsequent function call.

9562 **RATIONALE**

9563 None.

9564 **FUTURE DIRECTIONS**

9565 None.

9566 **SEE ALSO**9567 Section 2.3, the Base Definitions volume of IEEE Std. 1003.1-200x, `<errno.h>`9568 **CHANGE HISTORY**

9569 First released in Issue 1. Derived from Issue 1 of the SVID.

9570 **Issue 4**

9571 The FUTURE DIRECTIONS section is deleted.

9572 The following changes are incorporated for alignment with the ISO C standard:

- 9573 • The DESCRIPTION now guarantees that *errno* is set to 0 at program start-up, and that it is
9574 never reset to 0 by any XSI function.

- 9575 • The APPLICATION USAGE section is added. This issue is aligned with the ISO C standard,
9576 which permits *errno* to be a macro.

9577 **Issue 5**

9578 The following sentence is deleted from the DESCRIPTION: “The value of *errno* is 0 at program
9579 start-up, but is never set to 0 by any XSI function”. The DESCRIPTION also no longer states that
9580 conforming implementations may support the declaration:

9581 extern int errno;

9582 **Issue 6**

9583 Obsolescent text regarding defining *errno* as:

9584 extern int errno

9585 is removed.

9586 Text regarding no function setting *errno* to zero to indicate an error is changed to no function
9587 shall set *errno* to zero. This is for alignment with the ISO/IEC 9899:1999 standard.

9588 NAME

9589 environ, execl, execv, execl, execve, execlp, execvp — execute a file

9590 SYNOPSIS

```

9591     #include <unistd.h>

9592     extern char **environ;
9593     int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
9594     int execv(const char *path, char *const argv[]);
9595     int execl(const char *path, const char *arg0, ... /*,
9596             (char *)0, char *const envp[] */);
9597     int execve(const char *path, char *const argv[], char *const envp[]);
9598     int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
9599     int execvp(const char *file, char *const argv[]);

```

9600 DESCRIPTION

9601 The *exec* functions shall replace the current process image with a new process image. The new
 9602 image is constructed from a regular, executable file called the *new process image file*. There shall
 9603 be no return from a successful *exec*, because the calling process image is overlaid by the new
 9604 process image.

9605 When a C-language program is executed as a result of this call, it shall be entered as a C-
 9606 language function call as follows:

```
9607     int main (int argc, char *argv[]);
```

9608 where *argc* is the argument count and *argv* is an array of character pointers to the arguments
 9609 themselves. In addition, the following variable:

```
9610     extern char **environ;
```

9611 is initialized as a pointer to an array of character pointers to the environment strings. The *argv*
 9612 and *environ* arrays are each terminated by a null pointer. The null pointer terminating the *argv*
 9613 array is not counted in *argc*.

9614 **THR** Conforming multi-threaded applications shall not use the *environ* variable to access or modify
 9615 any environment variable while any other thread is concurrently modifying any environment
 9616 variable. A call to any function dependent on any environment variable shall be considered a use
 9617 of the *environ* variable to access that environment variable.

9618 The arguments specified by a program with one of the *exec* functions shall be passed on to the
 9619 new process image in the corresponding *main()* arguments.

9620 The argument *path* points to a path name that identifies the new process image file.

9621 The argument *file* is used to construct a path name that identifies the new process image file. If
 9622 the *file* argument contains a slash character, the *file* argument shall be used as the path name for
 9623 this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed
 9624 as the environment variable *PATH* (see the Base Definitions volume of IEEE Std. 1003.1-200x,
 9625 Chapter 8, Environment Variables). If this environment variable is not present, the results of the
 9626 search are implementation-defined.

9627 If the process image file is not a valid executable object, *execlp()* and *execvp()* shall use the
 9628 contents of that file as standard input to a command interpreter conforming to *system()*. In this
 9629 case, the command interpreter becomes the new process image.

9630 The arguments represented by *arg0,...* are pointers to null-terminated character strings. These
 9631 strings constitute the argument list available to the new process image. The list is terminated by
 9632 a null pointer. The argument *arg0* should point to a file name that is associated with the process

9633 being started by one of the *exec* functions.

9634 The argument *argv* is an array of character pointers to null-terminated strings. The application
9635 shall ensure that the last member of this array is a null pointer. These strings constitute the
9636 argument list available to the new process image. The value in *argv*[0] should point to a file
9637 name that is associated with the process being started by one of the *exec* functions.

9638 The argument *envp* is an array of character pointers to null-terminated strings. These strings
9639 constitute the environment for the new process image. The *envp* array is terminated by a null
9640 pointer.

9641 For those forms not containing an *envp* pointer (*execl*(), *execv*(), *execlp*(), and *execvp*()), the
9642 environment for the new process image is taken from the external variable *environ* in the calling
9643 process.

9644 The number of bytes available for the new process' combined argument and environment lists is
9645 {ARG_MAX}. It is implementation-defined whether null terminators, pointers, and/or any
9646 alignment bytes are included in this total.

9647 File descriptors open in the calling process image remain open in the new process image, except
9648 for those whose close-on-exec flag FD_CLOEXEC is set. For those file descriptors that remain
9649 open, all attributes of the open file description remain unchanged. For any file descriptor that is
9650 closed for this reason, file locks are removed as a result of the close as described in *close*(). Locks
9651 that are not removed by closing of file descriptors remain unchanged.

9652 Directory streams open in the calling process image shall be closed in the new process image.

9653 XSI The state of conversion descriptors and message catalog descriptors in the new process image is
9654 undefined. For the new process, the equivalent of:

```
9655 setlocale(LC_ALL, "C")
```

9656 is executed at start-up.

9657 Signals set to the default action (SIG_DFL) in the calling process image shall be set to the default
9658 action in the new process image. Signals set to be ignored (SIG_IGN) by the calling process
9659 image shall be set to be ignored by the new process image. Signals set to be caught by the calling
9660 XSI process image shall be set to the default action in the new process image (see <signal.h>). After
9661 a successful call to any of the *exec* functions, alternate signal stacks are not preserved and the
9662 SA_ONSTACK flag shall be cleared for all signals.

9663 After a successful call to any of the *exec* functions, any functions previously registered by *atexit*()
9664 are no longer registered.

9665 XSI If the ST_NOSUID bit is set for the file system containing the new process image file, then the
9666 effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged
9667 in the new process image. Otherwise, if the set-user-ID mode bit of the new process image file is
9668 set, the effective user ID of the new process image is set to the user ID of the new process image
9669 file. Similarly, if the set-group-ID mode bit of the new process image file is set, the effective
9670 group ID of the new process image is set to the group ID of the new process image file. The real
9671 user ID, real group ID, and supplementary group IDs of the new process image remain the same
9672 as those of the calling process image. The effective user ID and effective group ID of the new
9673 process image shall be saved (as the saved set-user-ID and the saved set-group-ID) for use by
9674 *setuid*().

9675 XSI Any shared memory segments attached to the calling process image shall not be attached to the
9676 new process image.

9677	SEM	Any named semaphores open in the calling process shall be closed as if by appropriate calls to <i>sem_close()</i> .
9678		
9679	TYM	Any blocks of typed memory that were mapped in the calling process are unmapped, as if <i>munmap()</i> was implicitly called to unmap them.
9680		
9681	ML	Memory locks established by the calling process via calls to <i>mlockall()</i> or <i>mlock()</i> shall be removed. If locked pages in the address space of the calling process are also mapped into the address spaces of other processes and are locked by those processes, the locks established by the other processes shall be unaffected by the call by this process to the <i>exec</i> function. If the <i>exec</i> function fails, the effect on memory locks is unspecified.
9682		
9683		
9684		
9685		
9686	MF SHM	Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image.
9687		
9688	PS	For the <i>SCHED_FIFO</i> and <i>SCHED_RR</i> scheduling policies, the policy and priority settings shall not be changed by a call to an <i>exec</i> function. For other scheduling policies, the policy and priority settings on <i>exec</i> are implementation-defined.
9689		
9690		
9691	TMR	Per-process timers created by the calling process shall be deleted before replacing the current process image with the new process image.
9692		
9693	MSG	All open message queue descriptors in the calling process shall be closed, as described in <i>mq_close()</i> .
9694		
9695	AIO	Any outstanding asynchronous I/O operations may be canceled. Those asynchronous I/O operations that are not canceled shall complete as if the <i>exec</i> function had not yet occurred, but any associated signal notifications shall be suppressed. It is unspecified whether the <i>exec</i> function itself blocks awaiting such I/O completion. In no event, however, shall the new process image created by the <i>exec</i> function be affected by the presence of outstanding asynchronous I/O operations at the time the <i>exec</i> function is called. Whether any I/O is canceled, and which I/O may be canceled upon <i>exec</i> , is implementation-defined.
9696		
9697		
9698		
9699		
9700		
9701		
9702	CPT	The new process image shall inherit the CPU-time clock of the calling process image. This inheritance means that the process CPU-time clock of the process being <i>execed</i> shall not be reinitialized or altered as a result of the <i>exec</i> function other than to reflect the time spent by the process executing the <i>exec</i> function itself.
9703		
9704		
9705		
9706	TCT	The initial value of the CPU-time clock of the initial thread of the new process image shall be set to zero.
9707		
9708	TRC	If the calling process is being traced, the new process image continues to be traced into the same trace stream as the original process image, but the new process image shall not inherit the mapping of trace event names to trace event type identifiers that was defined by calls to the <i>posix_trace_eventid_open()</i> or the <i>posix_trace_trid_eventid_open()</i> functions in the calling process image.
9709		
9710		
9711		
9712		
9713		If the calling process is a trace controller process, any trace streams that were created by the calling process shall be shut down as described in the <i>posix_trace_shutdown()</i> function.
9714		
9715		The new process also inherits at least the following attributes from the calling process image:
9716	XSI	<ul style="list-style-type: none"> • Nice value (see <i>nice()</i>)
9717	XSI	<ul style="list-style-type: none"> • <i>semadj</i> values (see <i>semop()</i>)
9718		<ul style="list-style-type: none"> • Process ID
9719		<ul style="list-style-type: none"> • Parent process ID

- 9720 • Process group ID
 - 9721 • Session membership
 - 9722 • Real user ID
 - 9723 • Real group ID
 - 9724 • Supplementary group IDs
 - 9725 • Time left until an alarm clock signal (see *alarm()*)
 - 9726 • Current working directory
 - 9727 • Root directory
 - 9728 • File mode creation mask (see *umask()*)
 - 9729 XSI • File size limit (see *ulimit()*)
 - 9730 • Process signal mask (see *sigprocmask()*)
 - 9731 • Pending signal (see *sigpending()*)
 - 9732 • *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* (see *times()*)
 - 9733 XSI • Resource limits
 - 9734 XSI • Controlling terminal
 - 9735 XSI • Interval timers
- 9736 All other process attributes defined in this volume of IEEE Std. 1003.1-200x shall be the same in
 9737 the new and old process images. The inheritance of process attributes not defined by this
 9738 volume of IEEE Std. 1003.1-200x is implementation-defined.
- 9739 A call to any *exec* function from a process with more than one thread results in all threads being
 9740 terminated and the new executable image being loaded and executed. No destructor functions
 9741 shall be called.
- 9742 Upon successful completion, the *exec* functions shall mark for update the *st_atime* field of the file.
 9743 If an *exec* function failed but was able to locate the *process image file*, whether the *st_atime* field is
 9744 marked for update is unspecified. Should the *exec* function succeed, the process image file shall
 9745 be considered to have been opened with *open()*. The corresponding *close()* shall be considered
 9746 to occur at a time after this open, but before process termination or successful completion of a
 9747 subsequent call to one of the *exec* functions. The *argv[]* and *envp[]* arrays of pointers and the
 9748 strings to which those arrays point shall not be modified by a call to one of the *exec* functions,
 9749 except as a consequence of replacing the process image.
- 9750 XSI The saved resource limits in the new process image are set to be a copy of the process'
 9751 corresponding hard and soft limits.
- 9752 **RETURN VALUE**
- 9753 If one of the *exec* functions returns to the calling process image, an error has occurred; the return
 9754 value shall be -1 , and *errno* shall be set to indicate the error.
- 9755 **ERRORS**
- 9756 The *exec* functions shall fail if:
- 9757 [E2BIG] The number of bytes used by the new process image's argument list and
 9758 environment list is greater than the system-imposed limit of {ARG_MAX}
 9759 bytes.

9760	[EACCES]	Search permission is denied for a directory listed in the new process image file's path prefix, or the new process image file denies execution permission, or the new process image file is not a regular file and the implementation does not support execution of files of its type.
9761		
9762		
9763		
9764	[EINVAL]	The new process image file has the appropriate permission and has a recognized executable binary format, but the system does not support execution of a file with this format.
9765		
9766		
9767	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> or <i>file</i> argument.
9768		
9769	[ENAMETOOLONG]	
9770		The length of the <i>path</i> or <i>file</i> arguments exceeds {PATH_MAX} or a path name component is longer than {NAME_MAX}.
9771		
9772	[ENOENT]	A component of <i>path</i> or <i>file</i> does not name an existing file or <i>path</i> or <i>file</i> is an empty string.
9773		
9774	[ENOTDIR]	A component of the new process image file's path prefix is not a directory.
9775		The <i>exec</i> functions, except for <i>execlp()</i> and <i>execvp()</i> , shall fail if:
9776	[ENOEXEC]	The new process image file has the appropriate access permission but has an unrecognized format.
9777		
9778		The <i>exec</i> functions may fail if:
9779	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> or <i>file</i> argument.
9780		
9781	[ENAMETOOLONG]	
9782		As a result of encountering a symbolic link in resolution of the <i>path</i> argument, the length of the substituted path name string exceeded {PATH_MAX}.
9783		
9784	[ENOMEM]	The new process image requires more memory than is allowed by the hardware or system-imposed memory management constraints.
9785		
9786	[ETXTBSY]	The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.
9787		

9788 EXAMPLES

9789 Using `execl()`

9790 The following example executes the `ls` command, specifying the path name of the executable (`/bin/ls`) and using arguments supplied directly to the command to produce single-column output.

```
9793 #include <unistd.h>
9794 int ret;
9795 ...
9796 ret = execl ("/bin/ls", "ls", "-l", NULL);
```

9797 Using execl()

9798 The following example is similar to **Using execl()** (on page 794). In addition, it specifies the
9799 environment for the new process image using the *env* argument.

```
9800 #include <unistd.h>
9801 int ret;
9802 char *env[] = { "HOME=/usr/home", "LOGNAME=home", NULL };
9803 ...
9804 ret = execl ("/bin/ls", "ls", "-l", NULL, env);
```

9805 Using execlp()

9806 The following example searches for the location of the *ls* command among the directories
9807 specified by the *PATH* environment variable.

```
9808 #include <unistd.h>
9809 int ret;
9810 ...
9811 ret = execlp ("ls", "ls", "-l", NULL);
```

9812 Using execv()

9813 The following example passes arguments to the *ls* command in the *cmd* array.

```
9814 #include <unistd.h>
9815 int ret;
9816 char *cmd[] = { "ls", "-l", NULL };
9817 ...
9818 ret = execv ("/bin/ls", cmd);
```

9819 Using execve()

9820 The following example passes arguments to the *ls* command in the *cmd* array, and specifies the
9821 environment for the new process image using the *env* argument.

```
9822 #include <unistd.h>
9823 int ret;
9824 char *cmd[] = { "ls", "-l", NULL };
9825 char *env[] = { "HOME=/usr/home", "LOGNAME=home", NULL };
9826 ...
9827 ret = execve ("/bin/ls", cmd, env);
```

9828 Using execvp()

9829 The following example searches for the location of the *ls* command among the directories
9830 specified by the *PATH* environment variable, and passes arguments to the *ls* command in the
9831 *cmd* array.

```
9832 #include <unistd.h>
9833 int ret;
9834 char *cmd[] = { "ls", "-l", NULL };
9835 ...
```

9836 ret = execvp ("ls", cmd);

9837 APPLICATION USAGE

9838 As the state of conversion descriptors and message catalog descriptors in the new process image
9839 is undefined, portable applications should not rely on their use and should close them prior to
9840 calling one of the *exec* functions.

9841 Applications that require other than the default POSIX locale should call *setlocale()* with the
9842 appropriate parameters to establish the locale of the new process.

9843 The *environ* array should not be accessed directly by the application.

9844 RATIONALE

9845 Early proposals required that the value of *argc* passed to *main()* be “one or greater”. This was
9846 driven by the same requirement in drafts of the ISO C standard. In fact, historical
9847 implementations have passed a value of zero when no arguments are supplied to the caller of
9848 the *exec* functions. This requirement was removed from the ISO C standard and subsequently
9849 removed from this volume of IEEE Std. 1003.1-200x as well. The wording, in particular the use of
9850 the word *should*, requires a Strictly Conforming POSIX Application to pass at least one argument
9851 to the *exec* function, thus guaranteeing that *argc* be one or greater when invoked by such an
9852 application. In fact, this is good practice, since many existing applications reference *argv[0]*
9853 without first checking the value of *argc*.

9854 The requirement on a Strictly Conforming POSIX Application also states that the value passed
9855 as the first argument be a file name associated with the process being started. Although some
9856 existing applications pass a path name rather than a file name in some circumstances, a file
9857 name is more generally useful, since the common usage of *argv[0]* is in printing diagnostics. In
9858 some cases the file name passed is not the actual file name of the file; for example, many
9859 implementations of the *login* utility use a convention of prefixing a hyphen (‘-’) to the actual
9860 file name, which indicates to the command interpreter being invoked that it is a “login shell”.

9861 Some systems can *exec* shell scripts.

9862 One common historical implementation is that the *execl()*, *execv()*, *execle()*, and *execve()*
9863 functions return an [ENOEXEC] error for any file not recognizable as executable, including a
9864 shell script. When the *execlp()* and *execvp()* functions encounter such a file, they assume the file
9865 to be a shell script and invoke a known command interpreter to interpret such files. These
9866 implementations of *execvp()* and *execlp()* only give the [ENOEXEC] error in the rare case of a
9867 problem with the command interpreter’s executable file. Because of these implementations, the
9868 [ENOEXEC] error is not mentioned for *execlp()* or *execvp()*, although implementations can still
9869 give it.

9870 Another way that some historical implementations handle shell scripts is by recognizing the first
9871 two bytes of the file as the character string “#!” and using the remainder of the first line of the
9872 file as the name of the command interpreter to execute.

9873 Some implementations provide a third argument to *main()* called *envp*. This is defined as a
9874 pointer to the environment. The ISO C standard specifies invoking *main()* with two arguments,
9875 so implementations must support applications written this way. Since this volume of
9876 IEEE Std. 1003.1-200x defines the global variable *environ*, which is also provided by historical
9877 implementations and can be used anywhere that *envp* could be used, there is no functional need
9878 for the *envp* argument. Applications should use the *getenv()* function rather than accessing the
9879 environment directly via either *envp* or *environ*. Implementations are required to support the
9880 two-argument calling sequence, but this does not prohibit an implementation from supporting
9881 *envp* as an optional third argument.

9882 This volume of IEEE Std. 1003.1-200x specifies that signals set to SIG_IGN remain set to
 9883 SIG_IGN, and that the process signal mask be unchanged across an *exec*. This is consistent with
 9884 historical implementations, and it permits some useful functionality, such as the *nohup*
 9885 command. However, it should be noted that many existing applications wrongly assume that
 9886 they start with certain signals set to the default action and/or unblocked. In particular,
 9887 applications written with a simpler signal model that does not include blocking of signals, such
 9888 as the one in the ISO C standard, may not behave properly if invoked with some signals blocked.
 9889 Therefore, it is best not to block or ignore signals across *execs* without explicit reason to do so,
 9890 and especially not to block signals across *execs* of arbitrary (not closely co-operating) programs.

9891 The *exec* functions always save the value of the effective user ID and effective group ID of the
 9892 process at the completion of the *exec*, whether or not the set-user-ID or the set-group-ID bit of
 9893 the process image file is set.

9894 The statement about *argv[]* and *envp[]* being constants is included to make explicit to future
 9895 writers of language bindings that these objects are completely constant. Due to a limitation of
 9896 the ISO C standard, it is not possible to state that idea in standard C. Specifying two levels of
 9897 *const-qualification* for the *argv[]* and *envp[]* parameters for the *exec* functions may seem to be the
 9898 natural choice, given that these functions do not modify either the array of pointers or the
 9899 characters to which the function points, but this would disallow existing correct code. Instead,
 9900 only the array of pointers is noted as constant. The table of assignment compatibility for *dst=src*,
 9901 derived from the ISO C standard summarizes the compatibility:

9902	<i>dst:</i>	char *[]	const char *[]	char *const[]	const char *const[]
9903	<i>src:</i>				
9904	char *[]	VALID	—	VALID	—
9905	const char *[]	—	VALID	—	VALID
9906	char * const []	—	—	VALID	—
9907	const char *const[]	—	—	—	VALID

9908 Since all existing code has a source type matching the first row, the column that gives the most
 9909 valid combinations is the third column. The only other possibility is the fourth column, but
 9910 using it would require a cast on the *argv* or *envp* arguments. It is unfortunate that the fourth
 9911 column cannot be used, because the declaration a non-expert would naturally use would be that
 9912 in the second row.

9913 The ISO C standard and this volume of IEEE Std. 1003.1-200x do not conflict on the use of
 9914 *environ*, but some historical implementations of *environ* may cause a conflict. As long as *environ*
 9915 is treated in the same way as an entry point (for example, *fork()*), it conforms to both standards.
 9916 A library can contain *fork()*, but if there is a user-provided *fork()*, that *fork()* is given precedence
 9917 and no problem ensues. The situation is similar for *environ*: the definition in this volume of
 9918 IEEE Std. 1003.1-200x is to be used if there is no user-provided *environ* to take precedence. At
 9919 least three implementations are known to exist that solve this problem.

9920 [E2BIG] The limit {ARG_MAX} applies not just to the size of the argument list, but to
 9921 the sum of that and the size of the environment list.

9922 [EFAULT] Some historical systems return [EFAULT] rather than [ENOEXEC] when the
 9923 new process image file is corrupted. They are non-conforming.

9924 [EINVAL] This error condition was added to IEEE Std. 1003.1-200x to allow an
 9925 implementation to detect executable files generated for different architectures,
 9926 and indicate this situation to the application. Historical implementations of
 9927 shells, *execvp()*, and *execlp()* that encounter an [ENOEXEC] error will execute
 9928 a shell on the assumption that the file is a shell script. This will not produce
 9929 the desired effect when the file is a valid executable for a different

9930 architecture. An implementation may now choose to avoid this problem by
 9931 returning [EINVAL] when a valid executable for a different architecture is
 9932 encountered. Some historical implementations return [EINVAL] to indicate
 9933 that the *path* argument contains a character with the high order bit set. The
 9934 standard developers chose to deviate from historical practice for the following
 9935 reasons:

- 9936 1. The new utilization of [EINVAL] will provide some measure of utility to
 9937 the user community.
- 9938 2. Historical use of [EINVAL] is not acceptable in an internationalized
 9939 operating environment.

9940 [ENAMETOOLONG]

9941 Since the file path name may be constructed by taking elements in the *PATH*
 9942 variable and putting them together with the file name, the
 9943 [ENAMETOOLONG] error condition could also be reached this way.

9944 [ETXTBSY]

9945 System V returns this error when the executable file is currently open for
 9946 writing by some process. This volume of IEEE Std. 1003.1-200x neither
 requires nor prohibits this behavior.

9947 Other systems (such as System V) may return [EINTR] from *exec*. This is not addressed by this
 9948 volume of IEEE Std. 1003.1-200x, but implementations may have a window between the call to
 9949 *exec* and the time that a signal could cause one of the *exec* calls to return with [EINTR].

9950 FUTURE DIRECTIONS

9951 None.

9952 SEE ALSO

9953 *alarm()*, *atexit()*, *chmod()*, *close()*, *exit()*, *fcntl()*, *fork()*, *fstatvfs()*, *getenv()*, *getitimer()*, *getrlimit()*,
 9954 *mmap()*, *nice()*, <REFERENCE UNDEFINED>(posix_trace_eventid), *posix_trace_shutdown()*,
 9955 *posix_trace_trid_eventid_open()*, *putenv()*, *semop()*, *setlocale()*, *shmat()*, *sigaction()*, *sigaltstack()*,
 9956 *sigpending()*, *sigprocmask()*, *system()*, *times()*, *ulimit()*, *umask()*, the Base Definitions volume of
 9957 IEEE Std. 1003.1-200x, <**unistd.h**>, the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter
 9958 11, General Terminal Interface

9959 CHANGE HISTORY

9960 First released in Issue 1. Derived from Issue 1 of the SVID.

9961 Issue 4

9962 The <**unistd.h**> header is added to the SYNOPSIS section.

9963 The **const** keyword is added to identifiers of constant type (for example, *path*, *file*).

9964 In the DESCRIPTION:

- 9965 • An indication of the disposition of conversion descriptors after a call to one of the *exec*
 9966 functions is added.
- 9967 • A statement about the interaction between *exec* and *atexit()* is added.
- 9968 • *usually* in the descriptions of argument pointers is removed.
- 9969 • *owner ID* is changed to *user ID*.
- 9970 • Shared memory is no longer optional.
- 9971 • The penultimate paragraph is changed to correct an error in Issue 3. It now refers to “All
 9972 other process attributes ...” instead of “All the process attributes ...”

- 9973 A note about the initialization of locales is added to the APPLICATION USAGE section.
- 9974 The following change is incorporated for alignment with the ISO POSIX-1 standard:
- 9975 • In the ERRORS section, the description of the [ENOEXEC] error is changed to indicate that
 - 9976 this error does not apply to *execlp()* and *execvp()*, and the [ENOMEM] error is added.
- 9977 The following change is incorporated for alignment with the FIPS requirements:
- 9978 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path
 - 9979 name component is larger than {NAME_MAX} is now defined as mandatory and marked as
 - 9980 an extension.
- 9981 **Issue 4, Version 2**
- 9982 The following changes are incorporated for X/OPEN UNIX conformance:
- 9983 • The DESCRIPTION is changed:
 - 9984 — To indicate the disposition of alternate signal stacks, the SA_ONSTACK flag, and
 - 9985 mappings established through *mmap()* after a successful call to one of the *exec* functions.
 - 9986 — The effects of ST_NOSUID being set for a file system are defined.
 - 9987 — A statement is added that mappings established through *mmap()* are not preserved across
 - 9988 an *exec*.
 - 9989 — The list of inherited process attributes is extended to include resource limits, the
 - 9990 controlling terminal, and interval timers.
 - 9991 • In the ERRORS section:
 - 9992 — The condition whereby [ELOOP] is returned if too many symbolic links are encountered
 - 9993 during path name resolution is defined as mandatory.
 - 9994 — A second [ENAMETOOLONG] condition is defined that may report excessive length of
 - 9995 an intermediate result of path name resolution of a symbolic link.
- 9996 **Issue 5**
- 9997 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
- 9998 Threads Extension.
- 9999 Large File Summit extensions are added.
- 10000 **Issue 6**
- 10001 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:
- 10002 • The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.
 - 10003 This is since behavior may vary from one file system to another.
- 10004 The following new requirements on POSIX implementations derive from alignment with the
- 10005 Single UNIX Specification:
- 10006 • In the DESCRIPTION, behavior is defined for when the process image file is not a valid
 - 10007 executable.
 - 10008 • In this issue, `_POSIX_SAVED_IDS` is mandated, thus the effective user ID and effective group
 - 10009 ID of the new process image shall be saved (as the saved set-user-ID and the saved set-
 - 10010 group-ID) for use by the *setuid()* function.
 - 10011 • The [ELOOP] mandatory error condition is added.
 - 10012 • A second [ENAMETOOLONG] is added as an optional error condition.

- 10013 • The [ETXTBSY] optional error condition is added.
- 10014 The following changes were made to align with the IEEE P1003.1a draft standard:
- 10015 • The [EINVAL] mandatory error condition is added.
- 10016 • The [ELOOP] optional error condition is added.
- 10017 The description of CPU-time clock semantics is added for alignment with
10018 IEEE Std. 1003.1d-1999.
- 10019 The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by adding semantics
10020 for typed memory.
- 10021 The DESCRIPTION is updated to avoid use of the term “must” for application requirements. |
- 10022 The description of tracing semantics is added for alignment with IEEE Std. 1003.1q-2000. |

10023 NAME

10024 exit, _Exit, _exit — terminate a process

10025 SYNOPSIS

10026 #include <stdlib.h>

10027 void exit(int status);

10028 #include <unistd.h>

10029 void _Exit(int status);

10030 void _exit(int status);

10031 DESCRIPTION

10032 CX The functionality described on this reference page for the *exit()* function is aligned with the
 10033 ISO C standard. Any conflict between the requirements described here and the ISO C standard
 10034 are unintentional. This volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

10035 The *exit()* function shall first call all functions registered by *atexit()*, in the reverse order of their
 10036 registration, except that a function is called after any previously registered functions that had
 10037 already been called at the time it was registered. Each function is called as many times as it was
 10038 registered. If, during the call to any such function, a call to the *longjmp()* function is made that
 10039 would terminate the call to the registered function, the behavior is undefined.

10040 If a function registered by a call to *atexit()* fails to return, the remaining registered functions shall
 10041 not be called and the rest of the *exit()* processing shall not be completed. If *exit()* is called more
 10042 than once, the effects are undefined.

10043 CX The *exit()* function then flushes all output streams, closes all open streams, and removes all files
 10044 created by *tmpfile()*. Finally, control is returned to the host environment as described below. The
 10045 values of *status* can be *EXIT_SUCCESS* or *EXIT_FAILURE*, as described in <stdlib.h>, or any
 10046 CX implementation-defined value, although note that only the range 0 through 255 shall be
 10047 available to a waiting parent process.

10048 The *_Exit()* and *_exit()* functions shall be functionally identical.

10049 CX The *_Exit()*, *_exit()*, and *exit()* functions shall terminate the calling process with the following
 10050 consequences:

- 10051 XSI • All of the file descriptors, directory streams, conversion descriptors, and message catalog
 10052 descriptors open in the calling process are closed.
- 10053 XSI • If the parent process of the calling process is executing a *wait()*, *waitid()*, or *waitpid()*, and has
 10054 neither set its *SA_NOCLDWAIT* flag nor set *SIGCHLD* to *SIG_IGN*, it is notified of the
 10055 calling process' termination and the low-order eight bits (that is, bits 0377) of *status* are made
 10056 available to it. If the parent is not waiting, the child's status shall be made available to it
 10057 XSI when the parent subsequently executes *wait()*, *waitid()*, or *waitpid()*.
- 10058 XSI • If the parent process of the calling process is not executing a *wait()*, *waitid()*, or *waitpid()*, and
 10059 has not set its *SA_NOCLDWAIT* flag, or set *SIGCHLD* to *SIG_IGN*, the calling process is
 10060 transformed into a *zombie process*. A *zombie process* is an inactive process and it shall be
 10061 XSI deleted at some later time when its parent process executes *wait()*, *waitid()*, or *waitpid()*.
- 10062 • Termination of a process does not directly terminate its children. The sending of a *SIGHUP*
 10063 signal as described below indirectly terminates children in some circumstances.
- 10064 • If the implementation supports the *SIGCHLD* signal, a *SIGCHLD* shall be sent to the parent
 10065 process.

- 10066 XSI
10067
10068
10069
- If the parent process has set its SA_NOCLDWAIT flag, or set SIGCHLD to SIG_IGN, the status shall be discarded, and the lifetime of the calling process shall end immediately. If SA_NOCLDWAIT is set, it is implementation-defined whether a SIGCHLD signal shall be sent to the parent process.
- 10070
10071
10072
- The parent process ID of all of the calling process' existing child processes and zombie processes is set to the process ID of an implementation-defined system process. That is, these processes are inherited by a special system process.
- 10073 XSI
10074
- Each attached shared-memory segment is detached and the value of *shm_nattch* (see *shmget()*) in the data structure associated with its shared memory ID is decremented by 1.
- 10075 XSI
10076
- For each semaphore for which the calling process has set a *semadj* value (see *semop()*), that value is added to the *semval* of the specified semaphore.
- 10077
10078
- If the process is a controlling process, the SIGHUP signal shall be sent to each process in the foreground process group of the controlling terminal belonging to the calling process.
- 10079
10080
- If the process is a controlling process, the controlling terminal associated with the session is disassociated from the session, allowing it to be acquired by a new controlling process.
- 10081
10082
10083
- If the exit of the process causes a process group to become orphaned, and if any member of the newly-orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal shall be sent to each process in the newly-orphaned process group.
- 10084 SEM
10085
- All open named semaphores in the calling process shall be closed as if by appropriate calls to *sem_close()*.
- 10086 ML
10087
10088
10089
- Any memory locks established by the process via calls to *mlockall()* or *mlock()* shall be removed. If locked pages in the address space of the calling process are also mapped into the address spaces of other processes and are locked by those processes, the locks established by the other processes shall be unaffected by the call by this process to *_Exit()* or *_exit()*.
- 10090 MF|SHM
- Memory mappings created in the process are unmapped before the process is destroyed.
- 10091 TYM
10092
- Any blocks of typed memory that were mapped in the calling process are unmapped, as if *munmap()* was implicitly called to unmap them.
- 10093 MSG
10094
- All open message queue descriptors in the calling process shall be closed as if by appropriate calls to *mq_close()*.
- 10095 AIO
10096
10097
10098
10099
10100
- Any outstanding cancelable asynchronous I/O operations may be canceled. Those asynchronous I/O operations that are not canceled shall complete as if the *_Exit()* or *_exit()* operation had not yet occurred, but any associated signal notifications shall be suppressed. The *_Exit()* or *_exit()* operation may block awaiting such I/O completion. Whether any I/O is canceled, and which I/O may be canceled upon *_Exit()* or *_exit()*, is implementation-defined.
- 10101
10102
- Threads terminated by a call to *_Exit()* or *_exit()* shall not invoke their cancelation cleanup handlers or per-thread data destructors.
- 10103 TRC
10104
10105
10106
- If the calling process is a trace controller process, any trace streams that were created by the calling process shall be shut down as described by the *posix_trace_shutdown()* function, and any process' mapping of trace event names to trace event type identifiers built for these trace streams may be deallocated.

10107 **RETURN VALUE**

10108 These functions do not return.

10109 **ERRORS**

10110 No errors are defined.

10111 **EXAMPLES**

10112 None.

10113 **APPLICATION USAGE**

10114 Normally applications should use *exit()* rather than *_Exit()* or *_exit()*.

10115 **RATIONALE**10116 **Process Termination**

10117 Early proposals drew a different distinction between normal and abnormal process termination.
10118 Abnormal termination was caused only by certain signals and resulted in implementation-
10119 defined “actions”, as discussed below. Subsequent proposals distinguished three types of
10120 termination: *normal termination* (as in the current specification), *simple abnormal termination*, and
10121 *abnormal termination with actions*. Again the distinction between the two types of abnormal
10122 termination was that they were caused by different signals and that implementation-defined
10123 actions would result in the latter case. Given that these actions were completely
10124 implementation-defined, the early proposals were only saying when the actions could occur and
10125 how their occurrence could be detected, but not what they were. This was of little or no use to
10126 portable applications, and thus the distinction is not made in this volume of
10127 IEEE Std. 1003.1-200x.

10128 The implementation-defined actions usually include, in most historical implementations, the
10129 creation of a file named **core** in the current working directory of the process. This file contains an
10130 image of the memory of the process, together with descriptive information about the process,
10131 perhaps sufficient to reconstruct the state of the process at the receipt of the signal.

10132 There is a potential security problem in creating a **core** file if the process was set-user-ID and the
10133 current user is not the owner of the program, if the process was set-group-ID and none of the
10134 user’s groups match the group of the program, or if the user does not have permission to write in
10135 the current directory. In this situation, an implementation either should not create a **core** file or
10136 should make it unreadable by the user.

10137 Despite the silence of this volume of IEEE Std. 1003.1-200x on this feature, applications are
10138 advised not to create files named **core** because of potential conflicts in many implementations.
10139 Some historical implementations use a different name than **core** for the file, such as by
10140 appending the process ID to the file name.

10141 **Terminating a Process**

10142 It is important that the consequences of process termination as described occur regardless of
10143 whether the process called *_exit()* (perhaps indirectly through *exit()*) or instead was terminated
10144 due to a signal or for some other reason. Note that in the specific case of *exit()* this means that
10145 the *status* argument to *exit()* is treated the same as the *status* argument to *_exit()*.

10146 A language other than C may have other termination primitives than the C-language *exit()*
10147 function, and programs written in such a language should use its native termination primitives,
10148 but those should have as part of their function the behavior of *_exit()* as described.
10149 Implementations in languages other than C are outside the scope of the present version of this
10150 volume of IEEE Std. 1003.1-200x, however.

10151 As required by the ISO C standard, using **return** from *main()* is equivalent to calling *exit()* with
10152 the same argument value. Also, reaching the end of the *main()* function is equivalent to using
10153 *exit()* with an unspecified value.

10154 A value of zero (or `EXIT_SUCCESS`, which is required to be zero) for the argument *status*
10155 conventionally indicates successful termination. This corresponds to the specification for *exit()*
10156 in the ISO C standard. The convention is followed by utilities such as *make* and various shells,
10157 which interpret a zero status from a child process as success. For this reason, applications should
10158 not call *exit(0)* or *_exit(0)* when they terminate unsuccessfully; for example, in signal-catching
10159 functions.

10160 Historically, the implementation-defined process that inherits children whose parents have
10161 terminated without waiting on them is called *init* and has a process ID of 1.

10162 The sending of a `SIGHUP` to the foreground process group when a controlling process
10163 terminates corresponds to somewhat different historical implementations. In System V, the
10164 kernel sends a `SIGHUP` on termination of (essentially) a controlling process. In 4.2 BSD, the
10165 kernel does not send `SIGHUP` in a case like this, but the termination of a controlling process is
10166 usually noticed by a system daemon, which arranges to send a `SIGHUP` to the foreground
10167 process group with the *vhangup()* function. However, in 4.2 BSD, due to the behavior of the
10168 shells that support job control, the controlling process is usually a shell with no other processes
10169 in its process group. Thus, a change to make *_exit()* behave this way in such systems should not
10170 cause problems with existing applications.

10171 The termination of a process may cause a process group to become orphaned in either of two
10172 ways. The connection of a process group to its parent(s) outside of the group depends on both
10173 the parents and their children. Thus, a process group may be orphaned by the termination of the
10174 last connecting parent process outside of the group or by the termination of the last direct
10175 descendant of the parent process(es). In either case, if the termination of a process causes a
10176 process group to become orphaned, processes within the group are disconnected from their job
10177 control shell, which no longer has any information on the existence of the process group.
10178 Stopped processes within the group would languish forever. In order to avoid this problem,
10179 newly orphaned process groups that contain stopped processes are sent a `SIGHUP` signal and a
10180 `SIGCONT` signal to indicate that they have been disconnected from their session. The `SIGHUP`
10181 signal causes the process group members to terminate unless they are catching or ignoring
10182 `SIGHUP`. Under most circumstances, all of the members of the process group are stopped if any
10183 of them are stopped.

10184 The action of sending a `SIGHUP` and a `SIGCONT` signal to members of a newly orphaned
10185 process group is similar to the action of 4.2 BSD, which sends `SIGHUP` and `SIGCONT` to each
10186 stopped child of an exiting process. If such children exit in response to the `SIGHUP`, any
10187 additional descendants receive similar treatment at that time. In this volume of
10188 IEEE Std. 1003.1-200x, the signals are sent to the entire process group at the same time. Also, in
10189 this volume of IEEE Std. 1003.1-200x, but not in 4.2 BSD, stopped processes may be orphaned,
10190 but may be members of a process group that is not orphaned; therefore, the action taken at
10191 *_exit()* must consider processes other than child processes.

10192 It is possible for a process group to be orphaned by a call to *setpgid()* or *setsid()*, as well as by
10193 process termination. This volume of IEEE Std. 1003.1-200x does not require sending `SIGHUP`
10194 and `SIGCONT` in those cases, because, unlike process termination, those cases are not caused
10195 accidentally by applications that are unaware of job control. An implementation can choose to
10196 send `SIGHUP` and `SIGCONT` in those cases as an extension; such an extension must be
10197 documented as required in `<signal.h>`.

10198 The ISO/IEC 9899:1999 standard adds the *_Exit()* function that results in immediate program
10199 termination without triggering signals or *atexit()*-registered functions. In IEEE Std. 1003.1-200x,

10200 this is equivalent to the `_exit()` function.

10201 FUTURE DIRECTIONS

10202 None.

10203 SEE ALSO

10204 `atexit()`, `close()`, `fclose()`, `longjmp()`, <REFERENCE UNDEFINED>(posix_trace_eventid),
 10205 `posix_trace_shutdown()`, `posix_trace_trid_eventid_open()`, `semop()`, `shmget()`, `sigaction()`, `wait()`,
 10206 `waitid()`, `waitpid()`, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>, <unistd.h>

10207 CHANGE HISTORY

10208 First released in Issue 1. Derived from Issue 1 of the SVID.

10209 Issue 4

10210 The <unistd.h> header is added to the SYNOPSIS for `_exit()`.

10211 In the DESCRIPTION, text is added describing the behavior when a function registered by
 10212 `atexit()` fails to return, and the consequences of calling `exit()` more than once.

10213 The phrase “If the implementation supports job control” is removed from the last bullet in the
 10214 DESCRIPTION. This is because job control is now defined as mandatory for all conforming
 10215 implementations.

10216 The following change is incorporated for alignment with the ISO C standard:

- 10217 • In the DESCRIPTION, interactions between `exit()` and `atexit()` are defined, and it is now
- 10218 stated explicitly that all files created by `tmpfile()` are removed.

10219 Issue 4, Version 2

10220 The following changes to the DESCRIPTION are incorporated for X/OPEN UNIX conformance:

- 10221 • References to the functions `wait3()` and `waitid()` are added in appropriate places throughout
- 10222 the text.
- 10223 • Interactions with the SA_NOCLDWAIT flag and SIGCHLD signal are defined.
- 10224 • It is specified that each mapped memory object is unmapped.

10225 Issue 5

10226 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 10227 Threads Extension.

10228 Interactions with the SA_NOCLDWAIT flag and SIGCHLD signal are further clarified.

10229 The values of `status` from `exit()` are better described.

10230 Issue 6

10231 Extensions beyond the ISO C standard are now marked.

10232 The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by adding semantics
 10233 for typed memory.

10234 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 10235 • The `_Exit()` function is included.
- 10236 • The DESCRIPTION is updated.

10237 The description of tracing semantics is added for alignment with IEEE Std. 1003.1q-2000.

10238 References to the `wait3()` function are removed.

10239 **NAME**

10240 exp, expf, expl — exponential function

10241 **SYNOPSIS**

10242 #include <math.h>

10243 double exp(double x);

10244 float expf(float x);

10245 long double expl(long double x);

10246 **DESCRIPTION**

10247 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 10248 conflict between the requirements described here and the ISO C standard is unintentional. This
 10249 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

10250 These functions shall compute the exponent of x , defined as e^x .

10251 An application wishing to check for error situations should set *errno* to 0 before calling *exp()*. If
 10252 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

10253 **RETURN VALUE**10254 Upon successful completion, these functions shall return the exponential value of x .

10255 If the correct value would cause overflow, *exp()* shall return HUGE_VAL and set *errno* to
 10256 [ERANGE].

10257 If the correct value would cause underflow, *exp()* shall return 0 and may set *errno* to [ERANGE].10258 **XSI** If x is NaN, NaN shall be returned and *errno* may be set to [EDOM].10259 **ERRORS**

10260 These functions shall fail if:

10261 [ERANGE] The result overflows.

10262 These functions may fail if:

10263 **XSI** [EDOM] The value of x is NaN.

10264 [ERANGE] The result underflows

10265 **XSI** No other errors shall occur.10266 **EXAMPLES**

10267 None.

10268 **APPLICATION USAGE**

10269 None.

10270 **RATIONALE**

10271 None.

10272 **FUTURE DIRECTIONS**

10273 None.

10274 **SEE ALSO**10275 *isnan()*, *log()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>10276 **CHANGE HISTORY**

10277 First released in Issue 1. Derived from Issue 1 of the SVID.

10278 **Issue 4**

10279 References to *matherr()* are removed.

10280 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the
10281 ISO C standard and to rationalize error handling in the mathematics functions.

10282 The return value specified for [EDOM] is marked as an extension. |

10283 **Issue 5**

10284 The DESCRIPTION is updated to indicate how an application should check for an error. This
10285 text was previously published in the APPLICATION USAGE section. |

10286 **Issue 6**

10287 The *expf()* and *expl()* functions are added for alignment with the ISO/IEC 9899:1999 standard. |

10288 **NAME**

10289 exp2, exp2f, exp2l — exponential base 2 functions

10290 **SYNOPSIS**

10291 #include <math.h>

10292 double exp2(double x);

10293 float exp2f(float x);

10294 long double exp2l(long double x);

10295 **DESCRIPTION**

10296 **cx** The functionality described on this reference page is aligned with the ISO C standard. Any
10297 conflict between the requirements described here and the ISO C standard is unintentional. This
10298 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

10299 These functions shall compute the base 2 exponent of x , defined as e^x .

10300 An application wishing to check for error situations should set *errno* to 0 before calling these
10301 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

10302 **RETURN VALUE**10303 Upon successful completion, these functions shall return 2^x .

10304 If the correct value would cause overflow, these functions shall return HUGE_VAL and set *errno*
10305 to [ERANGE].

10306 If the correct value would cause underflow, these functions shall return 0 and may set *errno* to
10307 [ERANGE].

10308 If x is NaN, NaN shall be returned and *errno* may be set to [EDOM].10309 **ERRORS**

10310 These functions shall fail if:

10311 [ERANGE] The result overflows.

10312 These functions may fail if:

10313 [EDOM] The value of x is NaN.

10314 [ERANGE] The result underflows

10315 No other errors shall occur.

10316 **EXAMPLES**

10317 None.

10318 **APPLICATION USAGE**

10319 None.

10320 **RATIONALE**

10321 None.

10322 **FUTURE DIRECTIONS**

10323 None.

10324 **SEE ALSO**10325 *exp()*, *isnan()*, *log()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

10326 **CHANGE HISTORY**

10327 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

10328 **NAME**

10329 expm1, expm1f, expm1l — compute exponential functions

10330 **SYNOPSIS**

10331 #include <math.h>

10332 double expm1(double x);

10333 float expm1f(float x);

10334 long double expm1l(long double x);

10335 **DESCRIPTION**10336 These functions shall compute $e^x - 1.0$.10337 **RETURN VALUE**10338 If x is NaN, then these functions shall return NaN and *errno* may be set to [EDOM].10339 If x is positive infinity, these functions shall return positive infinity.10340 If x is negative infinity, these functions shall return -1.0 .10341 If the value overflows, these functions shall return HUGE_VAL and may set *errno* to [ERANGE].10342 **ERRORS**

10343 These functions may fail if:

10344 [EDOM] The value of x is NaN.

10345 [ERANGE] The result overflows.

10346 **EXAMPLES**

10347 None.

10348 **APPLICATION USAGE**10349 The value of *expm1*(x) may be more accurate than *exp*(x) -1.0 for small values of x .10350 The *expm1*() and *log1p*() functions are useful for financial calculations of $((1+x)^n - 1)/x$, namely:10351
$$\text{expm1}(n * \text{log1p}(x)) / x$$
10352 when x is very small (for example, when calculating small daily interest rates). These functions
10353 also simplify writing accurate inverse hyperbolic functions.10354 **RATIONALE**

10355 None.

10356 **FUTURE DIRECTIONS**

10357 None.

10358 **SEE ALSO**10359 *exp*(), *ilogb*(), *log1p*(), the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>10360 **CHANGE HISTORY**

10361 First released in Issue 4, Version 2.

10362 **Issue 5**

10363 Moved from X/OPEN UNIX extension to BASE.

10364 **Issue 6**10365 The *expm1f*() and *expm1l*() functions are added for alignment with the ISO/IEC 9899:1999
10366 standard.

10367 **NAME**

10368 fabs, fabsf, fabsl — absolute value function

10369 **SYNOPSIS**

10370 #include <math.h>

10371 double fabs(double x);

10372 float fabsf(float x);

10373 long double fabsl(long double x);

10374 **DESCRIPTION**

10375 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 10376 conflict between the requirements described here and the ISO C standard is unintentional. This
 10377 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

10378 These functions shall compute the absolute value of x , $|x|$.

10379 An application wishing to check for error situations should set *errno* to 0 before calling *fabs()*. If
 10380 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

10381 **RETURN VALUE**10382 Upon successful completion, these functions shall return the absolute value of x .10383 **XSI** If x is NaN, NaN shall be returned and *errno* may be set to [EDOM].10384 If the result underflows, 0 shall be returned and *errno* may be set to [ERANGE].10385 **ERRORS**

10386 These functions may fail if:

10387 **XSI** [EDOM] The value of x is NaN.

10388 [ERANGE] The result underflows

10389 **XSI** No other errors shall occur.10390 **EXAMPLES**

10391 None.

10392 **APPLICATION USAGE**

10393 None.

10394 **RATIONALE**

10395 None.

10396 **FUTURE DIRECTIONS**

10397 None.

10398 **SEE ALSO**10399 *isnan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>10400 **CHANGE HISTORY**

10401 First released in Issue 1. Derived from Issue 1 of the SVID.

10402 **Issue 4**10403 References to *matherr()* are removed.

10404 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the
 10405 ISO C standard and to rationalize error handling in the mathematics functions.

10406 The return value specified for [EDOM] is marked as an extension.

10407 **Issue 5**

10408 The DESCRIPTION is updated to indicate how an application should check for an error. This
10409 text was previously published in the APPLICATION USAGE section.

10410 **Issue 6**

10411 The *fabsf()* and *fabsl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

10412 **NAME**

10413 fattach — attach a STREAMS-based file descriptor to a file in the file system name space
 10414 (**STREAMS**)

10415 **SYNOPSIS**

```
10416 XSR #include <stropts.h>
```

```
10417 int fattach(int fildev, const char *path);
```

10418

10419 **DESCRIPTION**10420 **Notes to Reviewers**

10421 *This section with side shading will not appear in the final copy. - Ed.*

10422 Re D1, XSH, ERN 111: if the original file had multiple links, the streams file still has only one? I
 10423 presume that a stream is actually attached to an inode, not a file name. If so, there continue to
 10424 exist multiple links to the object, even though it shows a link count of 1. If it associated the
 10425 stream with a file name, then the following sentence is wrong.

10426 The *fattach()* function attaches a STREAMS-based file descriptor to a file, effectively associating
 10427 a path name with *fildev*. The application shall ensure that the *fildev* argument is a valid open file
 10428 descriptor associated with a STREAMS file. The *path* argument points to a path name of an
 10429 existing file. The application shall ensure that the process has appropriate privileges, or is the
 10430 owner of the file named by *path* and has write permission. A successful call to *fattach()* shall
 10431 cause all path names that name the file named by *path* to name the STREAMS file associated
 10432 with *fildev*, until the STREAMS file is detached from the file. A STREAMS file can be attached to
 10433 more than one file and can have several path names associated with it.

10434 The attributes of the named STREAMS file shall be initialized as follows: the permissions, user
 10435 ID, group ID, and times are set to those of the file named by *path*, the number of links is set to 1,
 10436 and the size and device identifier are set to those of the STREAMS file associated with *fildev*. If
 10437 any attributes of the named STREAMS file are subsequently changed (for example, by *chmod()*),
 10438 neither the attributes of the underlying file nor the attributes of the STREAMS file to which *fildev*
 10439 refers shall be affected.

10440 File descriptors referring to the underlying file, opened prior to an *fattach()* call, shall continue to
 10441 refer to the underlying file.

10442 **RETURN VALUE**

10443 Upon successful completion, *fattach()* shall return 0. Otherwise, -1 shall be returned and *errno*
 10444 set to indicate the error.

10445 **ERRORS**

10446 The *fattach()* function shall fail if:

10447 [EACCES] Search permission is denied for a component of the path prefix, or the process
 10448 is the owner of *path* but does not have write permissions on the file named by
 10449 *path*.

10450 [EBADF] The *fildev* argument is not a valid open file descriptor.

10451 [EBUSY] The file named by *path* is currently a mount point or has a STREAMS file
 10452 attached to it.

10453 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 10454 argument.

10455	[ENAMETOOLONG]	
10456		The size of <i>path</i> exceeds {PATH_MAX} or a component of <i>path</i> is longer than
10457		{NAME_MAX}.
10458	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
10459	[ENOTDIR]	A component of the path prefix is not a directory.
10460	[EPERM]	The effective user ID of the process is not the owner of the file named by <i>path</i>
10461		and the process does not have appropriate privilege.
10462		The <i>fattach()</i> function may fail if:
10463	[EINVAL]	The <i>fildev</i> argument does not refer to a STREAMS file.
10464	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
10465		resolution of the <i>path</i> argument.
10466	[ENAMETOOLONG]	
10467		Path name resolution of a symbolic link produced an intermediate result
10468		whose length exceeds {PATH_MAX}.
10469	[EXDEV]	A link to a file on another file system was attempted.

10470 EXAMPLES

10471 Attaching a File Descriptor to a File

10472 In the following example, *fd* refers to an open STREAMS file. The call to *fattach()* associates this
 10473 STREAM with the file **/tmp/named-STREAM**, such that any future calls to open **/tmp/named-**
 10474 **STREAM**, prior to breaking the attachment via a call to *fdetach()*, will instead create a new file
 10475 handle referring to the STREAMS file associated with *fd*.

```
10476 #include <stropts.h>
10477 ...
10478     int fd;
10479     char *filename = "/tmp/named-STREAM";
10480     int ret;
10481     ret = fattach(fd, filename);
```

10482 APPLICATION USAGE

10483 The *fattach()* function behaves similarly to the traditional *mount()* function in the way a file is
 10484 temporarily replaced by the root directory of the mounted file system. In the case of *fattach()*, the
 10485 replaced file need not be a directory and the replacing file is a STREAMS file.

10486 RATIONALE

10487 None.

10488 FUTURE DIRECTIONS

10489 None.

10490 SEE ALSO

10491 *fdetach()*, *isastream()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**stropts.h**>

10492 CHANGE HISTORY

10493 First released in Issue 4, Version 2.

10494 **Issue 5**

10495 Moved from X/OPEN UNIX extension to BASE.

10496 The [EXDEV] error is added to the list of optional errors in the ERRORS section.

10497 **Issue 6**

10498 This function is marked as part of the XSI STREAMS Option Group.

10499 The DESCRIPTION is updated to avoid use of the term “must” for application requirements. |

10500 The wording of the mandatory [ELOOP] error condition is updated, and a second optional |
10501 [ELOOP] error condition is added. |

10502 **NAME**

10503 fchdir — change working directory

10504 **SYNOPSIS**

10505 xSI #include <unistd.h>

10506 int fchdir(int *fildev*);

10507

10508 **DESCRIPTION**10509 The *fchdir()* function has the same effect as *chdir()* except that the directory that is to be the new
10510 current working directory is specified by the file descriptor *fildev*.10511 **RETURN VALUE**10512 Upon successful completion, *fchdir()* shall return 0. Otherwise, it shall return -1 and set *errno* to
10513 indicate the error. On failure the current working directory shall remain unchanged.10514 **ERRORS**10515 The *fchdir()* function shall fail if:10516 [EACCES] Search permission is denied for the directory referenced by *fildev*.10517 [EBADF] The *fildev* argument is not an open file descriptor.10518 [ENOTDIR] The open file descriptor *fildev* does not refer to a directory.10519 The *fchdir()* may fail if:10520 [EINTR] A signal was caught during the execution of *fchdir()*.

10521 [EIO] An I/O error occurred while reading from or writing to the file system.

10522 **EXAMPLES**

10523 None.

10524 **APPLICATION USAGE**

10525 None.

10526 **RATIONALE**

10527 None.

10528 **FUTURE DIRECTIONS**

10529 None.

10530 **SEE ALSO**10531 *chdir()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <unistd.h>10532 **CHANGE HISTORY**

10533 First released in Issue 4, Version 2.

10534 **Issue 5**

10535 Moved from X/OPEN UNIX extension to BASE.

10536 **NAME**

10537 fchmod — change mode of a file

10538 **SYNOPSIS**

10539 #include <sys/stat.h>

10540 int fchmod(int *fildev*, mode_t *mode*);10541 **DESCRIPTION**10542 The *fchmod()* function has the same effect as *chmod()* except that the file whose permissions are
10543 changed is specified by the file descriptor *fildev*.10544 SHM If *fildev* references a shared memory object, the *fchmod()* function need only affect the S_IRUSR,
10545 S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.10546 TYM If *fildev* references a typed memory object, the behavior of *fchmod()* is unspecified.10547 **Notes to Reviewers**10548 *This section with side shading will not appear in the final copy. - Ed.*10549 D3, XSH, ERN 178 suggests adding text as follows: "If *fildev* refers to a STREAM (which is
10550 fattached() into the file system name space) the call returns successfully, doing nothing. If *fildev*
10551 refers to a stream, <do what?>."10552 **RETURN VALUE**10553 Upon successful completion, *fchmod()* shall return 0. Otherwise, it shall return -1 and set *errno* to
10554 indicate the error.10555 **ERRORS**10556 The *fchmod()* function shall fail if:10557 [EBADF] The *fildev* argument is not an open file descriptor.10558 [EPERM] The effective user ID does not match the owner of the file and the process
10559 does not have appropriate privilege.10560 [EROFS] The file referred to by *fildev* resides on a read-only file system.10561 The *fchmod()* function may fail if:10562 XSI [EINTR] The *fchmod()* function was interrupted by a signal.10563 XSI [EINVAL] The value of the *mode* argument is invalid.10564 [EINVAL] The *fildev* argument refers to a pipe and the implementation disallows
10565 execution of *fchmod()* on a pipe.10566 **EXAMPLES**10567 **Changing the Current Permissions for a File**10568 The following example shows how to change the permissions for a file named */home/cnd/mod1*
10569 so that the owner and group have read/write/execute permissions, but the world only has
10570 read/write permissions.

10571 #include <sys/stat.h>

10572 #include <fcntl.h>

10573 mode_t mode;

10574 int fildev;

10575 ...

```
10576     fildes = open("/home/cnd/mod1", O_RDWR);
10577     fchmod(fildes, S_IRWXU | S_IRWXG | S_IROTH | S_IWOTH);
```

10578 APPLICATION USAGE

10579 None.

10580 RATIONALE

10581 None.

10582 FUTURE DIRECTIONS

10583 None.

10584 SEE ALSO

10585 *chmod()*, *chown()*, *creat()*, *fcntl()*, *fstatvfs()*, *mknod()*, *open()*, *read()*, *stat()*, *write()*, the Base
10586 Definitions volume of IEEE Std. 1003.1-200x, <sys/stat.h>

10587 CHANGE HISTORY

10588 First released in Issue 4, Version 2.

10589 Issue 5

10590 Moved from X/OPEN UNIX extension to BASE and aligned with *fchmod()* in the POSIX
10591 Realtime Extension. Specifically, the second paragraph of the DESCRIPTION is added and a
10592 second instance of [EINVAL] is defined in the list of optional errors.

10593 Issue 6

10594 The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by stating that
10595 *fchmod()* behavior is unspecified for typed memory objects.

10596 **NAME**

10597 fchown — change owner and group of a file

10598 **SYNOPSIS**

10599 #include <unistd.h>

10600 int fchown(int *fildev*, uid_t *owner*, gid_t *group*);10601 **DESCRIPTION**10602 The *fchown()* function has the same effect as *chown()* except that the file whose owner and group
10603 are changed is specified by the file descriptor *fildev*.10604 **RETURN VALUE**10605 Upon successful completion, *fchown()* shall return 0. Otherwise, it shall return -1 and set *errno* to
10606 indicate the error.10607 **ERRORS**10608 The *fchown()* function shall fail if:10609 [EBADF] The *fildev* argument is not an open file descriptor.10610 [EPERM] The effective user ID does not match the owner of the file or the process does
10611 not have appropriate privilege and `_POSIX_CHOWN_RESTRICTED` indicates
10612 that such privilege is required.10613 [EROFS] The file referred to by *fildev* resides on a read-only file system.10614 The *fchown()* function may fail if:10615 [EINVAL] The owner or group ID is not a value supported by the implementation. The
10616 *fildev* argument refers to a pipe and the implementation disallows execution of
10617 *fchown()* on a pipe.10618 **Notes to Reviewers**10619 *This section with side shading will not appear in the final copy. - Ed.*10620 D3, XSH, ERN 177 states that STREAMS ignore the call, but raises a question
10621 about AF_UNIX sockets in the file system name space.

10622 [EIO] A physical I/O error has occurred.

10623 [EINTR] The *fchown()* function was interrupted by a signal which was caught.10624 **EXAMPLES**10625 **Changing the Current Owner of a File**10626 The following example shows how to change the owner of a file named `/home/cnd/mod1` to
10627 “jones” and the group to “cnd”.10628 The numeric value for the user ID is obtained by extracting the user ID from the user database
10629 entry associated with “jones”. Similarly, the numeric value for the group ID is obtained by
10630 extracting the group ID from the group database entry associated with “cnd”. This example
10631 assumes the calling program has appropriate privileges.

10632 #include <sys/types.h>

10633 #include <unistd.h>

10634 #include <fcntl.h>

10635 #include <pwd.h>

10636 #include <grp.h>

```
10637     struct passwd *pwd;
10638     struct group  *grp;
10639     int           fildes;
10640     ...
10641     fildes = open("/home/cnd/mod1", O_RDWR);
10642     pwd = getpwnam("jones");
10643     grp = getgrnam("cnd");
10644     fchown(fildes, pwd->pw_uid, grp->gr_gid);
```

10645 APPLICATION USAGE

10646 None.

10647 RATIONALE

10648 None.

10649 FUTURE DIRECTIONS

10650 None.

10651 SEE ALSO

10652 *chown()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>

10653 CHANGE HISTORY

10654 First released in Issue 4, Version 2.

10655 Issue 5

10656 Moved from X/OPEN UNIX extension to BASE.

10657 Issue 6

10658 The following changes were made to align with the IEEE P1003.1a draft standard:

10659 • Clarification is added that a call to *fchown()* may not be allowed on a pipe.

10660 The *fchwon()* function is now defined as mandatory.

10661 **NAME**

10662 fclose — close a stream

10663 **SYNOPSIS**

10664 #include <stdio.h>

10665 int fclose(FILE *stream);

10666 **DESCRIPTION**

10667 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 10668 conflict between the requirements described here and the ISO C standard is unintentional. This
 10669 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

10670 The *fclose()* function shall cause the stream pointed to by *stream* to be flushed and the associated
 10671 file to be closed. Any unwritten buffered data for the stream shall be written to the file; any
 10672 unread buffered data shall be discarded. Whether or not the call succeeds, the stream shall be
 10673 disassociated from the file and any buffer set by the *setbuf()* or *setvbuf()* function shall be
 10674 disassociated from the stream. If the associated buffer was automatically allocated, it shall be
 10675 CX deallocated. It shall mark for update the *st_ctime* and *st_mtime* fields of the underlying file, if the
 10676 stream was writable, and if buffered data remains that has not yet been written to the file. The
 10677 *fclose()* function shall perform the equivalent of a *close()* on the file descriptor that is associated
 10678 with the stream pointed to by *stream*.

10679 After the call to *fclose()*, any use of *stream* results in undefined behavior.

10680 **RETURN VALUE**

10681 CX Upon successful completion, *fclose()* shall return 0; otherwise, it shall return EOF and set *errno* to
 10682 indicate the error.

10683 **ERRORS**

10684 The *fclose()* function shall fail if:

10685 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 10686 process would be delayed in the write operation.

10687 CX [EBADF] The file descriptor underlying stream is not valid.

10688 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

10689 XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

10690 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 10691 offset maximum associated with the corresponding stream.

10692 CX [EINTR] The *fclose()* function was interrupted by a signal.

10693 CX [EIO] The process is a member of a background process group attempting to write
 10694 to its controlling terminal, TOSTOP is set, the process is neither ignoring nor
 10695 blocking SIGTTOU, and the process group of the process is orphaned. This
 10696 error may also be returned under implementation-defined conditions.

10697 CX [ENOSPC] There was no free space remaining on the device containing the file.

10698 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 10699 any process. A SIGPIPE signal shall also be sent to the thread.

10700 The *fclose()* function may fail if:

10701 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 10702 capabilities of the device.

10703 **EXAMPLES**

10704 None.

10705 **APPLICATION USAGE**

10706 None.

10707 **RATIONALE**

10708 None.

10709 **FUTURE DIRECTIONS**

10710 None.

10711 **SEE ALSO**

10712 *close()*, *fopen()*, *getrlimit()*, *ulimit()*, the Base Definitions volume of IEEE Std. 1003.1-200x,
10713 <stdio.h>

10714 **CHANGE HISTORY**

10715 First released in Issue 1. Derived from Issue 1 of the SVID.

10716 **Issue 4**

10717 The last sentence of the first paragraph in the DESCRIPTION is changed to say *close()* instead of
10718 *fclose()*. This was an error in Issue 3.

10719 The following paragraph is withdrawn from the DESCRIPTION (by POSIX as well as X/Open)
10720 because of the possibility of causing applications to malfunction, and the impossibility of
10721 implementing these mechanisms for pipes:

10722 If the file is not already at EOF, and the file is one capable of seeking, the file *offset* of the
10723 underlying open file description is adjusted so that the next operation on the open file
10724 description deals with the byte after the last one read from or written to the stream being
10725 closed.

10726 It is replaced with a statement that any subsequent use of *stream* is undefined.

10727 The [EFBIG] error is marked to indicate the extensions.

10728 **Issue 4, Version 2**10729 A cross-reference to *getrlimit()* is added.10730 **Issue 5**

10731 Large File Summit extensions are added.

10732 **Issue 6**

10733 Extensions beyond the ISO C standard are now marked.

10734 The following new requirements on POSIX implementations derive from alignment with the
10735 Single UNIX Specification:

- 10736 • The [EFBIG] error is added as part of the large file support extensions.
- 10737 • The [ENXIO] optional error condition is added.

10738 The DESCRIPTION is updated to note that the stream and any buffer are disassociated whether
10739 or not the call succeeds. This is for alignment with the ISO/IEC 9899:1999 standard.

10740 NAME

10741 fcntl — file control

10742 SYNOPSIS

10743 OH #include <unistd.h>

10744 #include <fcntl.h>

10745 int fcntl(int *fildev*, int *cmd*, ...);

10746 DESCRIPTION

10747 The *fcntl()* function provides for control over open files. The *fildev* argument is a file descriptor.10748 The available values for *cmd* are defined in <fcntl.h>, which include:

10749 **F_DUPFD** Return a new file descriptor which is the lowest numbered available (that is,
10750 not already open) file descriptor greater than or equal to the third argument,
10751 *arg*, taken as an integer of type **int**. The new file descriptor refers to the same
10752 open file description as the original file descriptor, and shares any locks. The
10753 **FD_CLOEXEC** flag associated with the new file descriptor is cleared to keep
10754 the file open across calls to one of the *exec* functions.

10755 **F_GETFD** Get the file descriptor flags defined in <fcntl.h> that are associated with the
10756 file descriptor *fildev*. File descriptor flags are associated with a single file
10757 descriptor and do not affect other file descriptors that refer to the same file.

10758 **F_SETFD** Set the file descriptor flags defined in <fcntl.h>, that are associated with *fildev*,
10759 to the third argument, *arg*, taken as type **int**. If the **FD_CLOEXEC** flag in the
10760 third argument is 0, the file shall remain open across the *exec* functions;
10761 otherwise, the file shall be closed upon successful execution of one of the *exec*
10762 functions.

10763 **F_GETFL** Get the file status flags and file access modes, defined in <fcntl.h>, for the file
10764 description associated with *fildev*. The file access modes can be extracted from
10765 the return value using the mask **O_ACCMODE**, which is defined in <fcntl.h>.
10766 File status flags and file access modes are associated with the file description
10767 and do not affect other file descriptors that refer to the same file with different
10768 open file descriptions.

10769 **F_SETFL** Set the file status flags, defined in <fcntl.h>, for the file description associated
10770 with *fildev* from the corresponding bits in the third argument, *arg*, taken as
10771 type **int**. Bits corresponding to the file access mode and the *oflag* values that
10772 are set in *arg* are ignored. If any bits in *arg* other than those mentioned here are
10773 changed by the application, the result is unspecified.

10774 **F_GETOWN** If *fildev* refers to a socket, get the process or process group ID specified to
10775 receive **SIGURG** signals when out-of-band data is available. Positive values
10776 indicate a process ID; negative values, other than -1, indicate a process group
10777 ID. If *fildev* does not refer to a socket, the results are unspecified.

10778 **F_SETOWN** If *fildev* refers to a socket, set the process or process group ID specified to
10779 receive **SIGURG** signals when out-of-band data is available, using the value of
10780 the third argument, *arg*, taken as type **int**. Positive values indicate a process
10781 ID; negative values, other than -1, indicate a process group ID. If *fildev* does
10782 not refer to a socket, the results are unspecified.

10783 The following values for *cmd* are available for advisory record locking. Record locking is
10784 supported for regular files, and may be supported for other files.

10785 F_GETLK Get the first lock which blocks the lock description pointed to by the third
10786 argument, *arg*, taken as a pointer to type **struct flock**, defined in `<fcntl.h>`.
10787 The information retrieved overwrites the information passed to *fcntl()* in the
10788 structure **flock**. If no lock is found that would prevent this lock from being
10789 created, then the structure shall be left unchanged except for the lock type
10790 which shall be set to F_UNLCK.

10791 F_SETLK Set or clear a file segment lock according to the lock description pointed to by
10792 the third argument, *arg*, taken as a pointer to type **struct flock**, defined in
10793 `<fcntl.h>`. F_SETLK is used to establish shared (or read) locks (F_RDLCK) or
10794 exclusive (or write) locks (F_WRLCK), as well as to remove either type of lock
10795 (F_UNLCK). F_RDLCK, F_WRLCK, and F_UNLCK are defined in `<fcntl.h>`.
10796 If a shared or exclusive lock cannot be set, *fcntl()* shall return immediately
10797 with a return value of -1.

10798 F_SETLKW This command is the same as F_SETLK except that if a shared or exclusive
10799 lock is blocked by other locks, the thread shall wait until the request can be
10800 satisfied. If a signal that is to be caught is received while *fcntl()* is waiting for a
10801 region, *fcntl()* shall be interrupted. Upon return from the signal handler,
10802 *fcntl()* shall return -1 with *errno* set to [EINTR], and the lock operation shall
10803 not be done.

10804 Additional implementation-defined values for *cmd* may be defined in `<fcntl.h>`. Their names
10805 shall start with F_.

10806 When a shared lock is set on a segment of a file, other processes shall be able to set shared locks
10807 on that segment or a portion of it. A shared lock prevents any other process from setting an
10808 exclusive lock on any portion of the protected area. A request for a shared lock shall fail if the
10809 file descriptor was not opened with read access.

10810 An exclusive lock shall prevent any other process from setting a shared lock or an exclusive lock
10811 on any portion of the protected area. A request for an exclusive lock shall fail if the file
10812 descriptor was not opened with write access.

10813 The structure **flock** describes the type (*l_type*), starting offset (*l_whence*), relative offset (*l_start*),
10814 size (*l_len*), and process ID (*l_pid*) of the segment of the file to be affected.

10815 The value of *l_whence* is {SEEK_SET}, {SEEK_CUR}, or {SEEK_END}, to indicate that the relative
10816 offset *l_start* bytes shall be measured from the start of the file, current position, or end of the file,
10817 respectively. The value of *l_len* is the number of consecutive bytes to be locked. The value of
10818 *l_len* may be negative (where the definition of **off_t** permits negative values of *l_len*). The *l_pid*
10819 field is only used with F_GETLK to return the process ID of the process holding a blocking lock.
10820 After a successful F_GETLK request, when a blocking lock is found, the values returned in the
10821 **flock** structure shall be as follows:

10822 *l_type* Type of blocking lock found.

10823 *l_whence* {SEEK_SET}.

10824 *l_start* Start of the blocking lock.

10825 *l_len* Length of the blocking lock.

10826 *l_pid* Process ID of the process that holds the blocking lock.

10827 If the command is F_SETLKW and the process must wait for another process to release a lock,
10828 then the range of bytes to be locked shall be determined before the *fcntl()* function blocks. If the
10829 file size or file descriptor seek offset change while *fcntl()* is blocked, this shall not affect the
10830 range of bytes locked.

10831 If *l_len* is positive, the area affected starts at *l_start* and ends at *l_start*+ *l_len*-1. If *l_len* is
 10832 negative, the area affected starts at *l_start*+ *l_len* and ends at *l_start*-1. Locks may start and
 10833 extend beyond the current end of a file, but the application shall ensure that they are not
 10834 negative relative to the beginning of the file. A lock shall be set to extend to the largest possible
 10835 value of the file offset for that file by setting *l_len* to 0. If such a lock also has *l_start* set to 0 and
 10836 *l_whence* is set to {SEEK_SET}, the whole file shall be locked.

10837 There shall be at most one type of lock set for each byte in the file. Before a successful return
 10838 from an F_SETLK or an F_SETLKW request when the calling process has previously existing
 10839 locks on bytes in the region specified by the request, the previous lock type for each byte in the
 10840 specified region shall be replaced by the new lock type. As specified above under the
 10841 descriptions of shared locks and exclusive locks, an F_SETLK or an F_SETLKW request
 10842 (respectively) shall fail or block when another process has existing locks on bytes in the specified
 10843 region and the type of any of those locks conflicts with the type specified in the request.

10844 All locks associated with a file for a given process shall be removed when a file descriptor for
 10845 that file is closed by that process or the process holding that file descriptor terminates. Locks are
 10846 not inherited by a child process.

10847 A potential for deadlock occurs if a process controlling a locked region is put to sleep by
 10848 attempting to lock another process' locked region. If the system detects that sleeping until a
 10849 locked region is unlocked would cause a deadlock, *fcntl()* shall fail with an [EDEADLK] error.

10850 SHM When the file descriptor *fdes* refers to a shared memory object, the behavior of *fcntl()* shall be
 10851 the same as for a regular file except the effect of the following values for the argument *cmd* shall
 10852 be unspecified: F_SETFL, F_GETLK, F_SETLK, and F_SETLKW.

10853 TYM If *fdes* refers to a typed memory object, the result of the *fcntl()* function is unspecified.

10854 An unlock (F_UNLCK) request in which *l_len* is non-zero and the offset of the last byte of the
 10855 requested segment is the maximum value for an object of type *off_t*, when the process has an
 10856 existing lock in which *l_len* is 0 and which includes the last byte of the requested segment, shall
 10857 be treated as a request to unlock from the start of the requested segment with an *l_len* equal to 0.
 10858 Otherwise, an unlock (F_UNLCK) request shall attempt to unlock only the requested segment.

10859 RETURN VALUE

10860 Upon successful completion, the value returned shall depend on *cmd* as follows:

10861	F_DUPFD	A new file descriptor.
10862	F_GETFD	Value of flags defined in <fcntl.h>. The return value shall not be negative.
10863	F_SETFD	Value other than -1.
10864	F_GETFL	Value of file status flags and access modes. The return value is not negative.
10865	F_SETFL	Value other than -1.
10866	F_GETLK	Value other than -1.
10867	F_SETLK	Value other than -1.
10868	F_SETLKW	Value other than -1.
10869	F_GETOWN	Value of the socket owner process or process group; this will not be -1.
10870	F_SETOWN	Value other than -1.
10871		Otherwise, -1 shall be returned and <i>errno</i> set to indicate the error.

10872 **ERRORS**

10873 The *fcntl()* function shall fail if:

10874 [EACCES] or [EAGAIN]

10875 The *cmd* argument is F_SETLK; the type of lock (*l_type*) is a shared (F_RDLCK)
 10876 or exclusive (F_WRLCK) lock and the segment of a file to be locked is already
 10877 exclusive-locked by another process, or the type is an exclusive lock and some
 10878 portion of the segment of a file to be locked is already shared-locked or
 10879 exclusive-locked by another process.

10880 [EBADF]

10881 The *fildev* argument is not a valid open file descriptor, or the argument *cmd* is
 10882 F_SETLK or F_SETLKW, the type of lock, *l_type*, is a shared lock (F_RDLCK),
 10883 and *fildev* is not a valid file descriptor open for reading, or the type of lock
 10884 *l_type*, is an exclusive lock (F_WRLCK), and *fildev* is not a valid file descriptor
 open for writing.

10885 [EINTR]

The *cmd* argument is F_SETLKW and the function was interrupted by a signal.

10886 [EINVAL]

10887 The *cmd* argument is invalid, or the *cmd* argument is F_DUPFD and *arg* is
 10888 negative or greater than or equal to {OPEN_MAX}, or the *cmd* argument is
 10889 F_GETLK, F_SETLK, or F_SETLKW and the data pointed to by *arg* is not valid,
 or *fildev* refers to a file that does not support locking.

10890 [EMFILE]

10891 The argument *cmd* is F_DUPFD and {OPEN_MAX} file descriptors are
 10892 currently open in the calling process, or no file descriptors greater than or
 equal to *arg* are available.

10893 [ENOLCK]

10894 The argument *cmd* is F_SETLK or F_SETLKW and satisfying the lock or unlock
 10895 request would result in the number of locked regions in the system exceeding
 a system-imposed limit.

10896 [EOVERFLOW] One of the values to be returned cannot be represented correctly.

10897 [EOVERFLOW] The *cmd* argument is F_GETLK, F_SETLK, or F_SETLKW and the smallest or,
 10898 if *l_len* is non-zero, the largest offset of any byte in the requested segment
 10899 cannot be represented correctly in an object of type **off_t**.

10900 The *fcntl()* function may fail if:

10901 [EDEADLK]

10902 The *cmd* argument is F_SETLKW, the lock is blocked by some lock from
 10903 another process and putting the calling process to sleep, waiting for that lock
 to become free would cause a deadlock.

10904 **EXAMPLES**

10905 None.

10906 **APPLICATION USAGE**

10907 None.

10908 **RATIONALE**

10909 The ellipsis in the SYNOPSIS is the syntax specified by the ISO C standard for a variable number
 10910 of arguments. It is used because System V uses pointers for the implementation of file locking
 10911 functions.

10912 The *arg* values to F_GETFD, F_SETFD, F_GETFL, and F_SETFL all represent flag values to allow
 10913 for future growth. Applications using these functions should do a read-modify-write operation
 10914 on them, rather than assuming that only the values defined by this volume of
 10915 IEEE Std. 1003.1-200x are valid. It is a common error to forget this, particularly in the case of
 10916 F_SETFD.

10917 This volume of IEEE Std. 1003.1-200x permits concurrent read and write access to file data using
10918 the *fcntl()* function; this is a change from the 1984 /usr/group standard and early proposals.
10919 Without concurrency controls, this feature may not be fully utilized without occasional loss of
10920 data.

10921 Data losses occur in several ways. One case occurs when several processes try to update the
10922 same record, without sequencing controls; several updates may occur in parallel and the last
10923 writer “wins”. Another case is a bit-tree or other internal list-based database that is undergoing
10924 reorganization. Without exclusive use to the tree segment by the updating process, other reading
10925 processes chance getting lost in the database when the index blocks are split, condensed,
10926 inserted, or deleted. While *fcntl()* is useful for many applications, it is not intended to be overly
10927 general and does not handle the bit-tree example well.

10928 This facility is only required for regular files because it is not appropriate for many devices such
10929 as terminals and network connections.

10930 Since *fcntl()* works with “any file descriptor associated with that file, however it is obtained”,
10931 the file descriptor may have been inherited through a *fork()* or *exec* operation and thus may
10932 affect a file that another process also has open.

10933 The use of the open file description to identify what to lock requires extra calls and presents
10934 problems if several processes are sharing an open file description, but there are too many
10935 implementations of the existing mechanism for this volume of IEEE Std. 1003.1-200x to use
10936 different specifications.

10937 Another consequence of this model is that closing any file descriptor for a given file (whether or
10938 not it is the same open file description that created the lock) causes the locks on that file to be
10939 relinquished for that process. Equivalently, any close for any file/process pair relinquishes the
10940 locks owned on that file for that process. But note that while an open file description may be
10941 shared through *fork()*, locks are not inherited through *fork()*. Yet locks may be inherited through
10942 one of the *exec* functions.

10943 The identification of a machine in a network environment is outside of the scope of this volume
10944 of IEEE Std. 1003.1-200x. Thus, an *_l_sysid* member, such as found in System V, is not included in
10945 the locking structure.

10946 Before successful return from an *F_SETLK* or *F_SETLKW* request, the previous lock type for
10947 each byte in the specified region shall be replaced by the new lock type. This can result in a
10948 previously locked region being split into smaller regions. If this would cause the number of
10949 regions being held by all processes in the system to exceed a system-imposed limit, the *fcntl()*
10950 function shall return -1 with *errno* set to [ENOLCK].

10951 Mandatory locking was a major feature of the 1984 /usr/group standard.

10952 For advisory file record locking to be effective, all processes that have access to a file must
10953 cooperate and use the advisory mechanism before doing I/O on the file. Enforcement-mode
10954 record locking is important when it cannot be assumed that all processes are cooperating. For
10955 example, if one user uses an editor to update a file at the same time that a second user executes
10956 another process that updates the same file and if only one of the two processes is using advisory
10957 locking, the processes are not cooperating. Enforcement-mode record locking would protect
10958 against accidental collisions.

10959 Secondly, advisory record locking requires a process using locking to bracket each I/O operation
10960 with lock (or test) and unlock operations. With enforcement-mode file and record locking, a
10961 process can lock the file once and unlock when all I/O operations have been completed.
10962 Enforcement-mode record locking provides a base that can be enhanced; for example, with
10963 sharable locks. That is, the mechanism could be enhanced to allow a process to lock a file so

- 10964 other processes could read it, but none of them could write it.
- 10965 Mandatory locks were omitted for several reasons:
- 10966 1. Mandatory lock setting was done by multiplexing the set-group-ID bit in most
10967 implementations; this was confusing, at best.
 - 10968 2. The relationship to file truncation as supported in 4.2 BSD was not well specified.
 - 10969 3. Any publicly readable file could be locked by anyone. Many historical implementations
10970 keep the password database in a publicly readable file. A malicious user could thus
10971 prohibit logins. Another possibility would be to hold open a long-distance telephone line.
 - 10972 4. Some demand-paged historical implementations offer memory mapped files, and
10973 enforcement cannot be done on that type of file.
- 10974 Since sleeping on a region is interrupted with any signal, *alarm()* may be used to provide a
10975 timeout facility in applications requiring it. This is useful in deadlock detection. Because
10976 implementation of full deadlock detection is not always feasible, the [EDEADLK] error was
10977 made optional.
- 10978 **FUTURE DIRECTIONS**
- 10979 None.
- 10980 **SEE ALSO**
- 10981 *close()*, *exec*, *open()*, *sigaction()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<fcntl.h>**,
10982 **<signal.h>**, **<unistd.h>**
- 10983 **CHANGE HISTORY**
- 10984 First released in Issue 1. Derived from Issue 1 of the SVID.
- 10985 **Issue 4**
- 10986 The **<sys/types.h>** and **<unistd.h>** headers are now marked as optional (OH); these headers do
10987 not need to be included on XSI-conformant systems.
- 10988 In the DESCRIPTION, sentences describing behavior when *l_len* is negative are marked as an
10989 extension, and the description of locks is corrected to make it a requirement on the application.
- 10990 The following change is incorporated for alignment with the ISO POSIX-1 standard:
- 10991 • In the DESCRIPTION, the meaning of a successful F_SETLK or F_SETLKW request is
10992 clarified, after a POSIX Request for Interpretation.
- 10993 **Issue 5**
- 10994 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
10995 Threads Extension.
- 10996 Large File Summit extensions are added.
- 10997 **Issue 6**
- 10998 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.
- 10999 The following new requirements on POSIX implementations derive from alignment with the
11000 Single UNIX Specification:
- 11001 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
11002 required for conforming implementations of previous POSIX specifications, it was not
11003 required for UNIX applications.
 - 11004 • In the DESCRIPTION, sentences describing behavior when *l_len* is negative are now
11005 mandated, and the description of unlock (F_UNLOCK) when *l_len* is non-negative is
11006 mandated.

- 11007 • In the ERRORS section, the [EINVAL] error condition has the case mandated when the *cmd* is
11008 invalid, and two [EOVERFLOW] error conditions are added.
- 11009 The F_GETOWN and F_SETOWN values are added for sockets.
- 11010 The following changes were made to align with the IEEE P1003.1a draft standard:
- 11011 • Clarification is added that the extent of the bytes locked is determined prior to the blocking
11012 action.
- 11013 The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by specifying that
11014 *fcntl()* results are unspecified for typed memory objects.
- 11015 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

11016 **NAME**11017 fcvt — convert a floating-point number to a string (**LEGACY**)11018 **SYNOPSIS**

11019 XSI #include <stdlib.h>

11020 char *fcvt(double *value*, int *ndigit*, int *restrict *decpt*,
11021 int *restrict *sign*);

11022

11023 **DESCRIPTION**11024 Refer to *ecvt()*.

11025 **NAME**11026 fdatasync — synchronize the data of a file (**REALTIME**)11027 **SYNOPSIS**

11028 SIO #include <unistd.h>

11029 int fdatasync(int *fil*des);

11030

11031 **DESCRIPTION**11032 The *fdatasync()* function shall force all currently queued I/O operations associated with the file
11033 indicated by file descriptor *fil*des to the synchronized I/O completion state.11034 The functionality is as described for *fsync()* (with the symbol `_POSIX_SYNCHRONIZED_IO`
11035 defined), with the exception that all I/O operations shall be completed as defined for
11036 synchronized I/O data integrity completion.11037 **RETURN VALUE**11038 If successful, the *fdatasync()* function shall return the value 0; otherwise, the function shall return
11039 the value `-1` and set *errno* to indicate the error. If the *fdatasync()* function fails, outstanding I/O
11040 operations are not guaranteed to have been completed.11041 **ERRORS**11042 The *fdatasync()* function shall fail if:11043 [EBADF] The *fil*des argument is not a valid file descriptor open for writing. |

11044 [EINVAL] This implementation does not support synchronized I/O for this file. |

11045 In the event that any of the queued I/O operations fail, *fdatasync()* shall return the error
11046 conditions defined for *read()* and *write()*.11047 **EXAMPLES**

11048 None.

11049 **APPLICATION USAGE**

11050 None.

11051 **RATIONALE**

11052 None.

11053 **FUTURE DIRECTIONS**

11054 None.

11055 **SEE ALSO**11056 *ai*o_ *fsync()*, *fcntl()*, *fsync()*, *open()*, *read()*, *write()*, the Base Definitions volume of |
11057 IEEE Std. 1003.1-200x, <unistd.h>11058 **CHANGE HISTORY**

11059 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

11060 **Issue 6**11061 The [ENOSYS] error condition has been removed as stubs need not be provided if an
11062 implementation does not support the Synchronized Input and Output option. |11063 The *fdatasync()* function is marked as part of the Synchronized Input and Output option. |

11064 **NAME**11065 fdetach — detach a name from a STREAMS-based file descriptor (**STREAMS**)11066 **SYNOPSIS**

11067 XSR #include <stropts.h>

11068 int fdetach(const char *path);

11069

11070 **DESCRIPTION**

11071 The *fdetach()* function detaches a STREAMS-based file from the file to which it was attached by a
 11072 previous call to *fattach()*. The *path* argument points to the path name of the attached STREAMS
 11073 file. The application shall ensure that the process has appropriate privileges or be the owner of
 11074 the file. A successful call to *fdetach()* shall cause all path names that named the attached
 11075 STREAMS file to again name the file to which the STREAMS file was attached. All subsequent
 11076 operations on *path* shall operate on the underlying file and not on the STREAMS file.

11077 All open file descriptions established while the STREAMS file was attached to the file referenced
 11078 by *path* shall still refer to the STREAMS file after the *fdetach()* has taken effect.

11079 If there are no open file descriptors or other references to the STREAMS file, then a successful
 11080 call to *fdetach()* shall have the same effect as performing the last *close()* on the attached file.

11081 **RETURN VALUE**

11082 Upon successful completion, *fdetach()* shall return 0; otherwise, it shall return -1 and set *errno* to
 11083 indicate the error.

11084 **ERRORS**11085 The *fdetach()* function shall fail if:

- | | | | |
|-------|----------------|--|--|
| 11086 | [EACCES] | Search permission is denied on a component of the path prefix. | |
| 11087 | [EINVAL] | The <i>path</i> argument names a file that is not currently attached. | |
| 11088 | [ELOOP] | A loop exists in symbolic links encountered during resolution of the <i>path</i> | |
| 11089 | | argument. | |
| 11090 | [ENAMETOOLONG] | | |
| 11091 | | The size of a path name exceeds {PATH_MAX} or a path name component is | |
| 11092 | | longer than {NAME_MAX}. | |
| 11093 | [ENOENT] | A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string. | |
| 11094 | [ENOTDIR] | A component of the path prefix is not a directory. | |
| 11095 | [EPERM] | The effective user ID is not the owner of <i>path</i> and the process does not have | |
| 11096 | | appropriate privileges. | |

11097 The *fdetach()* function may fail if:

- | | | | |
|-------|----------------|---|--|
| 11098 | [ELOOP] | More than {SYMLOOP_MAX} symbolic links were encountered during | |
| 11099 | | resolution of the <i>path</i> argument. | |
| 11100 | [ENAMETOOLONG] | | |
| 11101 | | Path name resolution of a symbolic link produced an intermediate result | |
| 11102 | | whose length exceeds {PATH_MAX}. | |

11103 **EXAMPLES**11104 **Detaching a File**

11105 The following example detaches the STREAMS-based file **/tmp/named-STREAM** from the file to
11106 which it was attached by a previous, successful call to *fattach()*. Subsequent calls to open this
11107 file refer to the underlying file, not to the STREAMS file.

```
11108 #include <stropts.h>
11109 ...
11110     char *filename = "/tmp/named-STREAM";
11111     int ret;
11112     ret = fdetach(filename);
```

11113 **APPLICATION USAGE**

11114 None.

11115 **RATIONALE**

11116 None.

11117 **FUTURE DIRECTIONS**

11118 None.

11119 **SEE ALSO**

11120 *fattach()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stropts.h>

11121 **CHANGE HISTORY**

11122 First released in Issue 4, Version 2.

11123 **Issue 5**

11124 Moved from X/OPEN UNIX extension to BASE.

11125 **Issue 6**

11126 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

11127 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
11128 [ELOOP] error condition is added.

11129 **NAME**

11130 `fdim`, `fdimf`, `fdiml` — compute positive difference between two floating-point numbers

11131 **SYNOPSIS**

11132 `#include <math.h>`

11133 `double fdim(double x, double y);`

11134 `float fdimf(float x, float y);`

11135 `long double fdiml(long double x, long double y);`

11136 **DESCRIPTION**

11137 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
11138 conflict between the requirements described here and the ISO C standard is unintentional. This
11139 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11140 These functions shall determine the positive difference between their arguments. If x is greater
11141 than y , $x-y$ is returned. If x is less than or equal to y , $+0$ is returned.

11142 An application wishing to check for error situations should set *errno* to 0 before calling these
11143 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

11144 **RETURN VALUE**

11145 Upon successful completion, these functions shall return the positive difference value.

11146 If x or y is NaN, NaN shall be returned and *errno* may be set to [EDOM].

11147 If the magnitude of the result is too large or too small, the numeric result is unspecified and *errno*
11148 may be set to [ERANGE].

11149 **ERRORS**

11150 These functions may fail if:

11151 [EDOM] The value of x or y is NaN.

11152 [ERANGE] The magnitude of the result is too large or too small.

11153 **EXAMPLES**

11154 None.

11155 **APPLICATION USAGE**

11156 None.

11157 **RATIONALE**

11158 None.

11159 **FUTURE DIRECTIONS**

11160 None.

11161 **SEE ALSO**

11162 *fmax()*, *fmin()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<math.h>`

11163 **CHANGE HISTORY**

11164 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11165 **NAME**

11166 fdopen — associate a stream with a file descriptor

11167 **SYNOPSIS**

11168 #include <stdio.h>

11169 FILE *fdopen(int *fil-des*, const char **mode*);11170 **DESCRIPTION**11171 The *fdopen()* function shall associate a stream with a file descriptor.11172 The *mode* argument is a character string having one of the following values:11173 *r* or *rb* Open a file for reading.11174 *w* or *wb* Open a file for writing.11175 *a* or *ab* Open a file for writing at end of file.11176 *r+* or *rb+* or *r+b* Open a file for update (reading and writing).11177 *w+* or *wb+* or *w+b* Open a file for update (reading and writing).11178 *a+* or *ab+* or *a+b* Open a file for update (reading and writing) at end of file.11179 The meaning of these flags is exactly as specified in *fopen()*, except that modes beginning with *w*
11180 do not cause truncation of the file.11181 Additional values for the *mode* argument may be supported by an implementation.11182 The application shall ensure that the mode of the stream as expressed by the *type* argument is
11183 allowed by the file access mode of the open file description to which *fil-des* refers. The file
11184 position indicator associated with the new stream is set to the position indicated by the file
11185 offset associated with the file descriptor.11186 The error and end-of-file indicators for the stream shall be cleared. The *fdopen()* function may
11187 cause the *st_atime* field of the underlying file to be marked for update.11188 SHM If *fil-des* refers to a shared memory object, the result of the *fdopen()* function is unspecified.11189 TYM If *fil-des* refers to a typed memory object, the result of the *fdopen()* function is unspecified.11190 The *fdopen()* function shall preserve the offset maximum previously set for the open file
11191 description corresponding to *fil-des*.11192 **RETURN VALUE**11193 Upon successful completion, *fdopen()* shall return a pointer to a stream; otherwise, a null pointer
11194 shall be returned and *errno* set to indicate the error.11195 **ERRORS**11196 The *fdopen()* function may fail if:11197 [EBADF] The *fil-des* argument is not a valid file descriptor. |11198 [EINVAL] The *mode* argument is not a valid mode. |

11199 [EMFILE] {FOPEN_MAX} streams are currently open in the calling process. |

11200 [EMFILE] {STREAM_MAX} streams are currently open in the calling process. |

11201 [ENOMEM] Insufficient space to allocate a buffer. |

11202 **EXAMPLES**

11203 None.

11204 **APPLICATION USAGE**

11205 File descriptors are obtained from calls like *open()*, *dup()*, *creat()*, or *pipe()*, which open files but
 11206 do not return streams.

11207 **RATIONALE**

11208 The file descriptor may have been obtained from *open()*, *creat()*, *pipe()*, *dup()*, or *fcntl()*;
 11209 inherited through *fork()* or *exec*; or perhaps obtained by implementation-defined means, such as
 11210 the 4.3 BSD *socket()* call.

11211 The meanings of the *type* arguments of *fdopen()* and *fopen()* differ. With *fdopen()*, open for write
 11212 (*w* or *w+*) does not truncate, and append (*a* or *a+*) cannot create for writing. There is no need for *b*
 11213 in the format due to the equivalence of binary and text files in this volume of
 11214 IEEE Std. 1003.1-200x. Although not explicitly required by this volume of IEEE Std. 1003.1-200x,
 11215 a good implementation of append (*a*) mode would cause the `O_APPEND` flag to be set.

11216 **FUTURE DIRECTIONS**

11217 None.

11218 **SEE ALSO**

11219 *fclose()*, *fopen()*, *open()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<stdio.h>`, Section
 11220 2.5.1 (on page 535)

11221 **CHANGE HISTORY**

11222 First released in Issue 1. Derived from Issue 1 of the SVID.

11223 **Issue 4**

11224 In the DESCRIPTION, the use and settings of the *mode* argument are changed to include binary
 11225 streams and are marked as extensions.

11226 All errors identified in the ERRORS section are marked as extensions, and the [EMFILE] error is
 11227 added.

11228 The APPLICATION USAGE section is added.

11229 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 11230 • The type of argument *mode* is changed from `char*` to `const char*`.

11231 **Issue 5**

11232 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

11233 Large File Summit extensions are added.

11234 **Issue 6**

11235 The following new requirements on POSIX implementations derive from alignment with the
 11236 Single UNIX Specification:

- 11237 • In the DESCRIPTION, the use and setting of the *mode* argument are changed to include
 11238 binary streams.

- 11239 • In the DESCRIPTION, text is added for large file support to indicate setting of the offset
 11240 maximum in the open file description.

- 11241 • All errors identified in the ERRORS section are added.

- 11242 • In the DESCRIPTION, text is added that the *fdopen()* function may cause *st_atime* to be
 11243 updated.

- 11244 The following changes were made to align with the IEEE P1003.1a draft standard:
- 11245 • Clarification is added that it is the responsibility of the application to ensure that the mode is
11246 compatible with the open file descriptor.
- 11247 The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by specifying that |
11248 *fdopen()* results are unspecified for typed memory objects.

11249 **NAME**

11250 `feclearexcept` — clear floating-point exception

11251 **SYNOPSIS**

11252 `#include <fenv.h>`

11253 `void feclearexcept(int excepts);`

11254 **DESCRIPTION**

11255 *cx* The functionality described on this reference page is aligned with the ISO C standard. Any
11256 conflict between the requirements described here and the ISO C standard is unintentional. This
11257 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11258 The `feclearexcept()` function shall clear the supported floating-point exceptions represented by
11259 *excepts*.

11260 **RETURN VALUE**

11261 None.

11262 **ERRORS**

11263 No errors are defined.

11264 **EXAMPLES**

11265 None.

11266 **APPLICATION USAGE**

11267 None.

11268 **RATIONALE**

11269 None.

11270 **FUTURE DIRECTIONS**

11271 None.

11272 **SEE ALSO**

11273 `fegetexceptflag()`, `feraiseexcept()`, `fesetexceptflag()`, `fetestexcept()`, the Base Definitions volume of
11274 IEEE Std. 1003.1-200x, `<fenv.h>`

11275 **CHANGE HISTORY**

11276 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11277 **NAME**

11278 fegetenv, fesetenv — get and set current floating-point environment

11279 **SYNOPSIS**

11280 #include <fenv.h>

11281 void fegetenv(fenv_t *envp);

11282 void fesetenv(const fenv_t *envp);

11283 **DESCRIPTION**

11284 cx The functionality described on this reference page is aligned with the ISO C standard. Any
11285 conflict between the requirements described here and the ISO C standard is unintentional. This
11286 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11287 The *fegetenv()* function shall store the current floating-point environment in the object pointed to
11288 by *envp*.

11289 The *fesetenv()* function shall establish the floating-point environment represented by the object
11290 pointed to by *envp*. The argument *envp* shall point to an object set by a call to *fegetenv()* or
11291 *fehldexcept()*, or equal a floating-point environment macro. The *fesetenv()* function does not
11292 raise floating-point exceptions, but only installs the state of the floating-point status flags
11293 represented through its argument.

11294 **RETURN VALUE**

11295 None.

11296 **ERRORS**

11297 No errors are defined.

11298 **EXAMPLES**

11299 None.

11300 **APPLICATION USAGE**

11301 None.

11302 **RATIONALE**

11303 None.

11304 **FUTURE DIRECTIONS**

11305 None.

11306 **SEE ALSO**

11307 *fehldexcept()*, *feupdateenv()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <fenv.h>

11308 **CHANGE HISTORY**

11309 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11310 **NAME**

11311 fegetexceptflag, fesetexceptflag — get and set floating-point status flags

11312 **SYNOPSIS**

11313 #include <fenv.h>

11314 void fegetexceptflag(fexcept_t *flagp, int excepts);

11315 void fesetexceptflag(const fexcept_t *flagp, int excepts);

11316 **DESCRIPTION**

11317 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
11318 conflict between the requirements described here and the ISO C standard is unintentional. This
11319 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11320 The *fegetexceptflag()* function shall store an implementation-defined representation of the states
11321 of the floating-point status flags indicated by the argument *excepts* in the object pointed to by the
11322 argument *flagp*.

11323 The *fesetexceptflag()* function shall set the floating-point status flags indicated by the argument
11324 *excepts* to the states stored in the object pointed to by *flagp*. The value pointed to by *flagp* shall
11325 have been set by a previous call to *fegetexceptflag()* whose second argument represented at least
11326 those floating-point exceptions represented by the argument *excepts*. This function does not
11327 raise floating-point exceptions, but only sets the state of the flags.

11328 **RETURN VALUE**

11329 None.

11330 **ERRORS**

11331 No errors are defined.

11332 **EXAMPLES**

11333 None.

11334 **APPLICATION USAGE**

11335 None.

11336 **RATIONALE**

11337 None.

11338 **FUTURE DIRECTIONS**

11339 None.

11340 **SEE ALSO**

11341 *feclearexcept()*, *feraiseexcept()*, *fetestexcept()*, the Base Definitions volume of IEEE Std. 1003.1-200x,
11342 <fenv.h>

11343 **CHANGE HISTORY**

11344 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11345 **NAME**

11346 fegetround, fesetround — get and set current rounding direction

11347 **SYNOPSIS**

11348 #include <fenv.h>

11349 int fegetround(void);

11350 int fesetround(int round);

11351 **DESCRIPTION**

11352 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 11353 conflict between the requirements described here and the ISO C standard is unintentional. This
 11354 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11355 The *fegetround()* function shall get the current rounding direction.

11356 The *fesetround()* function shall establish the rounding direction represented by its argument
 11357 *round*. If the argument is not equal to the value of a rounding direction macro, the rounding
 11358 direction is not changed.

11359 **RETURN VALUE**

11360 The *fegetround()* function shall return the value of the rounding direction macro representing the
 11361 current rounding direction or a negative value if there is no such rounding direction macro or
 11362 the current rounding direction is not determinable.

11363 The *fesetround()* function shall return a zero value if and only if the argument is equal to a
 11364 rounding direction macro (that is, if and only if the requested rounding direction was
 11365 established).

11366 **ERRORS**

11367 No errors are defined.

11368 **EXAMPLES**

11369 The following example saves, sets, and restores the rounding direction, reporting an error and
 11370 aborting if setting the rounding direction fails:

```

11371 #include <fenv.h>
11372 #include <assert.h>
11373 void f(int round_dir)
11374 {
11375     #pragma STDC FENV_ACCESS ON
11376     int save_round;
11377     int setround_ok;
11378     save_round = fegetround();
11379     setround_ok = fesetround(round_dir);
11380     assert(setround_ok == 0);
11381     /* ... */
11382     fesetround(save_round);
11383     /* ... */
11384 }
```

11385 **APPLICATION USAGE**

11386 None.

11387 **RATIONALE**

11388 None.

11389 **FUTURE DIRECTIONS**

11390 None.

11391 **SEE ALSO**

11392 The Base Definitions volume of IEEE Std. 1003.1-200x, <fenv.h>

11393 **CHANGE HISTORY**

11394 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

11395 **NAME**

11396 feholdexcept — save current floating-point environment

11397 **SYNOPSIS**

11398 #include <fenv.h>

11399 int feholdexcept(fenv_t *envp);

11400 **DESCRIPTION**

11401 cx The functionality described on this reference page is aligned with the ISO C standard. Any
11402 conflict between the requirements described here and the ISO C standard is unintentional. This
11403 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11404 The *feholdexcept()* function shall save the current floating-point environment in the object
11405 pointed to by *envp*, clear the floating-point status flags, and then install a non-stop (continue on
11406 floating-point exceptions) mode, if available, for all floating-point exceptions.

11407 **RETURN VALUE**

11408 The *feholdexcept()* function shall return zero if and only if non-stop floating-point exception
11409 handling was successfully installed.

11410 **ERRORS**

11411 No errors are defined.

11412 **EXAMPLES**

11413 None.

11414 **APPLICATION USAGE**

11415 None.

11416 **RATIONALE**

11417 The *feholdexcept()* function should be effective on typical IEC 60559:1989 standard
11418 implementations which have the default non-stop mode and at least one other mode for trap
11419 handling or aborting. If the implementation provides only the non-stop mode, then installing the
11420 non-stop mode is trivial.

11421 **FUTURE DIRECTIONS**

11422 None.

11423 **SEE ALSO**

11424 *fegetenv()*, *fesetenv()*, *feupdateenv()*, the Base Definitions volume of IEEE Std. 1003.1-200x,
11425 <fenv.h>

11426 **CHANGE HISTORY**

11427 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11428 **NAME**

11429 feof — test end-of-file indicator on a stream

11430 **SYNOPSIS**

11431 #include <stdio.h>

11432 int feof(FILE *stream);

11433 **DESCRIPTION**

11434 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 11435 conflict between the requirements described here and the ISO C standard is unintentional. This
 11436 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11437 The *feof()* function shall test the end-of-file indicator for the stream pointed to by *stream*.

11438 **RETURN VALUE**

11439 The *feof()* function shall return non-zero if and only if the end-of-file indicator is set for *stream*.

11440 **ERRORS**

11441 No errors are defined.

11442 **EXAMPLES**

11443 None.

11444 **APPLICATION USAGE**

11445 None.

11446 **RATIONALE**

11447 None.

11448 **FUTURE DIRECTIONS**

11449 None.

11450 **SEE ALSO**

11451 *clearerr()*, *ferror()*, *fopen()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>

11452 **CHANGE HISTORY**

11453 First released in Issue 1. Derived from Issue 1 of the SVID.

11454 **Issue 4**

11455 The ERRORS section is rewritten, such that no error return values are now defined for this
 11456 function.

11457 **NAME**11458 `feraiseexcept` — raise floating-point exception11459 **SYNOPSIS**11460 `#include <fenv.h>`11461 `void feraiseexcept(int excepts);`11462 **DESCRIPTION**

11463 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
11464 conflict between the requirements described here and the ISO C standard is unintentional. This
11465 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11466 The *feraiseexcept()* function shall raise the supported floating-point exceptions represented by
11467 the argument *excepts*. The order in which these floating-point exceptions are raised is
11468 unspecified. Whether the *feraiseexcept()* function additionally raises the inexact floating-point
11469 exception whenever it raises the overflow or underflow floating-point exception is
11470 implementation-defined.

11471 **RETURN VALUE**

11472 None.

11473 **ERRORS**

11474 No errors are defined.

11475 **EXAMPLES**

11476 None.

11477 **APPLICATION USAGE**

11478 The effect is intended to be similar to that of floating-point exceptions raised by arithmetic
11479 operations. Hence, enabled traps for floating-point exceptions raised by this function are taken.

11480 **RATIONALE**

11481 Raising overflow or underflow is allowed to also raise inexact because on some architectures the
11482 only practical way to raise an exception is to execute an instruction that has the exception as a
11483 side effect. The function is not restricted to accept only valid coincident expressions for atomic
11484 operations, so the function can be used to raise exceptions accrued over several operations.

11485 **FUTURE DIRECTIONS**

11486 None.

11487 **SEE ALSO**

11488 *feclearexcept()*, *fegetexceptflag()*, *fesetexceptflag()*, *fetestexcept()*, the Base Definitions volume of
11489 IEEE Std. 1003.1-200x, `<fenv.h>`

11490 **CHANGE HISTORY**

11491 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11492 **NAME**

11493 ferror — test error indicator on a stream

11494 **SYNOPSIS**

11495 #include <stdio.h>

11496 int ferror(FILE **stream*);

11497 **DESCRIPTION**

11498 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
11499 conflict between the requirements described here and the ISO C standard is unintentional. This
11500 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11501 The *ferror()* function shall test the error indicator for the stream pointed to by *stream*.

11502 **RETURN VALUE**

11503 The *ferror()* function shall return non-zero if and only if the error indicator is set for *stream*.

11504 **ERRORS**

11505 No errors are defined.

11506 **EXAMPLES**

11507 None.

11508 **APPLICATION USAGE**

11509 None.

11510 **RATIONALE**

11511 None.

11512 **FUTURE DIRECTIONS**

11513 None.

11514 **SEE ALSO**

11515 *clearerr()*, *feof()*, *fopen()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**stdio.h**>

11516 **CHANGE HISTORY**

11517 First released in Issue 1. Derived from Issue 1 of the SVID.

11518 **Issue 4**

11519 The **ERRORS** section is rewritten, such that no error return values are now defined for this
11520 function.

11521 **NAME**

11522 fesetenv — set current floating-point environment

11523 **SYNOPSIS**

11524 #include <fenv.h>

11525 void fesetenv(const fenv_t *envp);

11526 **DESCRIPTION**11527 Refer to *fegetenv()*.

11528 **NAME**

11529 fesetexceptflag — set floating-point status flags

11530 **SYNOPSIS**

11531 #include <fenv.h>

11532 void fesetexceptflag(const fexcept_t *flagp, int excepts);

11533 **DESCRIPTION**

11534 Refer to *fegetexceptflag()*.

11535 **NAME**

11536 fesetround — set current rounding direction

11537 **SYNOPSIS**

11538 #include <fenv.h>

11539 int fesetround(int *round*);11540 **DESCRIPTION**11541 Refer to *fegetround()*.

11542 **NAME**

11543 fetestexcept — test floating-point exception flags

11544 **SYNOPSIS**

11545 #include <fenv.h>

11546 int fetestexcept(int *excepts*);11547 **DESCRIPTION**

11548 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
11549 conflict between the requirements described here and the ISO C standard is unintentional. This
11550 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11551 The *fetestexcept()* function shall determine which of a specified subset of the floating-point
11552 exception flags are currently set. The *excepts* argument specifies the floating-point status flags to
11553 be queried.

11554 **RETURN VALUE**

11555 The *fetestexcept()* function shall return the value of the bitwise-inclusive OR of the floating-point
11556 exception macros corresponding to the currently set floating-point exceptions included in
11557 *excepts*.

11558 **ERRORS**

11559 No errors are defined.

11560 **EXAMPLES**

11561 The following example calls function *f()* if an invalid exception is set, and then function *g()* if an
11562 overflow exception is set:

```
11563       #include <fenv.h>
11564       /* ... */
11565       {
11566           #pragma STDC FENV_ACCESS ON
11567           int set_excepts;
11568           feclearexcept(FE_INVALID | FE_OVERFLOW);
11569           // maybe raise exceptions
11570           set_excepts = fetestexcept(FE_INVALID | FE_OVERFLOW);
11571           if (set_excepts & FE_INVALID) f();
11572           if (set_excepts & FE_OVERFLOW) g();
11573        /* ... */
11574       }
```

11575 **APPLICATION USAGE**

11576 None.

11577 **RATIONALE**

11578 None.

11579 **FUTURE DIRECTIONS**

11580 None.

11581 **SEE ALSO**

11582 *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, the Base Definitions volume of
11583 IEEE Std. 1003.1-200x, <fenv.h>

11584 **CHANGE HISTORY**

11585 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

11586 **NAME**

11587 feupdateenv — update floating-point environment

11588 **SYNOPSIS**

11589 #include <fenv.h>

11590 void feupdateenv(const fenv_t *envp);

11591 **DESCRIPTION**

11592 cx The functionality described on this reference page is aligned with the ISO C standard. Any
11593 conflict between the requirements described here and the ISO C standard is unintentional. This
11594 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11595 The *feupdateenv()* function shall save the currently raised floating-point exceptions in its
11596 automatic storage, install the floating-point environment represented by the object pointed to by
11597 *envp*, and then raise the saved floating-point exceptions. The argument *envp* shall point to an
11598 object set by a call to *feholdexcept()* or *fegetenv()*, or equal a floating-point environment macro.

11599 **RETURN VALUE**

11600 None.

11601 **ERRORS**

11602 No errors are defined.

11603 **EXAMPLES**

11604 The following example shows sample code to hide spurious underflow floating-point
11605 exceptions:

```
11606 #include <fenv.h>
11607 double f(double x)
11608 {
11609     #pragma STDC FENV_ACCESS ON
11610     double result;
11611     fenv_t save_env;
11612     feholdexcept(&save_env);
11613     // compute result
11614     if (/* test spurious underflow */)
11615         feclearexcept(FE_UNDERFLOW);
11616     feupdateenv(&save_env);
11617     return result;
11618 }
```

11619 **APPLICATION USAGE**

11620 None.

11621 **RATIONALE**

11622 None.

11623 **FUTURE DIRECTIONS**

11624 None.

11625 **SEE ALSO**11626 *fegetenv()*, *feholdexcept()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <fenv.h>11627 **CHANGE HISTORY**

11628 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11629 **NAME**

11630 fflush — flush a stream

11631 **SYNOPSIS**

11632 #include <stdio.h>

11633 int fflush(FILE *stream);

11634 **DESCRIPTION**

11635 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11636 conflict between the requirements described here and the ISO C standard is unintentional. This
 11637 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11638 If *stream* points to an output stream or an update stream in which the most recent operation was
 11639 CX not input, *fflush()* causes any unwritten data for that stream to be written to the file, and the
 11640 *st_ctime* and *st_mtime* fields of the underlying file are marked for update.

11641 If *stream* is a null pointer, *fflush()* shall perform this flushing action on all streams for which the
 11642 behavior is defined above.

11643 **RETURN VALUE**

11644 Upon successful completion, *fflush()* shall return 0; otherwise, it shall set the error indicator for
 11645 CX the stream, return EOF, and set *errno* to indicate the error.

11646 **ERRORS**11647 The *fflush()* function shall fail if:

11648 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 11649 process would be delayed in the write operation.

11650 CX [EBADF] The file descriptor underlying *stream* is not valid.

11651 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

11652 XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

11653 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 11654 offset maximum associated with the corresponding stream.

11655 CX [EINTR] The *fflush()* function was interrupted by a signal.

11656 CX [EIO] The process is a member of a background process group attempting to write
 11657 to its controlling terminal, TOSTOP is set, the process is neither ignoring nor
 11658 blocking SIGTTOU, and the process group of the process is orphaned. This
 11659 error may also be returned under implementation-defined conditions.

11660 CX [ENOSPC] There was no free space remaining on the device containing the file.

11661 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 11662 any process. A SIGPIPE signal shall also be sent to the thread.

11663 The *fflush()* function may fail if:

11664 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 11665 capabilities of the device.

11666 **EXAMPLES**11667 **Sending Prompts to Standard Output**

11668 The following example uses *printf()* calls to print a series of prompts for information the user
11669 must enter from standard input. The *fflush()* calls force the output to standard output. The
11670 *fflush()* function is used because standard output is usually buffered and the prompt may not
11671 immediately be printed on the output or terminal. The *gets()* calls read strings from standard
11672 input and place the results in variables, for use later in the program

```
11673 #include <stdio.h>
11674 ...
11675 char user[100];
11676 char oldpasswd[100];
11677 char newpasswd[100];
11678 ...
11679 printf("User name: ");
11680 fflush(stdout);
11681 gets(user);

11682 printf("Old password: ");
11683 fflush(stdout);
11684 gets(oldpasswd);

11685 printf("New password: ");
11686 fflush(stdout);
11687 gets(newpasswd);
11688 ...
```

11689 **APPLICATION USAGE**

11690 None.

11691 **RATIONALE**

11692 Data buffered by the system may make determining the validity of the position of the current
11693 file descriptor impractical. Thus, enforcing the repositioning of the file descriptor after *fflush()*
11694 on streams open for *read()* is not mandated by IEEE Std. 1003.1-200x.

11695 **FUTURE DIRECTIONS**

11696 None.

11697 **SEE ALSO**

11698 *getrlimit()*, *ulimit()*, the Base Definitions volume of IEEE Std. 1003.1-200x, *<stdio.h>*

11699 **CHANGE HISTORY**

11700 First released in Issue 1. Derived from Issue 1 of the SVID.

11701 **Issue 4**

11702 The following change is incorporated for alignment with the ISO C standard:

- 11703 • The DESCRIPTION is changed to describe the behavior of *fflush()* if *stream* is a null pointer.

11704 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 11705 • The following two paragraphs are withdrawn from the DESCRIPTION (by POSIX as well as
11706 X/Open) because of the possibility of causing applications to malfunction, and the
11707 impossibility of implementing these mechanisms for pipes:

- 11708 If the stream is open for reading, any unread data buffered in the stream is discarded.
- 11709 For a stream open for reading, if the file is not already at EOF, and the file is one capable
11710 of seeking, the file offset of the underlying open file description is adjusted so that the
11711 next operation on the open file description deals with the byte after the last one read
11712 from, or written to, the stream being flushed.
- 11713
 - The [EFBIG] error is marked to indicate the extensions.
- 11714 **Issue 5**
- 11715 Large File Summit extensions are added.
- 11716 **Issue 6**
- 11717 Extensions beyond the ISO C standard are now marked.
- 11718 The following new requirements on POSIX implementations derive from alignment with the
11719 Single UNIX Specification:
- 11720
 - The [EFBIG] error is added as part of the large file support extensions.
- 11721
 - The [ENXIO] optional error condition is added.
- 11722 The RETURN VALUE section is updated to note that the error indicator shall be set for the
11723 stream. This is for alignment with the ISO/IEC 9899:1999 standard. |

11724 **NAME**

11725 ffs — find first set bit

11726 **Notes to Reviewers**11727 *This section with side shading will not appear in the final copy. - Ed.*11728 This function or these functions are recommended to become mandatory parts of POSIX.1 in the
11729 next draft.11730 **SYNOPSIS**11731 xSI `#include <strings.h>`11732 `int ffs(int i);`

11733

11734 **DESCRIPTION**11735 The `ffs()` function shall find the first bit set (beginning with the least significant bit) in *i*, and
11736 return the index of that bit. Bits are numbered starting at one (the least significant bit).11737 **RETURN VALUE**11738 The `ffs()` function shall return the index of the first bit set. If *i* is 0, then `ffs()` shall return 0.11739 **ERRORS**

11740 No errors are defined.

11741 **EXAMPLES**

11742 None.

11743 **APPLICATION USAGE**

11744 None.

11745 **RATIONALE**

11746 None.

11747 **FUTURE DIRECTIONS**

11748 None.

11749 **SEE ALSO**11750 The Base Definitions volume of IEEE Std. 1003.1-200x, `<strings.h>`11751 **CHANGE HISTORY**

11752 First released in Issue 4, Version 2.

11753 **Issue 5**

11754 Moved from X/OPEN UNIX extension to BASE.

11755 NAME

11756 fgetc — get a byte from a stream

11757 SYNOPSIS

11758 #include <stdio.h>

11759 int fgetc(FILE *stream);

11760 DESCRIPTION

11761 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11762 conflict between the requirements described here and the ISO C standard is unintentional. This
 11763 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11764 If the end-of-file indicator for the input stream pointed to by *stream* is not set and a next
 11765 character is present, the *fgetc()* function obtains the next byte (if present) as an **unsigned char**
 11766 converted to an **int**, from the input stream pointed to by *stream*, and advances the associated file
 11767 position indicator for the stream (if defined).

11768 CX The *fgetc()* function may mark the *st_atime* field of the file associated with *stream* for update. The
 11769 *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
 11770 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
 11771 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

11772 RETURN VALUE

11773 Upon successful completion, *fgetc()* shall return the next byte from the input stream pointed to
 11774 by *stream*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the
 11775 end-of-file indicator for the stream shall be set and *fgetc()* shall return EOF. If a read error occurs,
 11776 CX the error indicator for the stream shall be set, *fgetc()* shall return EOF, and shall set *errno* to
 11777 indicate the error.

11778 ERRORS

11779 The *fgetc()* function shall fail if data needs to be read and:

11780 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 11781 process would be delayed in the *fgetc()* operation.

11782 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
 11783 reading.

11784 CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data
 11785 was transferred.

11786 CX [EIO] A physical I/O error has occurred, or the process is in a background process
 11787 group attempting to read from its controlling terminal, and either the process
 11788 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.
 11789 This error may also be generated for implementation-defined reasons.

11790 CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the
 11791 offset maximum associated with the corresponding stream.

11792 The *fgetc()* function may fail if:

11793 CX [ENOMEM] Insufficient storage space is available.

11794 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 11795 capabilities of the device.

11796 **EXAMPLES**

11797 None.

11798 **APPLICATION USAGE**

11799 If the integer value returned by *fgetc()* is stored into a variable of type **char** and then compared
11800 against the integer constant EOF, the comparison may never succeed, because sign-extension of
11801 a variable of type **char** on widening to integer is implementation-defined.

11802 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an
11803 end-of-file condition.

11804 **RATIONALE**

11805 None.

11806 **FUTURE DIRECTIONS**

11807 None.

11808 **SEE ALSO**

11809 *feof()*, *ferror()*, *fopen()*, *getchar()*, *getc()*, the Base Definitions volume of IEEE Std. 1003.1-200x,
11810 <**stdio.h**>

11811 **CHANGE HISTORY**

11812 First released in Issue 1. Derived from Issue 1 of the SVID.

11813 **Issue 4**

11814 In the DESCRIPTION:

- 11815 • The text is changed to make it clear that the function returns a byte value.
- 11816 • The list of functions that may cause the *st_atime* field to be updated is revised.

11817 In the ERRORS section, text is added to indicate that error returns are only generated when data
11818 needs to be read into the stream buffer.

11819 Also in the ERRORS section, in previous issues generation of the [EIO] error depended on
11820 whether or not an implementation supported Job Control. This functionality is now defined as
11821 mandatory.

11822 The [ENXIO] and [ENOMEM] errors are marked as extensions.

11823 In the APPLICATION USAGE section, text is added to indicate how an application can
11824 distinguish between an error condition and an end-of-file condition.

11825 The description of [EINTR] is amended.

11826 **Issue 4, Version 2**

11827 In the ERRORS section, the description of [EIO] is updated to include the case where a physical
11828 I/O error occurs.

11829 **Issue 5**

11830 Large File Summit extensions are added.

11831 **Issue 6**

11832 Extensions beyond the ISO C standard are now marked.

11833 The following new requirements on POSIX implementations derive from alignment with the
11834 Single UNIX Specification:

- 11835 • The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- 11836 • The [ENOMEM] and [ENXIO] optional error conditions are added.

- 11837 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
- 11838 • The DESCRIPTION is updated to clarify the behavior when the end-of-file indicator for the
11839 input stream is not set.
 - 11840 • The RETURN VALUE section is updated to note that the error indicator shall be set for the
11841 stream.

11842 **NAME**

11843 fgetpos — get current file position information

11844 **SYNOPSIS**

11845 #include <stdio.h>

11846 int fgetpos(FILE *restrict stream, fpos_t *restrict pos);

11847 **DESCRIPTION**

11848 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11849 conflict between the requirements described here and the ISO C standard is unintentional. This
11850 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11851 The *fgetpos()* function shall store the current value of the file position indicator for the stream
11852 pointed to by *stream* in the object pointed to by *pos*. The value stored contains unspecified
11853 information usable by *fsetpos()* for repositioning the stream to its position at the time of the call
11854 to *fgetpos()*.

11855 **RETURN VALUE**

11856 Upon successful completion, *fgetpos()* shall return 0; otherwise, it shall return a non-zero value
11857 and set *errno* to indicate the error.

11858 **ERRORS**11859 The *fgetpos()* function shall fail if:

11860 CX [EOVERFLOW] The current value of the file position cannot be represented correctly in an
11861 object of type **fpos_t**.

11862 The *fgetpos()* function may fail if:

11863 CX [EBADF] The file descriptor underlying *stream* is not valid.

11864 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.
11865

11866 **EXAMPLES**

11867 None.

11868 **APPLICATION USAGE**

11869 None.

11870 **RATIONALE**

11871 None.

11872 **FUTURE DIRECTIONS**

11873 None.

11874 **SEE ALSO**

11875 *fopen()*, *ftell()*, *rewind()*, *ungetc()*, the Base Definitions volume of IEEE Std. 1003.1-200x,
11876 <stdio.h>

11877 **CHANGE HISTORY**

11878 First released in Issue 4. Derived from the ISO C standard.

11879 **Issue 5**

11880 Large File Summit extensions are added.

11881 **Issue 6**

11882 Extensions beyond the ISO C standard are now marked.

11883 The following new requirements on POSIX implementations derive from alignment with the
11884 Single UNIX Specification:

- 11885 • The [EIO] mandatory error condition is added.
- 11886 • The [EBADF] and [ESPIPE] optional error conditions are added.
- 11887 An additional [ESPIPE] error condition is added for sockets.
- 11888 The prototype for *fgetpos()* is changed for alignment with the ISO/IEC 9899:1999 standard.

11889 **NAME**

11890 fgets — get a string from a stream

11891 **SYNOPSIS**

11892 #include <stdio.h>

11893 char *fgets(char *restrict *s*, int *n*, FILE *restrict *stream*);11894 **DESCRIPTION**

11895 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 11896 conflict between the requirements described here and the ISO C standard is unintentional. This
 11897 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11898 The *fgets()* function shall read bytes from *stream* into the array pointed to by *s*, until *n*−1 bytes
 11899 are read, or a <newline> character is read and transferred to *s*, or an end-of-file condition is
 11900 encountered. The string is then terminated with a null byte.

11901 cx The *fgets()* function may mark the *st_atime* field of the file associated with *stream* for update. The
 11902 *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
 11903 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
 11904 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

11905 **RETURN VALUE**

11906 Upon successful completion, *fgets()* shall return *s*. If the stream is at end-of-file, the end-of-file
 11907 indicator for the stream shall be set and *fgets()* shall return a null pointer. If a read error occurs,
 11908 cx the error indicator for the stream shall be set, *fgets()* shall return a null pointer, and shall set
 11909 *errno* to indicate the error.

11910 **ERRORS**11911 Refer to *fgetc()*.11912 **EXAMPLES**11913 **Reading Input**

11914 The following example uses *fgets()* to read each line of input. {LINE_MAX}, which defines the
 11915 maximum size of the input line, is defined in the <limits.h> header.

```
11916     #include <stdio.h>
11917     ...
11918     char line[LINE_MAX];
11919     ...
11920     while (fgets(line, LINE_MAX, fp) != NULL) {
11921        ...
11922     }
11923     ...
```

11924 **APPLICATION USAGE**

11925 None.

11926 **RATIONALE**

11927 None.

11928 **FUTURE DIRECTIONS**

11929 None.

11930 **SEE ALSO**

11931 *fopen()*, *fread()*, *gets()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>

11932 **CHANGE HISTORY**

11933 First released in Issue 1. Derived from Issue 1 of the SVID.

11934 **Issue 4**

11935 In the DESCRIPTION, the text is changed to make it clear that the function reads bytes rather
11936 than (possibly multi-byte) characters, and the list of functions that may cause the *st_atime* field to
11937 be updated is revised.

11938 **Issue 6**

11939 Extensions beyond the ISO C standard are now marked.

11940 The prototype for *fgets()* is changed for alignment with the ISO/IEC 9899:1999 standard.

11941 NAME

11942 fgetwc — get a wide-character code from a stream

11943 SYNOPSIS

11944 #include <stdio.h>

11945 #include <wchar.h>

11946 wint_t fgetwc(FILE *stream);

11947 DESCRIPTION

11948 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11949 conflict between the requirements described here and the ISO C standard is unintentional. This
 11950 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

11951 The *fgetwc()* function shall obtain the next character (if present) from the input stream pointed to
 11952 by *stream*, convert that to the corresponding wide-character code, and advance the associated
 11953 file position indicator for the stream (if defined).

11954 If an error occurs, the resulting value of the file position indicator for the stream is
 11955 indeterminate.

11956 CX The *fgetwc()* function may mark the *st_atime* field of the file associated with *stream* for update.
 11957 The *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
 11958 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
 11959 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

11960 RETURN VALUE

11961 Upon successful completion, the *fgetwc()* function shall return the wide-character code of the
 11962 character read from the input stream pointed to by *stream* converted to a type **wint_t**. If the
 11963 stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgetwc()* shall
 11964 return WEOF. If a read error occurs, the error indicator for the stream shall be set, *fgetwc()* shall
 11965 CX return WEOF, and shall set *errno* to indicate the error.

11966 ERRORS

11967 The *fgetwc()* function shall fail if data needs to be read and:

11968 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 11969 process would be delayed in the *fgetwc()* operation.

11970 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
 11971 reading.

11972 CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data
 11973 was transferred.

11974 CX [EIO] A physical I/O error has occurred, or the process is in a background process
 11975 group attempting to read from its controlling terminal, and either the process
 11976 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.
 11977 This error may also be generated for implementation-defined reasons.

11978 CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the
 11979 offset maximum associated with the corresponding stream.

11980 The *fgetwc()* function may fail if:

11981 CX [ENOMEM] Insufficient storage space is available.

11982 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 11983 capabilities of the device.

- 11984 CX [EILSEQ] The data obtained from the input stream does not form a valid character.
- 11985 **EXAMPLES**
- 11986 None.
- 11987 **APPLICATION USAGE**
- 11988 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an
11989 end-of-file condition.
- 11990 **RATIONALE**
- 11991 None.
- 11992 **FUTURE DIRECTIONS**
- 11993 None.
- 11994 **SEE ALSO**
- 11995 *feof()*, *ferror()*, *fopen()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**stdio.h**>,
11996 <**wchar.h**>
- 11997 **CHANGE HISTORY**
- 11998 First released in Issue 4. Derived from the MSE working draft.
- 11999 **Issue 4, Version 2**
- 12000 In the ERRORS section, the description of [EIO] is updated to include the case where a physical
12001 I/O error occurs.
- 12002 **Issue 5**
- 12003 The Optional Header (OH) marking is removed from <**stdio.h**>.
- 12004 Large File Summit extensions are added.
- 12005 **Issue 6**
- 12006 Extensions beyond the ISO C standard are now marked.
- 12007 The following new requirements on POSIX implementations derive from alignment with the
12008 Single UNIX Specification:
- 12009
 - The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- 12010
 - The [ENOMEM], [ENXIO], and [EILSEQ] optional error conditions are added.

12011 NAME

12012 fgetws — get a wide-character string from a stream

12013 SYNOPSIS

12014 #include <stdio.h>

12015 #include <wchar.h>

12016 wchar_t *fgetws(wchar_t *restrict ws, int n,

12017 FILE *restrict stream);

12018 DESCRIPTION

12019 CX The functionality described on this reference page is aligned with the ISO C standard. Any
12020 conflict between the requirements described here and the ISO C standard is unintentional. This
12021 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

12022 The *fgetws()* function shall read characters from the *stream*, convert these to the corresponding
12023 wide-character codes, place them in the **wchar_t** array pointed to by *ws*, until *n*–1 characters are
12024 read, or a <newline> character is read, converted, and transferred to *ws*, or an end-of-file
12025 condition is encountered. The wide-character string, *ws*, is then terminated with a null wide-
12026 character code.

12027 If an error occurs, the resulting value of the file position indicator for the stream is
12028 indeterminate.

12029 CX The *fgetws()* function may mark the *st_atime* field of the file associated with *stream* for update.
12030 The *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
12031 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
12032 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

12033 RETURN VALUE

12034 Upon successful completion, *fgetws()* shall return *ws*. If the stream is at end-of-file, the end-of-
12035 file indicator for the stream shall be set and *fgetws()* shall return a null pointer. If a read error
12036 CX occurs, the error indicator for the stream shall be set, *fgetws()* shall return a null pointer, and
12037 shall set *errno* to indicate the error.

12038 ERRORS

12039 Refer to *fgetwc()*.

12040 EXAMPLES

12041 None.

12042 APPLICATION USAGE

12043 None.

12044 RATIONALE

12045 None.

12046 FUTURE DIRECTIONS

12047 None.

12048 SEE ALSO

12049 *fopen()*, *fread()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>, <wchar.h>

12050 CHANGE HISTORY

12051 First released in Issue 4. Derived from the MSE working draft.

12052 **Issue 5**

12053 The Optional Header (OH) marking is removed from `<stdio.h>`.

12054 **Issue 6**

12055 Extensions beyond the ISO C standard are now marked.

12056 The prototype for `fgetws()` is changed for alignment with the ISO/IEC 9899:1999 standard.

12057 **NAME**

12058 fileno — map a stream pointer to a file descriptor

12059 **SYNOPSIS**

12060 #include <stdio.h>

12061 int fileno(FILE **stream*);

12062 **DESCRIPTION**

12063 The *fileno()* function shall return the integer file descriptor associated with the stream pointed to
12064 by *stream*.

12065 **RETURN VALUE**

12066 Upon successful completion, *fileno()* shall return the integer value of the file descriptor
12067 associated with *stream*. Otherwise, the value -1 shall be returned and *errno* set to indicate the
12068 error.

12069 **ERRORS**

12070 The *fileno()* function may fail if:

12071 [EBADF] The *stream* argument is not a valid stream.

12072 **EXAMPLES**

12073 None.

12074 **APPLICATION USAGE**

12075 None.

12076 **RATIONALE**

12077 Without some specification of which file descriptors are associated with these streams, it is
12078 impossible for an application to set up the streams for another application it starts with *fork()*
12079 and *exec*. In particular, it would not be possible to write a portable version of the *sh* command
12080 interpreter (although there may be other constraints that would prevent that portability).

12081 **FUTURE DIRECTIONS**

12082 None.

12083 **SEE ALSO**

12084 *fdopen()*, *fopen()*, *stdin*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**stdio.h**>, Section
12085 2.5.1 (on page 535)

12086 **CHANGE HISTORY**

12087 First released in Issue 1. Derived from Issue 1 of the SVID.

12088 **Issue 4**

12089 The [EBADF] error is marked as an extension.

12090 **Issue 6**

12091 The following new requirements on POSIX implementations derive from alignment with the
12092 Single UNIX Specification:

- 12093 • The [EBADF] optional error condition is added.

12094 **NAME**

12095 flockfile, ftrylockfile, funlockfile — stdio locking functions

12096 **SYNOPSIS**

```
12097 TSF      #include <stdio.h>
12098          void flockfile(FILE *file);
12099          int ftrylockfile(FILE *file);
12100          void funlockfile(FILE *file);
12101
```

12102 **DESCRIPTION**

12103 The *flockfile()*, *ftrylockfile()*, and *funlockfile()* functions provide for explicit application-level
 12104 locking of stdio (**FILE***) objects. These functions can be used by a thread to delineate a sequence
 12105 of I/O statements that are executed as a unit.

12106 The *flockfile()* function is used by a thread to acquire ownership of a (**FILE***) object.

12107 The *ftrylockfile()* function is used by a thread to acquire ownership of a (**FILE***) object if the
 12108 object is available; *ftrylockfile()* is a non-blocking version of *flockfile()*.

12109 The *funlockfile()* function is used to relinquish the ownership granted to the thread. The
 12110 behavior is undefined if a thread other than the current owner calls the *funlockfile()* function.

12111 Logically, there is a lock count associated with each (**FILE***) object. This count is implicitly
 12112 initialized to zero when the (**FILE***) object is created. The (**FILE***) object is unlocked when the
 12113 count is zero. When the count is positive, a single thread owns the (**FILE***) object. When the
 12114 *flockfile()* function is called, if the count is zero or if the count is positive and the caller owns the
 12115 (**FILE***) object, the count is incremented. Otherwise, the calling thread is suspended, waiting for
 12116 the count to return to zero. Each call to *funlockfile()* decrements the count. This allows matching
 12117 calls to *flockfile()* (or successful calls to *ftrylockfile()*) and *funlockfile()* to be nested.

12118 All functions that reference (**FILE***) objects shall behave as if they use *flockfile()* and *funlockfile()*
 12119 internally to obtain ownership of these (**FILE***) objects.

12120 **RETURN VALUE**

12121 None for *flockfile()* and *funlockfile()*. The *ftrylockfile()* function shall return zero for success and
 12122 non-zero to indicate that the lock cannot be acquired.

12123 **ERRORS**

12124 No errors are defined.

12125 **EXAMPLES**

12126 None.

12127 **APPLICATION USAGE**

12128 Applications using these functions may be subject to priority inversion, as discussed in the Base
 12129 Definitions volume of IEEE Std. 1003.1-200x, Section 3.287, Priority Inversion.

12130 **RATIONALE**

12131 The *flockfile()* and *funlockfile()* functions provide an orthogonal mutual exclusion lock for each
 12132 **FILE**. The *ftrylockfile()* function provides a non-blocking attempt to acquire a file lock,
 12133 analogous to *pthread_mutex_trylock()*.

12134 These locks behave as if they are the same as those used internally by *stdio* for thread-safety.
 12135 This both provides thread-safety of these functions without requiring a second level of internal
 12136 locking and allows functions in *stdio* to be implemented in terms of other *stdio* functions.

12137 Application writers and implementors should be aware that there are potential deadlock
 12138 problems on **FILE** objects. For example, the line-buffered flushing semantics of *stdio* (requested

12139 via {_IOLBF}) require that certain input operations sometimes cause the buffered contents of
12140 implementation-defined line-buffered output streams to be flushed. If two threads each hold the
12141 lock on the other's **FILE**, deadlock ensues. This type of deadlock can be avoided by acquiring
12142 **FILE** locks in a consistent order. In particular, the line-buffered output stream deadlock can
12143 typically be avoided by acquiring locks on input streams before locks on output streams if a
12144 thread would be acquiring both.

12145 In summary, threads sharing *stdio* streams with other threads can use *flockfile()* and *funlockfile()*
12146 to cause sequences of I/O performed by a single thread to be kept bundled. The only case where
12147 the use of *flockfile()* and *funlockfile()* is required is to provide a scope protecting uses of the
12148 *_unlocked() functions/macros. This moves the cost/performance tradeoff to the optimal point.

12149 **FUTURE DIRECTIONS**

12150 None.

12151 **SEE ALSO**

12152 *getc_unlocked()*, *putc_unlocked()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**stdio.h**>

12153 **CHANGE HISTORY**

12154 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

12155 **Issue 6**

12156 These functions are marked as part of the Thread-Safe Functions option.

12157 **NAME**

12158 floor, floorf, floorl — floor function

12159 **SYNOPSIS**

12160 #include <math.h>

12161 double floor(double x);

12162 float floorf(float x);

12163 long double floorl(long double x);

12164 **DESCRIPTION**

12165 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 12166 conflict between the requirements described here and the ISO C standard is unintentional. This
 12167 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

12168 These functions shall compute the largest integral value not greater than *x*.

12169 An application wishing to check for error situations should set *errno* to 0 before calling *floor()*. If
 12170 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

12171 **RETURN VALUE**

12172 Upon successful completion, these functions shall return the largest integral value not greater
 12173 than *x*, expressed as a **double**.

12174 **XSI** If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

12175 If the correct value would cause overflow, -HUGE_VAL shall be returned and *errno* set to
 12176 [ERANGE].

12177 **XSI** If *x* is ±Inf or ±0, the value of *x* shall be returned.12178 **ERRORS**

12179 These functions shall fail if:

12180 [ERANGE] The result would cause an overflow.

12181 These functions may fail if:

12182 **XSI** [EDOM] The value of *x* is NaN.12183 **XSI** No other errors shall occur.12184 **EXAMPLES**

12185 None.

12186 **APPLICATION USAGE**

12187 The integral value returned by *floor()* as a **double** might not be expressible as an **int** or **long**. The
 12188 return value should be tested before assigning it to an integer type to avoid the undefined results
 12189 of an integer overflow.

12190 The *floor()* function can only overflow when the floating point representation has
 12191 DBL_MANT_DIG > DBL_MAX_EXP.

12192 **RATIONALE**

12193 None.

12194 **FUTURE DIRECTIONS**

12195 None.

12196 **SEE ALSO**

12197 *ceil()*, *isnan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

12198 **CHANGE HISTORY**

12199 First released in Issue 1. Derived from Issue 1 of the SVID.

12200 **Issue 4**

12201 References to *matherr()* are removed.

12202 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the
12203 ISO C standard and to rationalize handling in the mathematics functions.

12204 The word **long** has been replaced with the words **long** in the APPLICATION USAGE section.

12205 The return value specified for [EDOM] is marked as an extension.

12206 **Issue 5**

12207 The DESCRIPTION is updated to indicate how an application should check for an error. This
12208 text was previously published in the APPLICATION USAGE section.

12209 **Issue 6**

12210 The *floorf()* and *floorl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

12211 **NAME**

12212 fma, fmaf, fmal — floating-point multiply-add

12213 **SYNOPSIS**

12214 #include <math.h>

12215 double fma(double x, double y, double z);

12216 float fmaf(float x, float y, float z);

12217 long double fmal(long double x, long double y, long double z);

12218 **DESCRIPTION**

12219 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
12220 conflict between the requirements described here and the ISO C standard is unintentional. This
12221 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

12222 These functions shall compute $(x * y) + z$, rounded as one ternary operation: they shall compute
12223 the value (as if) to infinite precision and round once to the result format, according to the
12224 rounding mode characterized by the value of FLT_ROUNDS.

12225 An application wishing to check for error situations should set *errno* to 0 before calling these
12226 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

12227 **RETURN VALUE**

12228 Upon successful completion, these functions shall return $(x * y) + z$, rounded as one ternary
12229 operation.

12230 If *x*, *y*, or *z* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

12231 **ERRORS**

12232 These functions may fail if:

12233 [EDOM] The value of *x*, *y*, or *z* is NaN.

12234 **EXAMPLES**

12235 None.

12236 **APPLICATION USAGE**

12237 None.

12238 **RATIONALE**

12239 In many cases, clever use of floating (*fused*) multiply-add leads to much improved code; but its
12240 unexpected use by the compiler can undermine carefully written code. The FP_CONTRACT
12241 macro can be used to disallow use of floating multiply-add; and the *fma()* function guarantees
12242 its use where desired. Many current machines provide hardware floating multiply-add
12243 instructions; software implementation can be used for others.

12244 **FUTURE DIRECTIONS**

12245 None.

12246 **SEE ALSO**

12247 The Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

12248 **CHANGE HISTORY**

12249 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

12250 **NAME**

12251 fmax, fmaxf, fmaxl — determine maximum numeric value of two floating-point numbers

12252 **SYNOPSIS**

12253 #include <math.h>

12254 double fmax(double x, double y);

12255 float fmaxf(float x, float y);

12256 long double fmaxl(long double x, long double y);

12257 **DESCRIPTION**

12258 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
12259 conflict between the requirements described here and the ISO C standard is unintentional. This
12260 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

12261 These functions shall determine the maximum numeric value of their arguments. NaN
12262 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,
12263 then the *fmax()*, *fmaxf()*, and *fmaxl()* functions shall choose the numeric value.

12264 An application wishing to check for error situations should set *errno* to 0 before calling these
12265 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

12266 **RETURN VALUE**

12267 Upon successful completion, these functions shall return the maximum numeric value of their
12268 arguments.

12269 If *x* and *y* are NaN, NaN shall be returned and *errno* may be set to [EDOM].

12270 **ERRORS**

12271 These functions may fail if:

12272 [EDOM] The value of *x* and *y* is NaN.

12273 **EXAMPLES**

12274 None.

12275 **APPLICATION USAGE**

12276 None.

12277 **RATIONALE**

12278 None.

12279 **FUTURE DIRECTIONS**

12280 None.

12281 **SEE ALSO**

12282 *fdim()*, *fmin()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

12283 **CHANGE HISTORY**

12284 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

12285 **NAME**

12286 `fmin`, `fminf`, `fminl` — determine minimum numeric value of two floating-point numbers

12287 **SYNOPSIS**

12288 `#include <math.h>`

12289 `double fmin(double x, double y);`

12290 `float fminf(float x, float y);`

12291 `long double fminl(long double x, long double y);`

12292 **DESCRIPTION**

12293 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
12294 conflict between the requirements described here and the ISO C standard is unintentional. This
12295 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

12296 These functions shall determine the minimum numeric value of their arguments. NaN
12297 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,
12298 then these functions shall choose the numeric value.

12299 An application wishing to check for error situations should set *errno* to 0 before calling these
12300 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

12301 **RETURN VALUE**

12302 Upon successful completion, these functions shall return the minimum numeric value of their
12303 arguments.

12304 If *x* and *y* are NaN, NaN shall be returned and *errno* may be set to [EDOM].

12305 **ERRORS**

12306 These functions may fail if:

12307 [EDOM] The value of *x* and *y* is NaN.

12308 **EXAMPLES**

12309 None.

12310 **APPLICATION USAGE**

12311 None.

12312 **RATIONALE**

12313 None.

12314 **FUTURE DIRECTIONS**

12315 None.

12316 **SEE ALSO**

12317 *fdim()*, *fmax()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<math.h>`

12318 **CHANGE HISTORY**

12319 First released in Issue 6. Derived from ISO/IEC 9899:1999 standard.

12320 **NAME**

12321 fmod, fmodf, fmodl — floating-point remainder value function

12322 **SYNOPSIS**

12323 #include <math.h>

12324 double fmod(double x, double y);

12325 float fmodf(float x, float y);

12326 long double fmodl(long double x, long double y);

12327 **DESCRIPTION**

12328 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 12329 conflict between the requirements described here and the ISO C standard is unintentional. This
 12330 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

12331 These functions shall return the floating-point remainder of the division of x by y .

12332 An application wishing to check for error situations should set *errno* to 0 before calling *fmod()*. If
 12333 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

12334 **RETURN VALUE**

12335 These functions shall return the value $x-i*y$, for some integer i such that, if y is non-zero, the
 12336 result has the same sign as x and magnitude less than the magnitude of y .

12337 **XSI** If x or y is NaN, NaN shall be returned and *errno* may be set to [EDOM].

12338 **XSI** If y is 0, NaN shall be returned and *errno* set to [EDOM], or 0 shall be returned and *errno* may be
 12339 set to [EDOM].

12340 **XSI** If x is $\pm\text{Inf}$, either 0 shall be returned and *errno* set to [EDOM], or NaN shall be returned and *errno*
 12341 may be set to [EDOM].

12342 If y is non-zero, *fmod*($\pm 0, y$) shall return the value of x . If x is not $\pm\text{Inf}$, *fmod*($x, \pm\text{Inf}$) shall return
 12343 the value of x .

12344 If the result underflows, 0 shall be returned and *errno* may be set to [ERANGE].12345 **ERRORS**

12346 These functions may fail if:

12347 **XSI** [EDOM] One or both of the arguments is NaN, or y is 0, or x is $\pm\text{Inf}$.

12348 [ERANGE] The result underflows

12349 **XSI** No other errors shall occur.12350 **EXAMPLES**

12351 None.

12352 **APPLICATION USAGE**

12353 Portable applications should not call *fmod()* with y equal to 0, because the result is
 12354 implementation-defined. The application should verify y is non-zero before calling *fmod()*.

12355 **RATIONALE**

12356 None.

12357 **FUTURE DIRECTIONS**

12358 None.

12359 **SEE ALSO**

12360 *isnan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

12361 **CHANGE HISTORY**

12362 First released in Issue 1. Derived from Issue 1 of the SVID.

12363 **Issue 4**

12364 References to *matherr()* are removed.

12365 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the
12366 ISO C standard and to rationalize error handling in the mathematics functions.

12367 The return value specified for [EDOM] is marked as an extension.

12368 **Issue 5**

12369 The DESCRIPTION is updated to indicate how an application should check for an error. This
12370 text was previously published in the APPLICATION USAGE section.

12371 **Issue 6**

12372 The *fmodf()* and *fmodl()* functions are added for alignment with the ISO/IEC 9899:1999
12373 standard.

12374 **NAME**

12375 fmtmsg — display a message in the specified format on standard error and/or a system console

12376 **SYNOPSIS**

```
12377 xSI       #include <fmtmsg.h>
12378           int fmtmsg(long classification, const char *label, int severity,
12379                    const char *text, const char *action, const char *tag);
12380
```

12381 **DESCRIPTION**12382 The *fmtmsg()* function can be used to display messages in a specified format instead of the
12383 traditional *printf()* function.12384 Based on a message's classification component, *fmtmsg()* writes a formatted message either to
12385 standard error, to the console, or to both.12386 A formatted message consists of up to five components as defined below. The component
12387 *classification* is not part of a message displayed to the user, but defines the source of the message
12388 and directs the display of the formatted message.

12389 *classification* Contains identifiers from the following groups of major classifications and
12390 subclassifications. Any one identifier from a subclass may be used in
12391 combination with a single identifier from a different subclass. Two or more
12392 identifiers from the same subclass should not be used together, with the
12393 exception of identifiers from the display subclass. (Both display subclass
12394 identifiers may be used so that messages can be displayed to both standard
12395 error and the system console).

12396 **Major Classifications**12397 Identifies the source of the condition. Identifiers are: MM_HARD
12398 (hardware), MM_SOFT (software), and MM_FIRM (firmware).12399 **Message Source Subclassifications**12400 Identifies the type of software in which the problem is detected.
12401 Identifiers are: MM_APPL (application), MM_UTIL (utility), and
12402 MM_OPYSYS (operating system).12403 **Display Subclassifications**12404 Indicates where the message is to be displayed. Identifiers are:
12405 MM_PRINT to display the message on the standard error stream,
12406 MM_CONSOLE to display the message on the system console. One or
12407 both identifiers may be used.12408 **Status Subclassifications**12409 Indicates whether the application can recover from the condition.
12410 Identifiers are: MM_RECOVER (recoverable) and MM_NRECOV (non-
12411 recoverable).12412 An additional identifier, MM_NULLMC, indicates that no classification
12413 component is supplied for the message.12414 *label* Identifies the source of the message. The format is two fields separated by a
12415 colon. The first field is up to 10 bytes, the second is up to 14 bytes.12416 *severity* Indicates the seriousness of the condition. Identifiers for the levels of *severity*
12417 are:

12418		MM_HALT	Indicates that the application has encountered a severe fault and is halting. Produces the string "HALT".
12419			
12420		MM_ERROR	Indicates that the application has detected a fault. Produces the string "ERROR".
12421			
12422		MM_WARNING	Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the string "WARNING".
12423			
12424			
12425		MM_INFO	Provides information about a condition that is not in error. Produces the string "INFO".
12426			
12427		MM_NOSEV	Indicates that no severity level is supplied for the message.
12428	<i>text</i>		Describes the error condition that produced the message. The character string is not limited to a specific size. If the character string is empty, then the text produced is unspecified.
12429			
12430			
12431	<i>action</i>		Describes the first step to be taken in the error-recovery process. The <i>fmtmsg()</i> function precedes the action string with the prefix: "TO FIX:". The <i>action</i> string is not limited to a specific size.
12432			
12433			
12434	<i>tag</i>		An identifier that references on-line documentation for the message. Suggested usage is that <i>tag</i> includes the <i>label</i> and a unique identifying number. A sample <i>tag</i> is "XSI:cat:146".
12435			
12436			
12437			The <i>MSGVERB</i> environment variable (for message verbosity) tells <i>fmtmsg()</i> which message components it is to select when writing messages to standard error. The value of <i>MSGVERB</i> is a colon-separated list of optional keywords. Valid <i>keywords</i> are: <i>label</i> , <i>severity</i> , <i>text</i> , <i>action</i> , and <i>tag</i> . If <i>MSGVERB</i> contains a keyword for a component and the component's value is not the component's null value, <i>fmtmsg()</i> includes that component in the message when writing the message to standard error. If <i>MSGVERB</i> does not include a keyword for a message component, that component is not included in the display of the message. The keywords may appear in any order. If <i>MSGVERB</i> is not defined, if its value is the null string, if its value is not of the correct format, or if it contains keywords other than the valid ones listed above, <i>fmtmsg()</i> selects all components.
12438			
12439			
12440			
12441			
12442			
12443			
12444			
12445			
12446			
12447			<i>MSGVERB</i> affects only which components are selected for display to standard error. All message components are included in console messages.
12448			
12449	RETURN VALUE		
12450			The <i>fmtmsg()</i> function shall return one of the following values:
12451	MM_OK		The function succeeded.
12452	MM_NOTOK		The function failed completely.
12453	MM_NOMSG		The function was unable to generate a message on standard error, but otherwise succeeded.
12454			
12455	MM_NOCON		The function was unable to generate a console message, but otherwise succeeded.
12456			
12457	ERRORS		
12458			None.

12459 **EXAMPLES**

12460 1. The following example of *fmtmsg()*:

```
12461     fmtmsg(MM_PRINT, "XSI:cat", MM_ERROR, "illegal option",  
12462     "refer to cat in user's reference manual", "XSI:cat:001")
```

12463 produces a complete message in the specified message format:

```
12464     XSI:cat: ERROR: illegal option  
12465     TO FIX: refer to cat in user's reference manual XSI:cat:001
```

12466 2. When the environment variable *MSGVERB* is set as follows:

```
12467     MSGVERB=severity:text:action
```

12468 and Example 1 is used, *fmtmsg()* produces:

```
12469     ERROR: illegal option  
12470     TO FIX: refer to cat in user's reference manual
```

12471 **APPLICATION USAGE**

12472 One or more message components may be systematically omitted from messages generated by
12473 an application by using the null value of the argument for that component.

12474 **RATIONALE**

12475 None.

12476 **FUTURE DIRECTIONS**

12477 None.

12478 **SEE ALSO**

12479 *printf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <*fmtmsg.h*>

12480 **CHANGE HISTORY**

12481 First released in Issue 4, Version 2.

12482 **Issue 5**

12483 Moved from X/OPEN UNIX extension to BASE.

12484 **NAME**

12485 fnmatch — match a file name or a path name

12486 **SYNOPSIS**

12487 #include <fnmatch.h>

12488 int fnmatch(const char **pattern*, const char **string*, int *flags*);12489 **DESCRIPTION**

12490 The *fnmatch()* function shall match patterns as described in the Shell and Utilities volume of
 12491 IEEE Std. 1003.1-200x, Section 2.14.1, Patterns Matching a Single Character, and Section 2.14.2,
 12492 Patterns Matching Multiple Characters. It checks the string specified by the *string* argument to
 12493 see if it matches the pattern specified by the *pattern* argument.

12494 The *flags* argument modifies the interpretation of *pattern* and *string*. It is the bitwise-inclusive OR
 12495 of zero or more of the flags defined in <fnmatch.h>. If the FNM_PATHNAME flag is set in *flags*,
 12496 then a slash character ('/') in *string* shall be explicitly matched by a slash in *pattern*; it shall not
 12497 be matched by either the asterisk or question-mark special characters, nor by a bracket
 12498 expression. If the FNM_PATHNAME flag is not set, the slash character is treated as an ordinary
 12499 character.

12500 If FNM_NOESCAPE is not set in *flags*, a backslash character ('\') in *pattern* followed by any
 12501 other character shall match that second character in *string*. In particular, "\\\" shall match a
 12502 backslash in *string*. If FNM_NOESCAPE is set, a backslash character shall be treated as an
 12503 ordinary character.

12504 If FNM_PERIOD is set in *flags*, then a leading period ('.') in *string* shall match a period in
 12505 *pattern*; as described by rule 2 in the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section
 12506 2.14.3, Patterns Used for File Name Expansion where the location of “leading” is indicated by
 12507 the value of FNM_PATHNAME:

- 12508 • If FNM_PATHNAME is set, a period is “leading” if it is the first character in *string* or if it
 12509 immediately follows a slash.
- 12510 • If FNM_PATHNAME is not set, a period is “leading” only if it is the first character of *string*.

12511 If FNM_PERIOD is not set, then no special restrictions are placed on matching a period.

12512 **RETURN VALUE**

12513 If *string* matches the pattern specified by *pattern*, then *fnmatch()* shall return 0. If there is no
 12514 match, *fnmatch()* shall return FNM_NOMATCH, which is defined in <fnmatch.h>. If an error
 12515 occurs, *fnmatch()* shall return another non-zero value.

12516 **ERRORS**

12517 No errors are defined.

12518 **EXAMPLES**

12519 None.

12520 **APPLICATION USAGE**

12521 The *fnmatch()* function has two major uses. It could be used by an application or utility that
 12522 needs to read a directory and apply a pattern against each entry. The *find* utility is an example of
 12523 this. It can also be used by the *pax* utility to process its *pattern* operands, or by applications that
 12524 need to match strings in a similar manner.

12525 The name *fnmatch()* is intended to imply *file name* match, rather than *path name* match. The
 12526 default action of this function is to match file names, rather than path names, since it gives no
 12527 special significance to the slash character. With the FNM_PATHNAME flag, *fnmatch()* does
 12528 match path names, but without tilde expansion, parameter expansion, or special treatment for a

12529 period at the beginning of a file name.

12530 **RATIONALE**

12531 This function replaced the REG_FILENAME flag of *regcomp()* in early proposals of this volume
12532 of IEEE Std. 1003.1-200x. It provides virtually the same functionality as the *regcomp()* and
12533 *regexec()* functions using the REG_FILENAME and REG_FSLASH flags (the REG_FSLASH flag
12534 was proposed for *regcomp()*, and would have had the opposite effect from FNM_PATHNAME),
12535 but with a simpler function and less system overhead.

12536 **FUTURE DIRECTIONS**

12537 None.

12538 **SEE ALSO**

12539 *glob()*, *wordexp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <fnmatch.h>, the Shell
12540 and Utilities volume of IEEE Std. 1003.1-200x

12541 **CHANGE HISTORY**

12542 First released in Issue 4. Derived from the ISO POSIX-2 standard.

12543 **Issue 5**

12544 Moved from POSIX2 C-language Binding to BASE.

12545 NAME

12546 fopen — open a stream

12547 SYNOPSIS

12548 #include <stdio.h>

12549 FILE *fopen(const char *restrict filename, const char *restrict mode);

12550 DESCRIPTION

12551 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 12552 conflict between the requirements described here and the ISO C standard is unintentional. This
 12553 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

12554 The *fopen()* function shall open the file whose path name is the string pointed to by *filename*, and
 12555 associates a stream with it.

12556 The argument *mode* points to a string. If the string is one of the following, the file is open in the
 12557 indicated mode. Otherwise, the behavior is undefined.

12558	<i>r</i> or <i>rb</i>	Open file for reading.
12559	<i>w</i> or <i>wb</i>	Truncate to zero length or create file for writing.
12560	<i>a</i> or <i>ab</i>	Append; open or create file for writing at end-of-file.
12561	<i>r+</i> or <i>rb+</i> or <i>r+b</i>	Open file for update (reading and writing).
12562	<i>w+</i> or <i>wb+</i> or <i>w+b</i>	Truncate to zero length or create file for update.
12563	<i>a+</i> or <i>ab+</i> or <i>a+b</i>	Append; open or create file for update, writing at end-of-file.

12564 CX The character 'b' has no effect, but is allowed for ISO C standard conformance. Opening a file
 12565 with read mode (*r* as the first character in the *mode* argument) shall fail if the file does not exist or
 12566 cannot be read.

12567 Opening a file with append mode (*a* as the first character in the *mode* argument) shall cause all
 12568 subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening
 12569 calls to *fseek()*.

12570 When a file is opened with update mode ('+' as the second or third character in the *mode*
 12571 argument), both input and output may be performed on the associated stream. However, the
 12572 application shall ensure that output is not directly followed by input without an intervening call
 12573 to *fflush()* or to a file positioning function (*fseek()*, *fsetpos()*, or *rewind()*), and input is not directly
 12574 followed by output without an intervening call to a file positioning function, unless the input
 12575 operation encounters end-of-file.

12576 When opened, a stream is fully buffered if and only if it can be determined not to refer to an
 12577 interactive device. The error and end-of-file indicators for the stream shall be cleared.

12578 CX If *mode* is *w*, *a*, *w+*, or *a+*, and the file did not previously exist, upon successful completion,
 12579 *fopen()* function shall mark for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the file and
 12580 the *st_ctime* and *st_mtime* fields of the parent directory.

12581 If *mode* is *w* or *w+* and the file did previously exist, upon successful completion, *fopen()* shall
 12582 mark for update the *st_ctime* and *st_mtime* fields of the file. The *fopen()* function shall allocate a
 12583 file descriptor as *open()* does.

12584 XSI After a successful call to the *fopen()* function, the orientation of the stream shall be cleared, the
 12585 encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an
 12586 initial conversion state.

12587 The largest value that can be represented correctly in an object of type `off_t` shall be established
 12588 as the offset maximum in the open file description.

12589 RETURN VALUE

12590 Upon successful completion, `fopen()` shall return a pointer to the object controlling the stream.
 12591 CX Otherwise, a null pointer shall be returned, and `errno` shall be set to indicate the error.

12592 ERRORS

12593 The `fopen()` function shall fail if:

12594 CX [EACCES] Search permission is denied on a component of the path prefix, or the file
 12595 exists and the permissions specified by `mode` are denied, or the file does not
 12596 exist and write permission is denied for the parent directory of the file to be
 12597 created.

12598 CX [EINTR] A signal was caught during `fopen()`.

12599 CX [EISDIR] The named file is a directory and `mode` requires write access.

12600 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`
 12601 argument.

12602 CX [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

12603 CX [ENAMETOOLONG]

12604 The length of the `filename` argument exceeds {PATH_MAX} or a path name
 12605 component is longer than {NAME_MAX}.

12606 CX [ENFILE] The maximum allowable number of files is currently open in the system.

12607 CX [ENOENT] A component of `filename` does not name an existing file or `filename` is an empty
 12608 string.

12609 CX [ENOSPC] The directory or file system that would contain the new file cannot be
 12610 expanded, the file does not exist, and it was to be created.

12611 CX [ENOTDIR] A component of the path prefix is not a directory.

12612 CX [ENXIO] The named file is a character special or block special file, and the device
 12613 associated with this special file does not exist.

12614 CX [EOVERFLOW] The named file is a regular file and the size of the file cannot be represented
 12615 correctly in an object of type `off_t`.

12616 CX [EROFS] The named file resides on a read-only file system and `mode` requires write
 12617 access.

12618 The `fopen()` function may fail if:

12619 CX [EINVAL] The value of the `mode` argument is not valid.

12620 CX [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 12621 resolution of the `path` argument.

12622 CX [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

12623 CX [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

12624 CX [ENAMETOOLONG]

12625 Path name resolution of a symbolic link produced an intermediate result
 12626 whose length exceeds {PATH_MAX}.

12627	CX	[ENOMEM]	Insufficient storage space is available.
12628	CX	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>mode</i>
12629			requires write access.

12630 **EXAMPLES**12631 **Opening a File**

12632 The following example tries to open the file named **file** for reading. The *fopen()* function returns
 12633 a file pointer that is used in subsequent *fgets()* and *fclose()* calls. If the program cannot open the
 12634 file, it just ignores it.

```

12635 #include <stdio.h>
12636 ...
12637 FILE *fp;
12638 ...
12639 void rgrep(const char *file)
12640 {
12641     ...
12642     if ((fp = fopen(file, "r")) == NULL)
12643         return;
12644     ...
12645 }
```

12646 **APPLICATION USAGE**

12647 None.

12648 **RATIONALE**

12649 None.

12650 **FUTURE DIRECTIONS**

12651 None.

12652 **SEE ALSO**

12653 *fclose()*, *fdopen()*, *freopen()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<stdio.h>**

12654 **CHANGE HISTORY**

12655 First released in Issue 1. Derived from Issue 1 of the SVID.

12656 **Issue 4**

12657 In the DESCRIPTION, the descriptions of input and output operations on update streams are
 12658 changed to be requirements on the application.

12659 The [EMFILE] error is added to the ERRORS section, and all the optional errors are marked as
 12660 extensions.

12661 The following changes are incorporated for alignment with the ISO C standard:

- 12662 • The type of arguments *filename* and *mode* are changed from **char*** to **const char***.
- 12663 • In the DESCRIPTION, the use and settings of the *mode* argument are changed to support
 12664 binary streams, and *setpos()* is added to the list of file positioning functions.

12665 The following change is incorporated for alignment with the FIPS requirements:

- 12666 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path
 12667 name component is larger than {NAME_MAX} is now defined as mandatory and marked as
 12668 an extension.

12669 **Issue 4, Version 2**

12670 The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- 12671 • It states that [ELOOP] is returned if too many symbolic links are encountered during path
12672 name resolution.
- 12673 • A second [ENAMETOOLONG] condition is defined that may report excessive length of an
12674 intermediate result of path name resolution of a symbolic link.

12675 **Issue 5**

12676 Large File Summit extensions are added.

12677 **Issue 6**

12678 Extensions beyond the ISO C standard are now marked.

12679 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 12680 • The [ENAMETOOLONG] error is restored as an error dependent on _POSIX_NO_TRUNC.
12681 This is since behavior may vary from one file system to another.

12682 The following new requirements on POSIX implementations derive from alignment with the
12683 Single UNIX Specification:

- 12684 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file
12685 description. This change is to support large files.
- 12686 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support
12687 large files.
- 12688 • The [ELOOP] mandatory error condition is added.
- 12689 • The [EINVAL], [EMFILE], [ENAMETOOLONG], [ENOMEM], and [ETXTBSY] optional error
12690 conditions are added.

12691 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

12692 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- 12693 • The prototype for *fopen()* is updated.
- 12694 • The DESCRIPTION is updated to note that if the argument *mode* points to a string other than
12695 those listed, then the behavior is undefined.

12696 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
12697 [ELOOP] error condition is added.

12698 NAME

12699 fork — create a new process

12700 SYNOPSIS

12701 #include <unistd.h>

12702 pid_t fork(void);

12703 DESCRIPTION

12704 The *fork()* function creates a new process. The new process (child process) shall be an exact copy
 12705 of the calling process (parent process) except as detailed below:

- 12706 • The child process has a unique process ID.
- 12707 • The child process ID also does not match any active process group ID.
- 12708 • The child process has a different parent process ID (that is, the process ID of the parent
 12709 process).
- 12710 • The child process has its own copy of the parent's file descriptors. Each of the child's file
 12711 descriptors refers to the same open file description with the corresponding file descriptor of
 12712 the parent.
- 12713 • The child process has its own copy of the parent's open directory streams. Each open
 12714 directory stream in the child process may share directory stream positioning with the
 12715 corresponding directory stream of the parent.
- 12716 XSI • The child process may have its own copy of the parent's message catalog descriptors.
- 12717 • The child process' values of *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* are set to 0.
- 12718 • The time left until an alarm clock signal is reset to zero, and the alarm, if any, is canceled; see
 12719 *alarm()*.
- 12720 XSI • All *semadj* values are cleared.
- 12721 • File locks set by the parent process are not inherited by the child process.
- 12722 • The set of signals pending for the child process is initialized to the empty set.
- 12723 XSI • Interval timers are reset in the child process.
- 12724 SEM • Any semaphores that are open in the parent process shall also be open in the child process.
- 12725 ML • The child process does not inherit any address space memory locks established by the parent
 12726 process via calls to *mlockall()* or *mlock()*.
- 12727 MF|SHM • Memory mappings created in the parent are retained in the child process. MAP_PRIVATE
 12728 mappings inherited from the parent shall also be MAP_PRIVATE mappings in the child, and
 12729 any modifications to the data in these mappings made by the parent prior to calling *fork()*
 12730 shall be visible to the child. Any modifications to the data in MAP_PRIVATE mappings made
 12731 by the parent after *fork()* returns shall be visible only to the parent. Modifications to the data
 12732 in MAP_PRIVATE mappings made by the child shall be visible only to the child.
- 12733 PS • For the SCHED_FIFO and SCHED_RR scheduling policies, the child process shall inherit the
 12734 policy and priority settings of the parent process during a *fork()* function. For other
 12735 scheduling policies, the policy and priority settings on *fork()* are implementation-defined.
- 12736 TMR • Per-process timers created by the parent are not inherited by the child process.
- 12737 MSG • The child process has its own copy of the message queue descriptors of the parent. Each of
 12738 the message descriptors of the child refers to the same open message queue description as
 12739 the corresponding message descriptor of the parent.

- 12740 AIO
12741
- No asynchronous input or asynchronous output operations are inherited by the child process.
- 12742
- A process is created with a single thread. If a multi-threaded process calls *fork()*, the new process contains a replica of the calling thread and its entire address space, possibly including the states of mutexes and other resources. Consequently, to avoid errors, the child process may only execute async-signal-safe operations until such time as one of the *exec* functions is called. Fork handlers may be established by means of the *pthread_atfork()* function in order to maintain application invariants across *fork()* calls.
- 12743
12744
12745
12746 THR
12747
- 12748 TRC TRI
- If the Trace option and the Trace Inherit option are both supported:
- 12749 If the calling process was being traced in a trace stream that had its inheritance policy set to
12750 POSIX_TRACE_INHERITED, the child process shall be traced into that trace stream, and the
12751 child process shall inherit the parent's mapping of trace event names to trace event type
12752 identifiers. If the trace stream in which the calling process was being traced had its
12753 inheritance policy set to POSIX_TRACE_CLOSE_FOR_CHILD, the child process shall not be
12754 traced into that trace stream. The inheritance policy is set by a call to the
12755 *posix_trace_attr_setinherited()* function.
- 12756 TRC
- If the Trace option is supported, but the Trace Inherit option is not supported:
- 12757 The child process shall not be traced into any of the trace streams of its parent process.
- 12758
- If the Trace option is supported, the child process of a trace controller process shall not
12759 control the trace streams controlled by its parent process.
- 12760 CPT The initial value of the CPU-time clock of the child process shall be set to zero.
- 12761 TCT The initial value of the CPU-time clock of the single thread of the child process shall be set to
12762 zero.

12763 **Notes to Reviewers**

- 12764 *This section with side shading will not appear in the final copy. - Ed.*
- 12765 Check this text is correct after new addenda are rolled in. (Ref D1, XSH, ERN 126)
- 12766 For process characteristics not defined by this volume of IEEE Std. 1003.1-200x, their inheritance
12767 is not defined by this volume of IEEE Std. 1003.1-200x. Process characteristics defined by this
12768 volume of IEEE Std. 1003.1-200x have their inheritance explicitly defined.
- 12769 After *fork()*, both the parent and the child processes are capable of executing independently
12770 before either one terminates.

12771 **RETURN VALUE**

- 12772 Upon successful completion, *fork()* shall return 0 to the child process and shall return the
12773 process ID of the child process to the parent process. Both processes shall continue to execute
12774 from the *fork()* function. Otherwise, -1 shall be returned to the parent process, no child process
12775 shall be created, and *errno* shall be set to indicate the error.

12776 **ERRORS**

- 12777 The *fork()* function shall fail if:
- 12778 [EAGAIN] The system lacked the necessary resources to create another process, or the
12779 system-imposed limit on the total number of processes under execution
12780 system-wide or by a single user {CHILD_MAX} would be exceeded.
- 12781 The *fork()* function may fail if:

12782 [ENOMEM] Insufficient storage space is available.

12783 **EXAMPLES**

12784 None.

12785 **APPLICATION USAGE**

12786 None.

12787 **RATIONALE**

12788 Many historical implementations have timing windows where a signal sent to a process group
 12789 (for example, an interactive SIGINT) just prior to or during execution of *fork()* is delivered to the
 12790 parent following the *fork()* but not to the child because the *fork()* code clears the child's set of
 12791 pending signals. This volume of IEEE Std. 1003.1-200x does not require, or even permit, this
 12792 behavior. However, it is pragmatic to expect that problems of this nature may continue to exist
 12793 in implementations that appear to conform to this volume of IEEE Std. 1003.1-200x and pass
 12794 available verification suites. This behavior is only a consequence of the implementation failing to
 12795 make the interval between signal generation and delivery totally invisible. From the
 12796 application's perspective, a *fork()* call should appear atomic. A signal that is generated prior to
 12797 the *fork()* should be delivered prior to the *fork()*. A signal sent to the process group after the
 12798 *fork()* should be delivered to both parent and child. The implementation may actually initialize
 12799 internal data structures corresponding to the child's set of pending signals to include signals
 12800 sent to the process group during the *fork()*. Since the *fork()* call can be considered as atomic
 12801 from the application's perspective, the set would be initialized as empty and such signals would
 12802 have arrived after the *fork()*; see also <signal.h>.

12803 One approach that has been suggested to address the problem of signal inheritance across *fork()*
 12804 is to add an [EINTR] error, which would be returned when a signal is detected during the call.
 12805 While this is preferable to losing signals, it was not considered an optimal solution. Although it
 12806 is not recommended for this purpose, such an error would be an allowable extension for an
 12807 implementation.

12808 The [ENOMEM] error value is reserved for those implementations that detect and distinguish
 12809 such a condition. This condition occurs when an implementation detects that there is not enough
 12810 memory to create the process. This is intended to be returned when [EAGAIN] is inappropriate
 12811 because there can never be enough memory (either primary or secondary storage) to perform the
 12812 operation. Because *fork()* duplicates an existing process, this must be a condition where there is
 12813 sufficient memory for one such process, but not for two. Many historical implementations
 12814 actually return [ENOMEM] due to temporary lack of memory, a case that is not generally
 12815 distinct from [EAGAIN] from the perspective of a portable application.

12816 Part of the reason for including the optional error [ENOMEM] is because the SVID specifies it
 12817 and it should be reserved for the error condition specified there. The condition is not applicable
 12818 on many implementations.

12819 IEEE Std. 1003.1-1988 neglected to require concurrent execution of the parent and child of *fork()*.
 12820 A system that single-threads processes was clearly not intended and is considered an
 12821 unacceptable "toy implementation" of this volume of IEEE Std. 1003.1-200x. The only objection
 12822 anticipated to the phrase "executing independently" is testability, but this assertion should be
 12823 testable. Such tests require that both the parent and child can block on a detectable action of the
 12824 other, such as a write to a pipe or a signal. An interactive exchange of such actions should be
 12825 possible for the system to conform to the intent of this volume of IEEE Std. 1003.1-200x.

12826 The [EAGAIN] error exists to warn applications that such a condition might occur. Whether it
 12827 occurs or not is not in any practical sense under the control of the application because the
 12828 condition is usually a consequence of the user's use of the system, not of the application's code.
 12829 Thus, no application can or should rely upon its occurrence under any circumstances, nor

12830 should the exact semantics of what concept of “user” is used be of concern to the application
12831 writer. Validation writers should be cognizant of this limitation.

12832 There are two reasons why POSIX programmers call *fork()*. One reason is to create a new thread
12833 of control within the same program (which was originally only possible in POSIX by creating a
12834 new process); the other is to create a new process running a different program. In the latter case,
12835 the call to *fork()* is soon followed by a call to one of the *exec* functions.

12836 The general problem with making *fork()* work in a multi-threaded world is what to do with all
12837 of the threads. There are two alternatives. One is to copy all of the threads into the new process.
12838 This causes the programmer or implementation to deal with threads that are suspended on
12839 system calls or that might be about to execute system calls that should not be executed in the
12840 new process. The other alternative is to copy only the thread that calls *fork()*. This creates the
12841 difficulty that the state of process-local resources is usually held in process memory. If a thread
12842 that is not calling *fork()* holds a resource, that resource is never released in the child process
12843 because the thread whose job it is to release the resource does not exist in the child process.

12844 When a programmer is writing a multi-threaded program, the first described use of *fork()*,
12845 creating new threads in the same program, is provided by the *pthread_create()* function. The
12846 *fork()* function is thus used only to run new programs, and the effects of calling functions that
12847 require certain resources between the call to *fork()* and the call to an *exec* function are undefined.

12848 The addition of the *forkall()* function to the standard was considered and rejected. The *forkall()*
12849 function lets all the threads in the parent be duplicated in the child. This essentially duplicates
12850 the state of the parent in the child. This allows threads in the child to continue processing and
12851 allows locks and the state to be preserved without explicit *pthread_atfork()* code. The calling
12852 process has to ensure that the threads processing state that is shared between the parent and
12853 child (that is, file descriptors or MAP_SHARED memory) behaves properly after *forkall()*. For
12854 example, if a thread is reading a file descriptor in the parent when *forkall()* is called, then two
12855 threads (one in the parent and one in the child) are reading the file descriptor after the *forkall()*.
12856 If this is not desired behavior, the parent process has to synchronize with such threads before
12857 calling *forkall()*.

12858 When *forkall()* is called, threads, other than the calling thread, that are in POSIX System
12859 Interfaces functions that can return with an [EINTR] error may have those functions return
12860 [EINTR] if the implementation cannot ensure that the function behaves correctly in the parent
12861 and child. In particular, *pthread_cond_wait()* and *pthread_cond_timedwait()* need to return in order
12862 to ensure that the condition has not changed. These functions can be awakened by a spurious
12863 condition wakeup rather than returning [EINTR].

12864 FUTURE DIRECTIONS

12865 None.

12866 SEE ALSO

12867 *alarm()*, *exec*, *fcntl()*, *posix_trace_attr_getinherited()*, *posix_trace_trid_eventid_open()*, *semop()*,
12868 *signal()*, *times()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<sys/types.h>`,
12869 `<unistd.h>`

12870 CHANGE HISTORY

12871 First released in Issue 1. Derived from Issue 1 of the SVID.

12872 Issue 4

12873 The `<sys/types.h>` header is now marked as optional (OH); this header need not be included on
12874 XSI-conformant systems.

12875 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 12876
 - The argument list is explicitly defined as **void**.
- 12877
 - Though functionally identical to Issue 3, the DESCRIPTION has been reorganized to improve clarity and to align more closely with the ISO POSIX-1 standard.
- 12878
- 12879
 - The description of the [EAGAIN] error is updated to indicate that this error can also be returned if a system lacks the resources to create another process.
- 12880
- 12881 **Issue 4, Version 2**
- 12882 The DESCRIPTION is changed for X/OPEN UNIX conformance to identify that interval timers
- 12883 are reset in the child process.
- 12884 **Issue 5**
- 12885 The DESCRIPTION is changed for alignment with the POSIX Realtime Extension and the POSIX
- 12886 Threads Extension.
- 12887 **Issue 6**
- 12888 The following new requirements on POSIX implementations derive from alignment with the
- 12889 Single UNIX Specification:
 - The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- 12890
- 12891
- 12892
- 12893 The following changes were made to align with the IEEE P1003.1a draft standard:
 - The effect of `fork()` on a pending alarm call in the child process is clarified.
- 12894
- 12895 The description of CPU-time clock semantics is added for alignment with
- 12896 IEEE Std. 1003.1d-1999. |
- 12897 The description of tracing semantics is added for alignment with IEEE Std. 1003.1q-2000. |

12898 **NAME**

12899 `fpathconf`, `pathconf` — get configurable path name variables

12900 **SYNOPSIS**

12901 `#include <unistd.h>`

12902 `long fpathconf(int fildev, int name);`

12903 `long pathconf(const char *path, int name);`

12904 **DESCRIPTION**

12905 The `fpathconf()` and `pathconf()` functions provide a method for the application to determine the
 12906 current value of a configurable limit or option (*variable*) that is associated with a file or directory.

12907 For `pathconf()`, the *path* argument points to the path name of a file or directory.

12908 For `fpathconf()`, the *fildev* argument is an open file descriptor.

12909 The *name* argument represents the variable to be queried relative to that file or directory.
 12910 Implementations shall support all of the variables listed in the following table and may support
 12911 others. The variables in the following table come from `<limits.h>` or `<unistd.h>` and the
 12912 symbolic constants, defined in `<unistd.h>`, are the corresponding values used for *name*. Support
 12913 for some path name configuration variables is dependent on implementation options (see
 12914 shading and margin codes in the table below). Where an implementation option is not
 12915 supported, the variable need not be supported.

12916

12917

	Variable	Value of <i>name</i>	Notes
12918	{FILESIZEBITS}	_PC_FILESIZEBITS	3, 4
12919	{LINK_MAX}	_PC_LINK_MAX	1
12920	{MAX_CANON}	_PC_MAX_CANON	2
12921	{MAX_INPUT}	_PC_MAX_INPUT	2
12922	{NAME_MAX}	_PC_NAME_MAX	3, 4
12923	{PATH_MAX}	_PC_PATH_MAX	4, 5
12924	{PIPE_BUF}	_PC_PIPE_BUF	6
12925 ADV	{POSIX_ALLOC_SIZE_MIN}	_PC_ALLOC_SIZE_MIN	
12926 ADV	{POSIX_REC_INCR_XFER_SIZE}	_PC_REC_INCR_XFER_SIZE	
12927 ADV	{POSIX_REC_MAX_XFER_SIZE}	_PC_REC_MAX_XFER_SIZE	
12928 ADV	{POSIX_REC_MIN_XFER_SIZE}	_PC_REC_MIN_XFER_SIZE	
12929 ADV	{POSIX_REC_XFER_ALIGN}	_PC_REC_XFER_ALIGN	
12930	{SYMLINK_MAX}	_PC_SYMLINK_MAX	4, 9
12931	_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
12932	_POSIX_NO_TRUNC	_PC_NO_TRUNC	3, 4
12933	_POSIX_VDISABLE	_PC_VDISABLE	2
12934	_POSIX_ASYNC_IO	_PC_ASYNC_IO	8
12935	_POSIX_PRIO_IO	_PC_PRIO_IO	8
12936	_POSIX_SYNC_IO	_PC_SYNC_IO	8

12937 **Notes:**

- 12938 1. If *path* or *fildev* refers to a directory, the value returned applies to the directory
 12939 itself.
- 12940 2. If *path* or *fildev* does not refer to a terminal file, it is unspecified whether an
 12941 implementation supports an association of the variable name with the specified
 12942 file.

- 12943 3. If *path* or *filde*s refers to a directory, the value returned applies to file names
12944 within the directory.
- 12945 4. If *path* or *filde*s does not refer to a directory, it is unspecified whether an
12946 implementation supports an association of the variable name with the specified
12947 file.
- 12948 5. If *path* or *filde*s refers to a directory, the value returned is the maximum length
12949 of a relative path name when the specified directory is the working directory.
- 12950 6. If *path* refers to a FIFO, or *filde*s refers to a pipe or FIFO, the value returned
12951 applies to the referenced object. If *path* or *filde*s refers to a directory, the value
12952 returned applies to any FIFO that exists or can be created within the directory.
12953 If *path* or *filde*s refers to any other type of file, it is unspecified whether an
12954 implementation supports an association of the variable name with the specified
12955 file.
- 12956 7. If *path* or *filde*s refers to a directory, the value returned applies to any files, other
12957 than directories, that exist or can be created within the directory.
- 12958 8. If *path* or *filde*s refers to a directory, it is unspecified whether an implementation
12959 supports an association of the variable name with the specified file.
- 12960 9. If *path* or *filde*s refers to a directory, the value returned is the maximum length
12961 of the string that a symbolic link in that directory can contain.

12962 RETURN VALUE

12963 If *name* is an invalid value, both *pathconf()* and *fpathconf()* shall return -1 and set *errno* to
12964 indicate the error.

12965 If the variable corresponding to *name* has no limit for the *path* or file descriptor, both *pathconf()*
12966 and *fpathconf()* shall return -1 without changing *errno*. If the implementation needs to use *path*
12967 to determine the value of *name* and the implementation does not support the association of *name*
12968 with the file specified by *path*, or if the process did not have appropriate privileges to query the
12969 file specified by *path*, or *path* does not exist, *pathconf()* shall return -1 and set *errno* to indicate the
12970 error.

12971 If the implementation needs to use *filde*s to determine the value of *name* and the implementation
12972 does not support the association of *name* with the file specified by *filde*s, or if *filde*s is an invalid
12973 file descriptor, *fpathconf()* shall return -1 and set *errno* to indicate the error.

12974 Otherwise, *pathconf()* or *fpathconf()* shall return the current variable value for the file or
12975 directory without changing *errno*. The value returned shall not be more restrictive than the
12976 corresponding value available to the application when it was compiled with the
12977 implementation's `<limits.h>` or `<unistd.h>`.

12978 ERRORS

12979 The *pathconf()* function shall fail if:

- | | | | |
|-------|----------|--|--|
| 12980 | [EINVAL] | The value of <i>name</i> is not valid. | |
| 12981 | [ELOOP] | A loop exists in symbolic links encountered during resolution of the <i>path</i> | |
| 12982 | | argument. | |

12983 The *pathconf()* function may fail if:

- | | | | |
|-------|----------|---|--|
| 12984 | [EACCES] | Search permission is denied for a component of the path prefix. | |
| 12985 | [EINVAL] | The implementation does not support an association of the variable <i>name</i> with | |
| 12986 | | the specified file. | |

12987	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
12988		
12989	[ENAMETOOLONG]	
12990		The length of the <i>path</i> argument exceeds {PATH_MAX} or a path name component is longer than {NAME_MAX}.
12991		
12992	[ENAMETOOLONG]	
12993		As a result of encountering a symbolic link in resolution of the <i>path</i> argument, the length of the substituted path name string exceeded {PATH_MAX}.
12994		
12995	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
12996	[ENOTDIR]	A component of the path prefix is not a directory.
12997		The <i>fpathconf()</i> function shall fail if:
12998	[EINVAL]	The value of <i>name</i> is not valid.
12999		The <i>fpathconf()</i> function may fail if:
13000	[EBADF]	The <i>fdes</i> argument is not a valid file descriptor.
13001	[EINVAL]	The implementation does not support an association of the variable <i>name</i> with the specified file.
13002		
13003	EXAMPLES	
13004		None.
13005	APPLICATION USAGE	
13006		None.
13007	RATIONALE	
13008		The <i>pathconf()</i> function was proposed immediately after the <i>sysconf()</i> function when it was realized that some configurable values may differ across file system, directory, or device boundaries.
13009		
13010		
13011		For example, {NAME_MAX} frequently changes between System V and BSD-based file systems; System V uses a maximum of 14, BSD 255. On an implementation that provides both types of file systems, an application would be forced to limit all path name components to 14 bytes, as this would be the value specified in <limits.h> on such a system.
13012		
13013		
13014		
13015		Therefore, various useful values can be queried on any path name or file descriptor, assuming that the appropriate permissions are in place.
13016		
13017		The value returned for the variable {PATH_MAX} indicates the longest relative path name that could be given if the specified directory is the process' current working directory. A process may not always be able to generate a name that long and use it if a subdirectory in the path name crosses into a more restrictive file system.
13018		
13019		
13020		
13021		The value returned for the variable _POSIX_CHOWN_RESTRICTED also applies to directories that do not have file systems mounted on them. The value may change when crossing a mount point, so applications that need to know should check for each directory. (An even easier check is to try the <i>chown()</i> function and look for an error in case it happens.)
13022		
13023		
13024		
13025		Unlike the values returned by <i>sysconf()</i> , the path name-oriented variables are potentially more volatile and are not guaranteed to remain constant throughout the process' lifetime. For example, in between two calls to <i>pathconf()</i> , the file system in question may have been unmounted and remounted with different characteristics.
13026		
13027		
13028		

13029 Also note that most of the errors are optional. If one of the variables always has the same value
 13030 on an implementation, the implementation need not look at *path* or *filde*s to return that value and
 13031 is, therefore, not required to detect any of the errors except the meaning of [EINVAL] that
 13032 indicates that the value of *name* is not valid for that variable.

13033 If the value of any of the limits are indeterminate (logically infinite), they are defined in
 13034 <limits.h> and the *pathconf()* and *fpathconf()* functions return -1 without changing *errno*. This
 13035 can be distinguished from the case of giving an unrecognized *name* argument because *errno* is set
 13036 to [EINVAL] in this case.

13037 Since -1 is a valid return value for the *pathconf()* and *fpathconf()* functions, applications should
 13038 set *errno* to zero before calling them and check *errno* only if the return value is -1.

13039 For the case of {SYMLINK_MAX}, since both *pathconf()* and *open()* follow symbolic links, there
 13040 is no way that *path* or *filde*s could refer to a symbolic link.

13041 FUTURE DIRECTIONS

13042 None.

13043 SEE ALSO

13044 *confstr()*, *sysconf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <limits.h>, <unistd.h>,
 13045 the Shell and Utilities volume of IEEE Std. 1003.1-200x

13046 CHANGE HISTORY

13047 First released in Issue 3.

13048 Entry included for alignment with the POSIX.1-1988 standard.

13049 Issue 4

13050 The *fpathconf()* function now has the full **long** return type in the SYNOPSIS section.

13051 The following changes have been made for alignment with the ISO POSIX-1 standard:

- 13052 • The type of argument *path* is changed from **char*** to **const char***. Also, the return value of
 13053 both functions is changed from **long** to **long**.
- 13054 • In the DESCRIPTION, the words “The behavior is undefined if” have been replaced by “it is
 13055 unspecified whether an implementation supports an association of the variable name with
 13056 the specified file” in notes 2, 4, and 6.
- 13057 • In the RETURN VALUE section, errors associated with the use of *path* and *filde*s, when an
 13058 implementation does not support the requested association, are now specified separately.
- 13059 • The requirement that *errno* be set to indicate the error is added.

13060 The following change is incorporated for alignment with the FIPS requirements:

- 13061 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path
 13062 name component is larger than {NAME_MAX} is now defined as mandatory and marked as
 13063 an extension.

13064 Issue 4, Version 2

13065 The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- 13066 • It states that [ELOOP] is returned if too many symbolic links are encountered during path
 13067 name resolution.
- 13068 • A second [ENAMETOOLONG] condition is defined that may report excessive length of an
 13069 intermediate result of path name resolution of a symbolic link.

13070 **Issue 5**

13071 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

13072 Large File Summit extensions are added.

13073 **Issue 6**

13074 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 13075 • The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.
- 13076 This is since behavior may vary from one file system to another.

13077 The following new requirements on POSIX implementations derive from alignment with the
13078 Single UNIX Specification:

- 13079 • The DESCRIPTION is updated to include {FILESIZEBITS}.
- 13080 • The [ELOOP] mandatory error condition is added.
- 13081 • A second [ENAMETOOLONG] is added as an optional error condition.

13082 The following changes were made to align with the IEEE P1003.1a draft standard:

- 13083 • The `_PC_SYMLINK_MAX` entry is added to the table in the DESCRIPTION.

13084 The `pathconf()` variables {`POSIX_ALLOC_SIZE_MIN`}, {`POSIX_REC_INCR_XFER_SIZE`},
13085 {`POSIX_REC_MAX_XFER_SIZE`}, {`POSIX_REC_MIN_XFER_SIZE`},
13086 {`POSIX_REC_XFER_ALIGN`} and their associated names are added for alignment with |
13087 IEEE Std. 1003.1d-1999.

13088 **NAME**

13089 fpclassify — classify real floating type

13090 **SYNOPSIS**

13091 #include <math.h>

13092 int fpclassify(real-floating x);

13093 **DESCRIPTION**

13094 cx The functionality described on this reference page is aligned with the ISO C standard. Any
13095 conflict between the requirements described here and the ISO C standard is unintentional. This
13096 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

13097 The *fpclassify()* macro shall classify its argument value as NaN, infinite, normal, subnormal,
13098 zero, or into another implementation-defined category. First, an argument represented in a
13099 format wider than its semantic type is converted to its semantic type. Then classification is based
13100 on the type of the argument.

13101 **RETURN VALUE**

13102 The *fpclassify()* macro shall return the value of the number classification macro appropriate to
13103 the value of its argument.

13104 **ERRORS**

13105 No errors are defined.

13106 **EXAMPLES**

13107 None.

13108 **APPLICATION USAGE**

13109 None.

13110 **RATIONALE**

13111 None.

13112 **FUTURE DIRECTIONS**

13113 None.

13114 **SEE ALSO**

13115 *isfinite()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*, the Base Definitions volume of
13116 IEEE Std. 1003.1-200x, <math.h>

13117 **CHANGE HISTORY**

13118 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

13119 NAME

13120 fprintf, printf, snprintf, sprintf — print formatted output

13121 SYNOPSIS

13122 #include <stdio.h>

13123 int fprintf(FILE *restrict *stream*, const char *restrict *format*, ...);13124 int printf(const char *restrict *format*, ...);13125 int snprintf(char *restrict *s*, size_t *n*, const char *restrict *format*, ...);13126 int sprintf(char *restrict *s*, const char *restrict *format*, ...);

13127 DESCRIPTION

13128 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 13129 conflict between the requirements described here and the ISO C standard is unintentional. This
 13130 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

13131 The *fprintf()* function places output on the named output *stream*. The *printf()* function places
 13132 output on the standard output stream *stdout*. The *sprintf()* function places output followed by
 13133 the null byte, '\0', in consecutive bytes starting at *s*; it is the user's responsibility to ensure that
 13134 enough space is available.

13135 XSI The *snprintf()* function is identical to *sprintf()* with the addition of the *n* argument, which states
 13136 the size of the buffer referred to by *s*. If *n* is greater than zero, but not large enough to hold all
 13137 output bytes specified by the format, output bytes beyond the *n*-1st are discarded rather than
 13138 being written into the array.

13139 If copying takes place between objects that overlap as a result of a call to *sprintf()* or *snprintf()*,
 13140 the results are undefined.

13141 Each of these functions converts, formats, and prints its arguments under control of the *format*.
 13142 The *format* is a character string, beginning and ending in its initial shift state, if any. The *format* is
 13143 composed of zero or more directives: *ordinary characters*, which are simply copied to the output
 13144 stream, and *conversion specifications*, each of which results in the fetching of zero or more
 13145 arguments. The results are undefined if there are insufficient arguments for the *format*. If the
 13146 *format* is exhausted while arguments remain, the excess arguments are evaluated but are
 13147 otherwise ignored.

13148 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
 13149 to the next unused argument. In this case, the conversion character '%' (see below) is replaced
 13150 by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL_ARGMAX}], giving the
 13151 position of the argument in the argument list. This feature provides for the definition of format
 13152 strings that select arguments in an order appropriate to specific languages (see the EXAMPLES
 13153 section).

13154 In format strings containing the "%n\$" form of conversion specifications, numbered arguments
 13155 in the argument list can be referenced from the format string as many times as required.

13156 In format strings containing the '%' form of conversion specifications, each argument in the
 13157 argument list is used exactly once.

13158 All forms of the *fprintf()* functions allow for the insertion of a language-dependent radix
 13159 character in the output string. The radix character is defined in the program's locale (category
 13160 LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the
 13161 radix character defaults to a period ('.').

13162 XSI Each conversion specification is introduced by the '%' character or by the character sequence
 13163 "%n\$", after which the following appear in sequence:

- 13164 • Zero or more *flags* (in any order), which modify the meaning of the conversion specification.
- 13165 • An optional minimum *field width*. If the converted value has fewer bytes than the field
13166 width, it shall be padded with spaces by default on the left; it shall be padded on the right, if
13167 the left-adjustment flag ('-'), described below, is given to the field width. The field width
13168 takes the form of an asterisk ('*'), described below, or a decimal integer.
- 13169 • An optional *precision* that gives the minimum number of digits to appear for the *d*, *i*, *o*, *u*, *x*,
13170 and *X* conversions; the number of digits to appear after the radix character for the *e*, *E*, and *f*
13171 conversions; the maximum number of significant digits for the *g* and *G* conversions; or the
13172 XSI maximum number of bytes to be printed from a string in *s* and *S* conversions. The precision
13173 takes the form of a period ('.') followed either by an asterisk ('*'), described below, or an
13174 optional decimal digit string, where a null digit string is treated as 0. If a precision appears
13175 with any other conversion character, the behavior is undefined.
- 13176 • An optional length modifier that specifies the size of the argument.
- 13177 • A *conversion character* that indicates the type of conversion to be applied.
- 13178 A field width, or precision, or both, may be indicated by an asterisk ('*'). In this case an
13179 argument of type **int** supplies the field width or precision. Applications shall ensure that
13180 arguments specifying field width, or precision, or both appear in that order before the argument,
13181 if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field
13182 XSI width. A negative precision is taken as if the precision were omitted. In format strings
13183 containing the "%n\$" form of a conversion specification, a field width or precision may be
13184 indicated by the sequence "*m\$", where *m* is a decimal integer in the range [1,{NL_ARGMAX}]
13185 giving the position in the argument list (after the format argument) of an integer argument
13186 containing the field width or precision, for example:
- 13187

```
printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```
- 13188 The *format* can contain either numbered argument specifications (that is, "%n\$" and "*m\$"), or
13189 unnumbered argument specifications (that is, '%' and '*'), but normally not both. The only
13190 exception to this is that "%%" can be mixed with the "%n\$" form. The results of mixing
13191 numbered and unnumbered argument specifications in a *format* string are undefined. When
13192 numbered argument specifications are used, specifying the *N*th argument requires that all the
13193 leading arguments, from the first to the (*N*-1)th, are specified in the format string.
- 13194 The flag characters and their meanings are:
- 13195 XSI ' The integer portion of the result of a decimal conversion (%i, %d, %u, %f, %g, or %G)
13196 shall be formatted with thousands' grouping characters. For other conversions the
13197 behavior is undefined. The non-monetary grouping character is used.
- 13198 - The result of the conversion shall be left-justified within the field. The conversion is
13199 right-justified if this flag is not specified.
- 13200 + The result of a signed conversion shall always begin with a sign ('+' or '-'). The
13201 conversion shall begin with a sign only when a negative value is converted if this flag is
13202 not specified.
- 13203 <space> If the first character of a signed conversion is not a sign or if a signed conversion results
13204 in no characters, a space shall be prefixed to the result. This means that if the space and
13205 '+' flags both appear, the space flag shall be ignored.
- 13206 # This flag specifies that the value is to be converted to an alternative form. For *o*
13207 conversion, it increases the precision (if necessary) to force the first digit of the result to
13208 be 0. For *x* or *X* conversions, a non-zero result shall have 0x (or 0X) prefixed to it. For *e*,
13209 *E*, *f*, *g*, or *G* conversions, the result shall always contain a radix character, even if no

13210 digits follow the radix character. Without this flag, a radix character appears in the
 13211 result of these conversions only if a digit follows it. For *g* and *G* conversions, trailing
 13212 zeros shall *not* be removed from the result as they normally are. For other conversions,
 13213 the behavior is undefined.

13214 **0** For *d*, *i*, *o*, *u*, *x*, *X*, *e*, *E*, *f*, *g*, and *G* conversions, leading zeros (following any indication of
 13215 sign or base) are used to pad to the field width; no space padding is performed. If the
 13216 '0' and '-' flags both appear, the '0' flag is ignored. For *d*, *i*, *o*, *u*, *x*, and *X*
 13217 XSJ conversions, if a precision is specified, the '0' flag is ignored. If the '0' and '\\'
 13218 flags both appear, the grouping characters are inserted before zero padding. For other
 13219 conversions, the behavior is undefined.

13220 The length modifiers and their meanings are:

13221 **hh** Specifies that a following *d*, *i*, *o*, *u*, *x*, or *X* conversion specifier applies to a **signed char**
 13222 or **unsigned char** argument (the argument will have been promoted according to the
 13223 integer promotions, but its value shall be converted to **signed char** or **unsigned char**
 13224 before printing); or that a following *n* conversion specifier applies to a pointer to a
 13225 **signed char** argument.

13226 **h** Specifies that a following *d*, *i*, *o*, *u*, *x*, or *X* conversion specifier applies to a **short** or
 13227 **unsigned short** argument (the argument will have been promoted according to the
 13228 integer promotions, but its value shall be converted to **short** or **unsigned short** before
 13229 printing); or that a following *n* conversion specifier applies to a pointer to a **short**
 13230 argument.

13231 **l(ell)** Specifies that a following *d*, *i*, *o*, *u*, *x*, or *X* conversion specifier applies to a **long** or
 13232 **unsigned long** argument; that a following *n* conversion specifier applies to a pointer to
 13233 a **long** argument; that a following *c* conversion specifier applies to a **wint_t** argument;
 13234 that a following *s* conversion specifier applies to a pointer to a **wchar_t** argument; or
 13235 has no effect on a following *a*, *A*, *e*, *E*, *f*, *F*, *g*, or *G* conversion specifier.

13236 **ll(ell-ell)** Specifies that a following *d*, *i*, *o*, *u*, *x*, or *X* conversion specifier applies to a **long long** or
 13237 **unsigned long long** argument; or that a following *n* conversion specifier applies to a
 13238 pointer to a **long long** argument.

13239 **j** Specifies that a following *d*, *i*, *o*, *u*, *x*, or *X* conversion specifier applies to an **intmax_t** or
 13240 **uintmax_t** argument; or that a following *n* conversion specifier applies to a pointer to
 13241 an **intmax_t** argument.

13242 **z** Specifies that a following *d*, *i*, *o*, *u*, *x*, or *X* conversion specifier applies to a **size_t** or the
 13243 corresponding signed integer type argument; or that a following *n* conversion specifier
 13244 applies to a pointer to a signed integer type corresponding to **size_t** argument.

13245 **t** Specifies that a following *d*, *i*, *o*, *u*, *x*, or *X* conversion specifier applies to a **ptrdiff_t** or
 13246 the corresponding **unsigned** type argument; or that a following *n* conversion specifier
 13247 applies to a pointer to a **ptrdiff_t** argument.

13248 **L** Specifies that a following *a*, *A*, *e*, *E*, *f*, *F*, *g*, or *G* conversion specifier applies to a **long**
 13249 **double** argument.

13250 If a length modifier appears with any conversion specifier other than as specified above, the
 13251 behavior is undefined.

13252 The conversion characters and their meanings are:

13253 **d, i** The **int** argument is converted to a signed decimal in the style [-]dddd. The precision
 13254 specifies the minimum number of digits to appear; if the value being converted can be
 13255 represented in fewer digits, it shall be expanded with leading zeros. The default

13256		precision is 1. The result of converting 0 with an explicit precision of 0 is no characters.
13257	<i>o</i>	The unsigned argument is converted to unsigned octal format in the style <i>ddd</i> . The
13258		precision specifies the minimum number of digits to appear; if the value being
13259		converted can be represented in fewer digits, it shall be expanded with leading zeros.
13260		The default precision is 1. The result of converting 0 with an explicit precision of 0 is no
13261		characters.
13262	<i>u</i>	The unsigned argument is converted to unsigned decimal format in the style <i>ddd</i> . The
13263		precision specifies the minimum number of digits to appear; if the value being
13264		converted can be represented in fewer digits, it shall be expanded with leading zeros.
13265		The default precision is 1. The result of converting 0 with an explicit precision of 0 is no
13266		characters.
13267	<i>x</i>	The unsigned argument is converted to unsigned hexadecimal format in the style <i>ddd</i> ;
13268		the letters "abcdef" are used. The precision specifies the minimum number of digits
13269		to appear; if the value being converted can be represented in fewer digits, it shall be
13270		expanded with leading zeros. The default precision is 1. The result of converting 0 with
13271		an explicit precision of 0 is no characters.
13272	<i>X</i>	Behaves the same as the <i>x</i> conversion character except that letters "ABCDEF" are used
13273		instead of "abcdef".
13274	<i>f, F</i>	The double argument is converted to decimal notation in the style <i>[-]ddd.ddd</i> , where
13275		the number of digits after the radix character is equal to the precision specification. If
13276		the precision is missing, it is taken as 6; if the precision is explicitly 0 and no '#' flag is
13277		present, no radix character appears. If a radix character appears, at least one digit
13278		appears before it. The value is rounded to the appropriate number of digits.
13279		A double argument representing an infinity is converted in one of the styles <i>[-]inf</i> or
13280		<i>[-]infinity</i> ; which style is implementation-defined. A double argument representing a
13281		NaN is converted in one of the styles <i>[-]nan</i> or <i>[-]nan(n-char-sequence)</i> ; which style, and
13282		the meaning of any <i>n-char-sequence</i> , is implementation-defined. The <i>F</i> conversion
13283		specifier produces INF, INFINITY, or NAN instead of inf, infinity, or nan, respectively.
13284	<i>e, E</i>	The double argument is converted in the style <i>[-]d.ddde±dd</i> , where there is one digit
13285		before the radix character (which is non-zero if the argument is non-zero) and the
13286		number of digits after it is equal to the precision; if the precision is missing, it is taken
13287		as 6; if the precision is 0 and no '#' flag is present, no radix character appears. The
13288		value is rounded to the appropriate number of digits. The <i>E</i> conversion character will
13289		produce a number with <i>E</i> instead of <i>e</i> introducing the exponent. The exponent always
13290		contains at least two digits. If the value is 0, the exponent is 0.
13291		A double argument representing an infinity or NaN is converted in the style of an <i>f</i> or <i>F</i>
13292		conversion specifier.
13293	<i>g, G</i>	The double argument is converted in the style <i>f</i> or <i>e</i> (or in the style <i>E</i> in the case of a <i>G</i>
13294		conversion character), with the precision specifying the number of significant digits. If
13295		an explicit precision is 0, it is taken as 1. The style used depends on the value
13296		converted; style <i>e</i> (or <i>E</i>) shall be used only if the exponent resulting from such a
13297		conversion is less than -4 or greater than or equal to the precision. Trailing zeros are
13298		removed from the fractional portion of the result; a radix character appears only if it is
13299		followed by a digit.
13300		A double argument representing an infinity or NaN is converted in the style of an <i>f</i> or <i>F</i>
13301		conversion specifier.

13302	a, A	A double argument representing a floating-point number is converted in the style <code>[-]0xh.hhhh p1d</code> , where there is one hexadecimal digit (which is non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point character 235) and the number of hexadecimal digits after it is equal to the precision; if the precision is missing and <code>FLT_RADIX</code> is a power of 2, then the precision is sufficient for an exact representation of the value; if the precision is missing and <code>FLT_RADIX</code> is not a power of 2, then the precision is sufficient to distinguish 236) values of type double , except that trailing zeros may be omitted; if the precision is zero and the <code>'#'</code> flag is not specified, no decimal-point character appears. The letters <code>"abcdef"</code> are used for a conversion and the letters <code>"ABCDEF"</code> for A conversion. The A conversion specifier produces a number with <code>'X'</code> and <code>'P'</code> instead of <code>'x'</code> and <code>'p'</code> . The exponent always contains at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent is zero.
13303		
13304		
13305		
13306		
13307		
13308		
13309		
13310		
13311		
13312		A double argument representing an infinity or NaN is converted in the style of an <i>f</i> or <i>F</i> conversion specifier.
13317		
13318	c	The int argument is converted to an unsigned char , and the resulting byte is written.
13319		If an <i>l</i> (ell) qualifier is present, the wint_t argument is converted as if by an <i>ls</i> conversion specification with no precision and an argument that points to a two-element array of type wchar_t , the first element of which contains the wint_t argument to the <i>ls</i> conversion specification and the second element contains a null wide character.
13320		
13321		
13322		
13323	s	The argument shall be a pointer to an array of char . Bytes from the array are written up to (but not including) any terminating null byte. If the precision is specified, no more than that many bytes shall be written. If the precision is not specified or is greater than the size of the array, the application shall ensure that the array contains a null byte.
13324		
13325		
13326		
13327		
13328		
13329		
13330		
13331		
13332		
13333	p	If an <i>l</i> (ell) qualifier is present, the argument shall be a pointer to an array of type wchar_t . Wide characters from the array are converted to characters (each as if by a call to the <i>wcrtomb()</i> function, with the conversion state described by an mbstate_t object initialized to zero before the first wide character is converted) up to and including a terminating null wide character. The resulting characters shall be written up to (but not including) the terminating null character (byte). If no precision is specified, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many characters (bytes) shall be written (including shift sequences, if any), and the array shall contain a null wide character if, to equal the character sequence length given by the precision, the function would need to access a wide character one past the end of the array. In no case is a partial character written.
13334		
13335		
13336		
13337		
13338		
13339		
13340		
13341		
13342		
13343		
13344		
13345	XSI	C Same as <i>lc</i> .
13346	XSI	S Same as <i>ls</i> .
13347	%	Print a <code>'%'</code> ; no argument is converted. The entire conversion specification shall be <code>"%%"</code> .
13348		

13349 If a conversion specification does not match one of the above forms, the behavior is undefined. If
 13350 any argument is not the correct type for the corresponding conversion specification, the
 13351 behavior is undefined.

13352 In no case does a nonexistent or small field width cause truncation of a field; if the result of a
 13353 conversion is wider than the field width, the field is simply expanded to contain the conversion
 13354 result. Characters generated by *fprintf()* and *printf()* are printed as if *fputc()* had been called.

13355 For *a* and *A* conversions, if `FLT_RADIX` is a power of 2, the value is correctly rounded to a
 13356 hexadecimal floating number with the given precision.

13357 If `FLT_RADIX` is not a power of 2, the result should be one of the two adjacent numbers in
 13358 hexadecimal floating style with the given precision, with the extra stipulation that the error
 13359 should have a correct sign for the current rounding direction.

13360 For *e*, *E*, *f*, *F*, *g*, and *G* conversions, if the number of significant decimal digits is at most
 13361 `DECIMAL_DIG`, then the result should be correctly rounded. If the number of significant
 13362 decimal digits is more than `DECIMAL_DIG` but the source value is exactly representable with
 13363 `DECIMAL_DIG` digits, then the result should be an exact representation with trailing zeros.
 13364 Otherwise, the source value is bounded by two adjacent decimal strings "*L* < *U*", both having
 13365 `DECIMAL_DIG` significant digits; the value of the resultant decimal string "*D*" should satisfy "*L*
 13366 <= *D* <= *U*", with the extra stipulation that the error should have a correct sign for the current
 13367 rounding direction.

13368 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the call to a
 13369 successful execution of *fprintf()* or *printf()* and the next successful completion of a call to *fflush()*
 13370 or *fclose()* on the same stream or a call to *exit()* or *abort()*.

13371 RETURN VALUE

13372 Upon successful completion, the *fprintf()* and *printf()* functions shall return the number of bytes
 13373 transmitted.

13374 Upon successful completion, the *sprintf()* function shall return the number of bytes written to *s*,
 13375 excluding the terminating null byte.

13376 Upon successful completion, the *snprintf()* function shall return the number of bytes that would
 13377 be written to *s* had *n* been sufficiently large excluding the terminating null byte.

13378 If an output error was encountered, these functions shall return a negative value.

13379 If the value of *n* is zero on a call to *snprintf()*, nothing shall be written, the number of bytes that
 13380 would have been written had *n* been sufficiently large excluding the terminating null shall be
 13381 returned, and *s* may be a null pointer.

13382 ERRORS

13383 For the conditions under which *fprintf()* and *printf()* fail and may fail, refer to *fputc()* or
 13384 *fputwc()*.

13385 In addition, all forms of *fprintf()* may fail if:

13386 XSI [EILSEQ] A wide-character code that does not correspond to a valid character has been
 13387 detected.

13388 XSI [EINVAL] There are insufficient arguments.

13389 The *printf()* and *fprintf()* functions may fail if:

13390 XSI [ENOMEM] Insufficient storage space is available.

13391 The *snprintf()* function shall fail if:

13392 XSI [Eoverflow] The value of *n* is greater than {INT_MAX} or the number of bytes needed to
 13393 hold the output excluding the terminating null is greater than {INT_MAX}.

13394 EXAMPLES

13395 **Printing Language-Independent Date and Time**

13396 The following statement can be used to print date and time using a language-independent
 13397 format:

```
13398 printf(format, weekday, month, day, hour, min);
```

13399 For American usage, *format* could be a pointer to the following string:

```
13400 "%s, %s %d, %d:%.2d\n"
```

13401 This example would produce the following message:

```
13402 Sunday, July 3, 10:02
```

13403 For German usage, *format* could be a pointer to the following string:

```
13404 "%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

13405 This definition of *format* would produce the following message:

```
13406 Sonntag, 3. Juli, 10:02
```

13407 **Printing File Information**

13408 The following example prints information about the type, permissions, and number of links of a
 13409 specific file in a directory.

13410 The first two calls to *printf()* use data decoded from a previous *stat()* call. The user-defined
 13411 *strperm()* function shall return a string similar to the one at the beginning of the output for the
 13412 following command:

```
13413 ls -l
```

13414 The next call to *printf()* outputs the owner's name if it is found using *getpwuid()*; the *getpwuid()*
 13415 function shall return a **passwd** structure from which the name of the user is extracted. If the user
 13416 name is not found, the program instead prints out the numeric value of the user ID.

13417 The next call prints out the group name if it is found using *getgrgid()*; *getgrgid()* is very similar to
 13418 *getpwuid()* except that it shall return group information based on the group number. Once
 13419 again, if the group is not found, the program prints the numeric value of the group for the entry.

13420 The final call to *printf()* prints the size of the file.

```
13421 #include <stdio.h>
```

```
13422 #include <sys/types.h>
```

```
13423 #include <pwd.h>
```

```
13424 #include <grp.h>
```

```
13425 char *strperm (mode_t);
```

```
13426 ...
```

```
13427 struct stat statbuf;
```

```
13428 struct passwd *pwd;
```

```
13429 struct group *grp;
```

```
13430 ...
```

```
13431 printf("%10.10s", strperm (statbuf.st_mode));
```

```

13432     printf("%4d", statbuf.st_nlink);
13433     if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
13434         printf(" %-8.8s", pwd->pw_name);
13435     else
13436         printf(" %-8ld", (long) statbuf.st_uid);
13437     if ((grp = getgrgid(statbuf.st_gid)) != NULL)
13438         printf(" %-8.8s", grp->gr_name);
13439     else
13440         printf(" %-8ld", (long) statbuf.st_gid);
13441     printf("%9jd", (intmax_t) statbuf.st_size);
13442     ...

```

13443 **Printing a Localized Date String**

13444 The following example gets a localized date string. The *nl_langinfo()* function shall return the
 13445 localized date string, which specifies the order and layout of the date. The *strftime()* function
 13446 takes this information and, using the **tm** structure for values, places the date and time
 13447 information into *datestring*. The *printf()* function then outputs *datestring* and the name of the
 13448 entry.

```

13449     #include <stdio.h>
13450     #include <time.h>
13451     #include <langinfo.h>
13452     ...
13453     struct dirent *dp;
13454     struct tm *tm;
13455     char datestring[256];
13456     ...
13457     strftime(datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
13458     printf(" %s %s\n", datestring, dp->d_name);
13459     ...

```

13460 **Printing Error Information**

13461 The following example uses *fprintf()* to write error information to standard error.

13462 In the first group of calls, the program tries to open the password lock file named **LOCKFILE**. If
 13463 the file already exists, this is an error, as indicated by the **O_EXCL** flag on the *open()* function. If
 13464 the call fails, the program assumes that someone else is updating the password file, and the
 13465 program exits.

13466 The next group of calls saves a new password file as the current password file by creating a link
 13467 between **LOCKFILE** and the new password file **PASSWDFILE**.

```

13468     #include <sys/types.h>
13469     #include <sys/stat.h>
13470     #include <fcntl.h>
13471     #include <stdio.h>
13472     #include <stdlib.h>
13473     #include <unistd.h>
13474     #include <string.h>
13475     #include <errno.h>

```

```

13476     #define LOCKFILE "/etc/ptmp"
13477     #define PASSWDFILE "/etc/passwd"
13478     ...
13479     int pfd;
13480     ...
13481     if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
13482                  S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
13483     {
13484         fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
13485         exit(1);
13486     }
13487     ...
13488     if (link(LOCKFILE, PASSWDFILE) == -1) {
13489         fprintf(stderr, "Link error: %s\n", strerror(errno));
13490         exit(1);
13491     }
13492     ...

```

13493 **Printing Usage Information**

13494 The following example checks to make sure the program has the necessary arguments, and uses
 13495 *fprintf()* to print usage information if the expected number of arguments is not present.

```

13496     #include <stdio.h>
13497     #include <stdlib.h>
13498     ...
13499     char *Options = "hdbt1";
13500     ...
13501     if (argc < 2) {
13502         fprintf(stderr, "Usage: %s -%s <file\n", argv[0], Options); exit(1);
13503     }
13504     ...

```

13505 **Formatting a Decimal String**

13506 The following example prints a key and data pair on *stdout*. Note use of the '*' (asterisk) in the
 13507 format string; this ensures the correct number of decimal places for the element based on the
 13508 number of elements requested.

```

13509     #include <stdio.h>
13510     ...
13511     long i;
13512     char *keyst;
13513     int elementlen, len;
13514     ...
13515     while (len < elementlen) {
13516         ...
13517         printf("%s Element%0*ld\n", keyst, elementlen, i);
13518         ...
13519     }

```

13520 **Creating a File Name**

13521 The following example creates a file name using information from a previous *getpwnam()*
 13522 function that returned the HOME directory of the user.

```
13523 #include <stdio.h>
13524 #include <sys/types.h>
13525 #include <unistd.h>
13526 ...
13527 char filename[PATH_MAX+1];
13528 struct passwd *pw;
13529 ...
13530 sprintf(filename, "%s/%d.out", pw->pw_dir, getpid());
13531 ...
```

13532 **Reporting an Event**

13533 The following example loops until an event has timed out. The *pause()* function waits forever
 13534 unless it receives a signal. The *fprintf()* statement should never occur due to the possible return
 13535 values of *pause()*.

```
13536 #include <stdio.h>
13537 #include <unistd.h>
13538 #include <string.h>
13539 #include <errno.h>
13540 ...
13541 while (!event_complete) {
13542     ...
13543     if (pause() != -1 || errno != EINTR)
13544         fprintf(stderr, "pause: unknown error: %s\n", strerror(errno));
13545 }
13546 ...
```

13547 **Printing Monetary Information**

13548 The following example uses *strfmon()* to convert a number and store it as a formatted monetary
 13549 string named *convbuf*. If the first number is printed, the program prints the format and the
 13550 description; otherwise, it just prints the number.

```
13551 #include <monetary.h>
13552 #include <stdio.h>
13553 ...
13554 struct tblfmt {
13555     char *format;
13556     char *description;
13557 };
13558 struct tblfmt table[] = {
13559     { "%n", "default formatting" },
13560     { "%11n", "right align within an 11 character field" },
13561     { "%#5n", "aligned columns for values up to 99,999" },
13562     { "%=*#5n", "specify a fill character" },
13563     { "%=0#5n", "fill characters do not use grouping" },
13564     { "%^#5n", "disable the grouping separator" },
13565     { "%^#5.0n", "round off to whole units" },
```

```

13566         { "%^#5.4n", "increase the precision" },
13567         { "%(#5n", "use an alternative pos/neg style" },
13568         { "%!(#5n", "disable the currency symbol" },
13569     };
13570     ...
13571     float input[3];
13572     int i, j;
13573     char convbuf[100];
13574     ...
13575     strfmon(convbuf, sizeof(convbuf), table[i].format, input[j]);
13576
13577     if (j == 0) {
13578         printf("%s%s%s\n", table[i].format,
13579             convbuf, table[i].description);
13580     }
13581     else {
13582         printf("%s\n", convbuf);
13583     }
13584     ...

```

13584 APPLICATION USAGE

13585 If the application calling *fprintf()* has any objects of type **wint_t** or **wchar_t**, it must also include
 13586 the **<wchar.h>** header to have these objects defined.

13587 RATIONALE

13588 None.

13589 FUTURE DIRECTIONS

13590 None.

13591 SEE ALSO

13592 *fputc()*, *fscanf()*, *setlocale()*, *wcrtomb()*, the Base Definitions volume of IEEE Std. 1003.1-200x,
 13593 **<stdio.h>**, **<wchar.h>**, the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

13594 CHANGE HISTORY

13595 First released in Issue 1. Derived from Issue 1 of the SVID.

13596 Issue 4

13597 In the DESCRIPTION, references to *langinfo* data are marked as extensions. The reference to
 13598 *langinfo* data is removed from the description of the radix character.

13599 The ' ' (single-quote) flag is added to the list of flag characters and marked as an extension.
 13600 This flag directs that numeric conversion is formatted with the decimal grouping character.

13601 The detailed description of these functions is provided here instead of under *printf()*.

13602 The information in the APPLICATION USAGE section is moved to the DESCRIPTION. A new
 13603 APPLICATION USAGE section is added.

13604 The [EILSEQ] error is added to the ERRORS section and all errors are marked as extensions.

13605 The following changes are incorporated for alignment with the ISO C standard:

- 13606 • The type of the *format* arguments is changed from **char*** to **const char***.
- 13607 • The DESCRIPTION is reworded or presented differently in a number of places for alignment
 13608 with the ISO C standard, and also for clarity. There are no functional changes, except as
 13609 noted elsewhere in this CHANGE HISTORY section.

- 13610 The following changes are incorporated for alignment with the MSE working draft:
- 13611 • The *C* and *S* conversion characters are added, indicating respectively a wide character of type
 - 13612 `wchar_t` and pointer to a wide-character string of type `wchar_t*` in the argument list.
- 13613 **Issue 4, Version 2**
- 13614 The [ENOMEM] error is added to the ERRORS section as an optional error.
- 13615 **Issue 5**
- 13616 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the *l* (ell) qualifier can
- 13617 now be used with *c* and *s* conversion characters.
- 13618 The *snprintf()* function is new in Issue 5.
- 13619 **Issue 6**
- 13620 Extensions beyond the ISO C standard are now marked.
- 13621 The description of *snprintf()* has been aligned with The Open Group Base Resolution bwg98-006.
- 13622 This aligns *snprintf()* with historic BSD behavior.
- 13623 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 13624 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
- 13625 • The prototypes for *fprintf()*, *printf()*, *snprintf()*, and *sprintf()* are updated, and the XSI
 - 13626 shading is removed from *snprintf()*.
 - 13627 • The DESCRIPTION is updated.

13628 **NAME**

13629 fputc — put a byte on a stream

13630 **SYNOPSIS**

13631 #include <stdio.h>

13632 int fputc(int *c*, FILE **stream*);13633 **DESCRIPTION**

13634 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 13635 conflict between the requirements described here and the ISO C standard is unintentional. This
 13636 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

13637 The *fputc()* function shall write the byte specified by *c* (converted to an **unsigned char**) to the
 13638 output stream pointed to by *stream*, at the position indicated by the associated file-position
 13639 indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot
 13640 support positioning requests, or if the stream was opened with append mode, the byte shall be
 13641 appended to the output stream.

13642 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
 13643 execution of *fputc()* and the next successful completion of a call to *fflush()* or *fclose()* on the same
 13644 stream or a call to *exit()* or *abort()*.

13645 **RETURN VALUE**

13646 Upon successful completion, *fputc()* shall return the value it has written. Otherwise, it shall
 13647 CX return EOF, the error indicator for the stream shall be set, and *errno* shall be set to indicate the
 13648 error.

13649 **ERRORS**

13650 The *fputc()* function shall fail if either the *stream* is unbuffered or the *stream*'s buffer needs to be
 13651 flushed, and:

13652 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 13653 process would be delayed in the write operation.

13654 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
 13655 writing.

13656 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size.

13657 XSI [EFBIG] An attempt was made to write to a file that exceeds the process' file size limit.

13658 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 13659 offset maximum.

13660 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
 13661 was transferred.

13662 CX [EIO] A physical I/O error has occurred, or the process is a member of a
 13663 background process group attempting to write to its controlling terminal,
 13664 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the
 13665 process group of the process is orphaned. This error may also be returned
 13666 under implementation-defined conditions.

13667 CX [ENOSPC] There was no free space remaining on the device containing the file.

13668 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 13669 any process. A SIGPIPE signal shall also be sent to the thread.

13670 The *fputc()* function may fail if:

- 13671 CX [ENOMEM] Insufficient storage space is available.
- 13672 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
13673 capabilities of the device.
- 13674 **EXAMPLES**
- 13675 None.
- 13676 **APPLICATION USAGE**
- 13677 None.
- 13678 **RATIONALE**
- 13679 None.
- 13680 **FUTURE DIRECTIONS**
- 13681 None.
- 13682 **SEE ALSO**
- 13683 *ferror()*, *fopen()*, *getrlimit()*, *putc()*, *puts()*, *setbuf()*, *ulimit()*, the Base Definitions volume of
13684 IEEE Std. 1003.1-200x, <stdio.h>
- 13685 **CHANGE HISTORY**
- 13686 First released in Issue 1. Derived from Issue 1 of the SVID.
- 13687 **Issue 4**
- 13688 In the DESCRIPTION, the text is changed to make it clear that the function writes byte values,
13689 rather than (possibly multi-byte) character values.
- 13690 In the ERRORS section, text is added to indicate that error returns are only generated when
13691 either the stream is unbuffered, or if the stream buffer needs to be flushed.
- 13692 Also in the ERRORS section, in previous issues generation of the [EIO] error depended on
13693 whether or not an implementation supported Job Control. This functionality is now defined as
13694 mandatory.
- 13695 The [ENXIO] error is moved to the list of optional errors, and all the optional errors are marked
13696 as extensions.
- 13697 The description of [EINTR] is amended.
- 13698 The [EFBIG] error is marked to show extensions.
- 13699 **Issue 4, Version 2**
- 13700 In the ERRORS section, the description of [EIO] is updated to include the case where a physical
13701 I/O error occurs.
- 13702 **Issue 5**
- 13703 Large File Summit extensions are added.
- 13704 **Issue 6**
- 13705 Extensions beyond the ISO C standard are now marked.
- 13706 The following new requirements on POSIX implementations derive from alignment with the
13707 Single UNIX Specification:
- 13708
 - The [EIO] and [EFBIG] mandatory error conditions are added.
- 13709
 - The [ENOMEM] and [ENXIO] optional error conditions are added.

13710 **NAME**

13711 fputs — put a string on a stream

13712 **SYNOPSIS**

13713 #include <stdio.h>

13714 int fputs(const char *restrict *s*, FILE *restrict *stream*);13715 **DESCRIPTION**

13716 CX The functionality described on this reference page is aligned with the ISO C standard. Any
13717 conflict between the requirements described here and the ISO C standard is unintentional. This
13718 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

13719 The *fputs()* function shall write the null-terminated string pointed to by *s* to the stream pointed
13720 to by *stream*. The terminating null byte shall not be written.

13721 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
13722 execution of *fputs()* and the next successful completion of a call to *fflush()* or *fclose()* on the same
13723 stream or a call to *exit()* or *abort()*.

13724 **RETURN VALUE**

13725 Upon successful completion, *fputs()* shall return a non-negative number. Otherwise, it shall
13726 CX return EOF, set an error indicator for the stream, and set *errno* to indicate the error.

13727 **ERRORS**13728 Refer to *fputc()*.13729 **EXAMPLES**13730 **Printing to Standard Output**

13731 The following example gets the current time, converts it to a string using *localtime()* and
13732 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to
13733 an event for which it is waiting.

```
13734 #include <time.h>
13735 #include <stdio.h>
13736 ...
13737 time_t now;
13738 int minutes_to_event;
13739 ...
13740 time(&now);
13741 printf("The time is ");
13742 fputs(asctime(localtime(&now)), stdout);
13743 printf("There are still %d minutes to the event.\n",
13744       minutes_to_event);
13745 ...
```

13746 **APPLICATION USAGE**13747 The *puts()* function appends a <newline> character while *fputs()* does not.13748 **RATIONALE**

13749 None.

13750 **FUTURE DIRECTIONS**

13751 None.

13752 **SEE ALSO**

13753 *fopen()*, *putc()*, *puts()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>

13754 **CHANGE HISTORY**

13755 First released in Issue 1. Derived from Issue 1 of the SVID.

13756 **Issue 4**

13757 In the DESCRIPTION, the words “null character” are replaced by “null byte”, to make it clear
13758 that this function deals solely in byte values.

13759 The following change is incorporated for alignment with the ISO C standard:

- 13760 • The type of argument *s* is changed from **char*** to **const char***.

13761 **Issue 6**

13762 Extensions beyond the ISO C standard are now marked.

13763 The *fputs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

13764 NAME

13765 fputcw — put a wide-character code on a stream

13766 SYNOPSIS

13767 #include <stdio.h>

13768 #include <wchar.h>

13769 wint_t fputcw(wchar_t *wc*, FILE **stream*);

13770 DESCRIPTION

13771 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 13772 conflict between the requirements described here and the ISO C standard is unintentional. This
 13773 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

13774 The *fputcw()* function shall write the character corresponding to the wide-character code *wc* to
 13775 the output stream pointed to by *stream*, at the position indicated by the associated file-position
 13776 indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot
 13777 support positioning requests, or if the stream was opened with append mode, the character is
 13778 appended to the output stream. If an error occurs while writing the character, the shift state of
 13779 the output file is left in an undefined state.

13780 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
 13781 execution of *fputcw()* and the next successful completion of a call to *fflush()* or *fclose()* on the
 13782 same stream or a call to *exit()* or *abort()*.

13783 RETURN VALUE

13784 Upon successful completion, *fputcw()* shall return *wc*. Otherwise, it shall return WEOF, the error
 13785 CX indicator for the stream shall be set set, and *errno* shall be set to indicate the error.

13786 ERRORS

13787 The *fputcw()* function shall fail if either the stream is unbuffered or data in the *stream*'s buffer
 13788 needs to be written, and:

13789 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 13790 process would be delayed in the write operation.

13791 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
 13792 writing.

13793 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size or
 13794 the process' file size limit.

13795 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 13796 offset maximum associated with the corresponding stream.

13797 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
 13798 was transferred.

13799 CX [EIO] A physical I/O error has occurred, or the process is a member of a
 13800 background process group attempting to write to its controlling terminal,
 13801 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the
 13802 process group of the process is orphaned. This error may also be returned
 13803 under implementation-defined conditions.

13804 CX [ENOSPC] There was no free space remaining on the device containing the file.

13805 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 13806 any process. A SIGPIPE signal shall also be sent to the thread.

13807 The *fputwc()* function may fail if:

13808 CX [ENOMEM] Insufficient storage space is available.

13809 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
13810 capabilities of the device.

13811 CX [EILSEQ] The wide-character code *wc* does not correspond to a valid character.

13812 EXAMPLES

13813 None.

13814 APPLICATION USAGE

13815 None.

13816 RATIONALE

13817 None.

13818 FUTURE DIRECTIONS

13819 None.

13820 SEE ALSO

13821 *ferror()*, *fopen()*, *setbuf()*, *ulimit()*, the Base Definitions volume of IEEE Std. 1003.1-200x,
13822 `<stdio.h>`, `<wchar.h>`

13823 CHANGE HISTORY

13824 First released in Issue 4. Derived from the MSE working draft.

13825 Issue 4, Version 2

13826 In the ERRORS section, the description of [EIO] is updated to include the case where a physical
13827 I/O error occurs.

13828 Issue 5

13829 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*
13830 is changed from `wint_t` to `wchar_t`.

13831 The Optional Header (OH) marking is removed from `<stdio.h>`.

13832 Large File Summit extensions are added.

13833 Issue 6

13834 Extensions beyond the ISO C standard are now marked.

13835 The following new requirements on POSIX implementations derive from alignment with the
13836 Single UNIX Specification:

- 13837 • The [EFBIG] and [EIO] mandatory error conditions are added.

13838 **NAME**

13839 fputws — put a wide-character string on a stream

13840 **SYNOPSIS**

13841 #include <stdio.h>

13842 #include <wchar.h>

13843 int fputws(const wchar_t *restrict ws, FILE *restrict stream);

13844 **DESCRIPTION**

13845 CX The functionality described on this reference page is aligned with the ISO C standard. Any
13846 conflict between the requirements described here and the ISO C standard is unintentional. This
13847 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

13848 The *fputws()* function shall write a character string corresponding to the (null-terminated)
13849 wide-character string pointed to by *ws* to the stream pointed to by *stream*. No character
13850 corresponding to the terminating null wide-character code shall be written.

13851 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
13852 execution of *fputws()* and the next successful completion of a call to *fflush()* or *fclose()* on the
13853 same stream or a call to *exit()* or *abort()*.

13854 **RETURN VALUE**

13855 Upon successful completion, *fputws()* shall return a non-negative number. Otherwise, it shall
13856 CX return -1 , set an error indicator for the stream, and set *errno* to indicate the error.

13857 **ERRORS**13858 Refer to *fputwc()*.13859 **EXAMPLES**

13860 None.

13861 **APPLICATION USAGE**13862 The *fputws()* function does not append a <newline> character.13863 **RATIONALE**

13864 None.

13865 **FUTURE DIRECTIONS**

13866 None.

13867 **SEE ALSO**13868 *fopen()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>, <wchar.h>13869 **CHANGE HISTORY**

13870 First released in Issue 4. Derived from the MSE working draft.

13871 **Issue 5**

13872 The Optional Header (OH) marking is removed from <stdio.h>.

13873 **Issue 6**

13874 Extensions beyond the ISO C standard are now marked.

13875 The *fputws()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

13876 **NAME**

13877 fread — binary input

13878 **SYNOPSIS**

13879 #include <stdio.h>

```
13880 size_t fread(void *restrict ptr, size_t size, size_t nitems,
13881 FILE *restrict stream);
```

13882 **DESCRIPTION**

13883 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 13884 conflict between the requirements described here and the ISO C standard is unintentional. This
 13885 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

13886 The *fread()* function shall read into the array pointed to by *ptr* up to *nitems* members whose size
 13887 is specified by *size* in bytes, from the stream pointed to by *stream*. For each object, size calls are
 13888 made to the *fgetc()* function and the results stored, in the order read, in an array of **unsigned**
 13889 **char** exactly overlaying the object. The file position indicator for the stream (if defined) is
 13890 advanced by the number of bytes successfully read. If an error occurs, the resulting value of the
 13891 file position indicator for the stream is indeterminate. If a partial member is read, its value is
 13892 indeterminate.

13893 CX The *fread()* function may mark the *st_atime* field of the file associated with *stream* for update. The
 13894 *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
 13895 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
 13896 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

13897 **RETURN VALUE**

13898 Upon successful completion, *fread()* shall return the number of members successfully read
 13899 which is less than *nitems* only if a read error or end-of-file is encountered. If *size* or *nitems* is 0,
 13900 *fread()* shall return 0 and the contents of the array and the state of the stream remain unchanged.
 13901 CX Otherwise, if a read error occurs, the error indicator for the stream shall be set, and *errno* shall be
 13902 set to indicate the error.

13903 **ERRORS**13904 Refer to *fgetc()*.13905 **EXAMPLES**13906 **Reading from a Stream**13907 The following example reads a single element from the *fp* stream into the array pointed to by *buf*.

```
13908 #include <stdio.h>
13909 ...
13910 size_t bytes_read;
13911 char buf[100];
13912 FILE *fp;
13913 ...
13914 bytes_read = fread(buf, sizeof(buf), 1, fp);
13915 ...
```

13916 **APPLICATION USAGE**

13917 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an
 13918 end-of-file condition.

13919 Because of possible differences in member length and byte ordering, files written using *fwrite()*
 13920 are application-dependent, and possibly cannot be read using *fread()* by a different application

13921 or by the same application on a different processor.

13922 **RATIONALE**

13923 None.

13924 **FUTURE DIRECTIONS**

13925 None.

13926 **SEE ALSO**

13927 *feof()*, *ferror()*, *fgetc()*, *fopen()*, *getc()*, *gets()*, *scanf()*, the Base Definitions volume of
13928 IEEE Std. 1003.1-200x, `<stdio.h>`

13929 **CHANGE HISTORY**

13930 First released in Issue 1. Derived from Issue 1 of the SVID.

13931 **Issue 4**

13932 The list of functions that may cause the *st_atime* field to be updated is revised.

13933 The following change is incorporated for alignment with the ISO C standard:

- 13934
- In the RETURN VALUE section, the behavior if *size* or *nitems* is 0 is defined.

13935 **Issue 6**

13936 Extensions beyond the ISO C standard are now marked.

13937 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 13938
- The *fread()* prototype is updated.
- 13939
- The DESCRIPTION is updated to describe how the bytes from a call to *fgetc()* are stored.

13940 **NAME**

13941 free — free allocated memory

13942 **SYNOPSIS**

13943 #include <stdlib.h>

13944 void free(void *ptr);

13945 **DESCRIPTION**

13946 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 13947 conflict between the requirements described here and the ISO C standard is unintentional. This
 13948 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

13949 The *free()* function shall cause the space pointed to by *ptr* to be deallocated; that is, made
 13950 available for further allocation. If *ptr* is a null pointer, no action shall occur. Otherwise, if the
 13951 argument does not match a pointer earlier returned by the *calloc()*, *malloc()*, *posix_memalign()*, or
 13952 *realloc()* function, or if the space is deallocated by a call to *free()* or *realloc()*, the behavior is
 13953 undefined.

13954 Any use of a pointer that refers to freed space results in undefined behavior.

13955 **RETURN VALUE**13956 The *free()* function shall return no value.13957 **ERRORS**

13958 No errors are defined.

13959 **EXAMPLES**

13960 None.

13961 **APPLICATION USAGE**

13962 There is now no requirement for the implementation to support the inclusion of <malloc.h>.

13963 **RATIONALE**

13964 None.

13965 **FUTURE DIRECTIONS**

13966 None.

13967 **SEE ALSO**13968 *calloc()*, *malloc()*, *realloc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>13969 **CHANGE HISTORY**

13970 First released in Issue 1. Derived from Issue 1 of the SVID.

13971 **Issue 4**

13972 The APPLICATION USAGE section is changed to record that <malloc.h> need no longer be
 13973 supported on XSI-conformant systems.

13974 The following change is incorporated for alignment with the ISO C standard:

- 13975 • The DESCRIPTION now states that the behavior is undefined if any use is made of a pointer
 13976 that refers to freed space. This was implied but not stated explicitly in Issue 3.

13977 **Issue 4, Version 2**

13978 The DESCRIPTION is updated for X/OPEN UNIX conformance to indicate that the *free()*
 13979 function can also be used to free memory allocated by *valloc()*.

13980 **Issue 6**

13981 Reference to the *valloc()* function is removed.

13982 NAME

13983 freeaddrinfo, getaddrinfo — get address information

13984 SYNOPSIS

13985 #include <sys/socket.h>

13986 #include <netdb.h>

13987 void freeaddrinfo(struct addrinfo *ai);

13988 int getaddrinfo(const char *restrict nodename, const char *restrict servname, |

13989 const struct addrinfo *restrict hints, struct addrinfo **restrict res); |

13990 DESCRIPTION

13991 The *freeaddrinfo()* function frees one or more **addrinfo** structures returned by *getaddrinfo()*, along
 13992 with any additional storage associated with those structures. If the *ai_next* field of the structure
 13993 is not null, the entire list of structures is freed. The *freeaddrinfo()* function shall support the
 13994 freeing of arbitrary sublists of an **addrinfo** list originally returned by *getaddrinfo()*.

13995 The *getaddrinfo()* function shall translate the name of a service location (for example, a host
 13996 name) and/or a service name and shall return a set of socket addresses and associated
 13997 information to be used in creating a socket with which to address the specified service.

13998 The *freeaddrinfo()* and *getaddrinfo()* functions shall be thread-safe.

13999 The *nodename* and *servname* arguments are either null pointers or pointers to null-terminated
 14000 strings. One or both of these two arguments shall be supplied by the application as a non-null
 14001 pointer.

14002 The format of a valid name depends on the protocol family or families. If a specific family is not
 14003 given and the name could be interpreted as valid within multiple supported families, the
 14004 implementation shall attempt to resolve the name in all supported families and, in absence of
 14005 errors, one or more results shall be returned.

14006 If the *nodename* argument is not null, it can be a descriptive name or can be an address string. If
 14007 IP6 the specified address family is AF_INET, AF_INET6, or AF_UNSPEC, valid descriptive names
 14008 include host names. If the specified address family is AF_INET or AF_UNSPEC, address strings
 14009 using Internet standard dot notation as specified in *inet_addr()* are valid.

14010 IP6 If the specified address family is AF_INET6 or AF_UNSPEC, standard IPv6 text forms described
 14011 in *inet_ntop()* are valid.

14012 If *nodename* is not null, the requested service location is named by *nodename*; otherwise, the
 14013 requested service location is local to the caller.

14014 If *servname* is null, the call shall return network-level addresses for the specified *nodename*. If
 14015 *servname* is not null, it is a null-terminated character string identifying the requested service. This
 14016 can be either a descriptive name or a numeric representation suitable for use with the address
 14017 IP6 family or families. If the specified address family is AF_INET, AF_INET6, or AF_UNSPEC, the
 14018 service can be specified as a string specifying a decimal port number.

14019 If the *hints* argument is not null, it refers to a structure containing input values that may direct
 14020 the operation by providing options and by limiting the returned information to a specific socket
 14021 type, address family and/or protocol. In this *hints* structure every member other than *ai_flags*,
 14022 *ai_family*, *ai_socktype*, and *ai_protocol* shall be set to zero or a null pointer. A value of
 14023 AF_UNSPEC for *ai_family* means that the caller shall accept any protocol family. A value of zero
 14024 for *ai_socktype* means that the caller shall accept any socket type. A value of zero for *ai_protocol*
 14025 means that the caller shall accept any protocol. If *hints* is a null pointer, the behavior shall be as if
 14026 it referred to a structure containing the value zero for the *ai_flags*, *ai_socktype*, and *ai_protocol*
 14027 fields, and AF_UNSPEC for the *ai_family* field.

14028 **Notes:**

- 14029 1. If the caller handles only TCP and not UDP, for example, then the *ai_protocol*
 14030 member of the *hints* structure should be set to IPPROTO_TCP when
 14031 *getaddrinfo()* is called.
- 14032 2. If the caller handles only IPv4 and not IPv6, then the *ai_family* member of the
 14033 *hints* structure should be set to PF_INET when *getaddrinfo()* is called.

14034 The *ai_flags* field to which the *hints* parameter points shall be set to zero or be the bitwise-
 14035 inclusive OR of one or more of the values AI_PASSIVE, AI_CANONNAME, and
 14036 AI_NUMERICHOST.

14037 If the AI_PASSIVE flag is specified, the returned address information shall be suitable for use in
 14038 binding a socket for accepting incoming connections for the specified service. In this case, if the
 14039 *nodename* argument is null, then the IP address portion of the socket address structure shall be
 14040 set to INADDR_ANY for an IPv4 address or IN6ADDR_ANY_INIT for an IPv6 address. If the
 14041 AI_PASSIVE flag is not specified, the returned address information shall be suitable for a call to
 14042 *connect()* (for a connection-mode protocol) or for a call to *connect()*, *sendto()*, or *sendmsg()* (for a
 14043 connectionless protocol). In this case, if the *nodename* argument is null, then the IP address
 14044 portion of the socket address structure shall be set to the loopback address.

14045 If the AI_CANONNAME flag is specified and the *nodename* argument is not null, the function
 14046 attempts to determine the canonical name corresponding to *nodename* (for example, if *nodename*
 14047 is an alias or shorthand notation for a complete name).

14048 If the AI_NUMERICHOST flag is specified, then a non-null *nodename* string supplied shall be a
 14049 numeric host address string. Otherwise, an [EAI_NONAME] error is returned. This flag prevents
 14050 any type of name resolution service (for example, the DNS) from being invoked.

14051 If the AI_NUMERICSERV flag is specified, then a non-null *servname* string supplied shall be a
 14052 numeric port string. Otherwise, an [EAI_NONAME] error is returned. This flag prevents any
 14053 type of name resolution service (for example, NIS+) from being invoked.

14054 The *ai_socktype* field to which argument *hints* points specifies the socket type for the service, as
 14055 defined in *socket()*. If a specific socket type is not given (for example, a value of zero) and the
 14056 service name could be interpreted as valid with multiple supported socket types, the
 14057 implementation shall attempt to resolve the service name for all supported socket types and, in
 14058 the absence of errors, all possible results shall be returned. A non-zero socket type value shall
 14059 limit the returned information to values with the specified socket type.

14060 If the *ai_family* field to which *hints* points has the value AF_UNSPEC, addresses are returned for
 14061 use with any protocol family that can be used with the specified *nodename* and/or *servname*.
 14062 Otherwise, addresses are returned for use only with the specified protocol family. If *ai_family* is
 14063 not AF_UNSPEC and *ai_protocol* is not zero, then addresses are returned for use only with the
 14064 specified protocol family and protocol; the value of *ai_protocol* is interpreted as in a call to the
 14065 *socket()* function with the corresponding values of *ai_family* and *ai_protocol*.

14066 **RETURN VALUE**

14067 A zero return value for *getaddrinfo()* indicates successful completion; a non-zero return value
 14068 indicates failure. The possible values for the failures are listed in the ERRORS section.

14069 Upon successful return of *getaddrinfo()*, the location to which *res* points refers to a linked list of
 14070 **addrinfo** structures, each of which specifies a socket address and information for use in creating
 14071 a socket with which to use that socket address. The list shall include at least one **addrinfo**
 14072 structure. The *ai_next* field of each structure contains a pointer to the next structure on the list, or
 14073 a null pointer if it is the last structure on the list. Each structure on the list includes values for use
 14074 with a call to the *socket()* function, and a socket address for use with the *connect()* function or, if

14075 the AI_PASSIVE flag was specified, for use with the *bind()* function. The fields *ai_family*,
 14076 *ai_socktype*, and *ai_protocol* are usable as the arguments to the *socket()* function to create a socket
 14077 suitable for use with the returned address. The fields *ai_addr* and *ai_addrlen* are usable as the
 14078 arguments to the *connect()* or *bind()* functions with such a socket, according to the AI_PASSIVE
 14079 flag.

14080 If *nodename* is not null, and if requested by the AI_CANONNAME flag, the *ai_canonname* field of
 14081 the first returned **addrinfo** structure points to a null-terminated string containing the canonical
 14082 name corresponding to the input *nodename*; if the canonical name is not available, then
 14083 *ai_canonname* refers to the *nodename* argument or a string with the same contents. The contents of
 14084 the *ai_flags* field of the returned structures are undefined.

14085 All fields in socket address structures returned by *getaddrinfo()* that are not filled in through an
 14086 explicit argument (for example, *sin6_flowinfo* and *sin_zero*) shall be set to zero.

14087 **Note:** This makes it easier to compare socket address structures.

14088 ERRORS

14089 The *getaddrinfo()* function shall fail and return the corresponding value if:

14090 [EAI_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

14091 [EAI_BADFLAGS]

14092 The *flags* parameter had an invalid value.

14093 [EAI_FAIL] A non-recoverable error occurred when attempting to resolve the name.

14094 [EAI_FAMILY] The address family was not recognized.

14095 [EAI_MEMORY] There was a memory allocation failure when trying to allocate storage for the
 14096 return value.

14097 [EAI_NONAME] The name does not resolve for the supplied parameters.

14098 Neither *nodename* nor *servname* were supplied. At least one of these shall be
 14099 supplied.

14100 [EAI_SERVICE] The service passed was not recognized for the specified socket type.

14101 [EAI_SOCKTYPE]

14102 The intended socket type was not recognized.

14103 [EAI_SYSTEM] A system error occurred; the error code can be found in *errno*.

14104 EXAMPLES

14105 None.

14106 APPLICATION USAGE

14107 None.

14108 RATIONALE

14109 None.

14110 FUTURE DIRECTIONS

14111 None.

14112 SEE ALSO

14113 *connect()*, *gethostbyname()*, *getipnodebyname()*, *getnameinfo()*, *getservbyname()*, *socket()*, the Base
 14114 Definitions volume of IEEE Std. 1003.1-200x, <**netdb.h**>, <**sys/socket.h**>

14115 **CHANGE HISTORY**

- 14116 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
- 14117 The **restrict** keyword is added to the *getaddrinfo()* prototype for alignment with the
- 14118 ISO/IEC 9899:1999 standard.

14119 **NAME**

14120 freehostent — network host database functions

14121 **SYNOPSIS**

14122 #include <netdb.h>

14123 void freehostent(struct hostent *ptr);

14124 **DESCRIPTION**

14125 Refer to *endhostent()*.

14126 NAME

14127 freopen — open a stream

14128 SYNOPSIS

14129 #include <stdio.h>

14130 FILE *freopen(const char *restrict *filename*, const char *restrict *mode*,
14131 FILE *restrict *stream*);

14132 DESCRIPTION

14133 CX The functionality described on this reference page is aligned with the ISO C standard. Any
14134 conflict between the requirements described here and the ISO C standard is unintentional. This
14135 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.14136 The *freopen()* function shall first attempt to flush the stream and close any file descriptor
14137 associated with *stream*. Failure to flush or close the file successfully shall be ignored. The error
14138 and end-of-file indicators for the stream shall be cleared.14139 The *freopen()* function shall open the file whose path name is the string pointed to by *filename*
14140 and associate the stream pointed to by *stream* with it. The *mode* argument shall be used just as in
14141 *fopen()*.

14142 The original stream shall be closed regardless of whether the subsequent open succeeds.

14143 If *filename* is a null pointer, the *freopen()* function shall attempt to change the mode of the stream
14144 to that specified by *mode*, as if the name of the file currently associated with the stream had been
14145 used. It is implementation-defined which changes of mode are permitted (if any), and under
14146 what circumstances.14147 XSI After a successful call to the *freopen()* function, the orientation of the stream shall be cleared, the
14148 encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an
14149 initial conversion state.14150 CX The largest value that can be represented correctly in an object of type **off_t** shall be established
14151 as the offset maximum in the open file description.

14152 RETURN VALUE

14153 Upon successful completion, *freopen()* shall return the value of *stream*. Otherwise, a null pointer
14154 CX shall be returned, and *errno* shall be set to indicate the error.

14155 ERRORS

14156 The *freopen()* function shall fail if:14157 CX [EACCES] Search permission is denied on a component of the path prefix, or the file
14158 exists and the permissions specified by *mode* are denied, or the file does not
14159 exist and write permission is denied for the parent directory of the file to be
14160 created.14161 CX [EINTR] A signal was caught during *freopen()*.14162 CX [EISDIR] The named file is a directory and *mode* requires write access.14163 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
14164 argument.

14165 CX [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

14166 CX [ENAMETOOLONG]

14167 The length of the *filename* argument exceeds {PATH_MAX} or a path name
14168 component is longer than {NAME_MAX}.

14169 CX	[ENFILE]	The maximum allowable number of files is currently open in the system.
14170 CX 14171	[ENOENT]	A component of <i>filename</i> does not name an existing file or <i>filename</i> is an empty string.
14172 CX 14173	[ENOSPC]	The directory or file system that would contain the new file cannot be expanded, the file does not exist, and it was to be created.
14174 CX	[ENOTDIR]	A component of the path prefix is not a directory.
14175 CX 14176	[ENXIO]	The named file is a character special or block special file, and the device associated with this special file does not exist.
14177 CX 14178	[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented correctly in an object of type <code>off_t</code> .
14179 CX 14180	[EROFS]	The named file resides on a read-only file system and <i>mode</i> requires write access.
14181		The <i>freopen()</i> function may fail if:
14182 CX	[EINVAL]	The value of the <i>mode</i> argument is not valid.
14183 CX 14184	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
14185 CX 14186 14187	[ENAMETOOLONG]	Path name resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.
14188 CX	[ENOMEM]	Insufficient storage space is available.
14189 CX 14190	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
14191 CX 14192	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>mode</i> requires write access.

14193 EXAMPLES

14194 Directing Standard Output to a File

14195 The following example logs all standard output to the `/tmp/logfile` file.

```
14196 #include <stdio.h>
14197 ...
14198 FILE *fp;
14199 ...
14200 fp = freopen ("/tmp/logfile", "a+", stdout);
14201 ...
```

14202 APPLICATION USAGE

14203 The *freopen()* function is typically used to attach the preopened *streams* associated with *stdin*,
14204 *stdout*, and *stderr* to other files.

14205 RATIONALE

14206 None.

14207 **FUTURE DIRECTIONS**

14208 None.

14209 **SEE ALSO**14210 *fclose()*, *fopen()*, *fdopen()*, *mbsinit()*, the Base Definitions volume of IEEE Std. 1003.1-200x,
14211 `<stdio.h>`14212 **CHANGE HISTORY**

14213 First released in Issue 1. Derived from Issue 1 of the SVID.

14214 **Issue 4**14215 In the DESCRIPTION, the word “name” is replaced by “path name”, to make it clear that the
14216 function is not limited to accepting file names only.

14217 In the ERRORS section:

- 14218 • The description of the [EMFILE] error has been changed to refer to {OPEN_MAX} file
- 14219 descriptors rather than {FOPEN_MAX} file descriptors, directories, and message catalogs.
- 14220 • The errors [EINVAL], [ENOMEM], and [ETXTBSY] are marked as extensions.
- 14221 • The [ENXIO] error is added in the “may fail” section and marked as an extension.

14222 The following change is incorporated for alignment with the ISO C standard:

- 14223 • The type of arguments *filename* and *mode* are changed from **char*** to **const char***.

14224 The following change is incorporated for alignment with the FIPS requirements:

- 14225 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path
- 14226 name component is larger than {NAME_MAX} is now defined as mandatory and marked as
- 14227 an extension.

14228 **Issue 4, Version 2**

14229 The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- 14230 • It states that [ELOOP] is returned if too many symbolic links are encountered during path
- 14231 name resolution.
- 14232 • A second [ENAMETOOLONG] condition is defined that may report excessive length of an
- 14233 intermediate result of path name resolution of a symbolic link.

14234 **Issue 5**14235 The DESCRIPTION is updated to indicate that the orientation of the stream is cleared and the
14236 conversion state of the stream is set to an initial conversion state by a successful call to the
14237 *freopen()* function.

14238 Large File Summit extensions are added.

14239 **Issue 6**

14240 Extensions beyond the ISO C standard are now marked.

14241 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 14242 • The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.
- 14243 This is since behavior may vary from one file system to another.

14244 The following new requirements on POSIX implementations derive from alignment with the
14245 Single UNIX Specification:

- 14246 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file
- 14247 description. This change is to support large files.

- 14248 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support
- 14249 large files.
- 14250 • The [ELOOP] mandatory error condition is added.
- 14251 • A second [ENAMETOOLONG] is added as an optional error condition.
- 14252 • The [EINVAL], [ENOMEM], [ENXIO], and [ETXTBSY] optional error conditions are added.
- 14253 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
- 14254 • The *freopen()* prototype is updated.
- 14255 • The DESCRIPTION is updated.
- 14256 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
- 14257 [ELOOP] error condition is added.

14258 **NAME**

14259 frexp, frexpf, frexpl — extract mantissa and exponent from a double precision number

14260 **SYNOPSIS**

14261 #include <math.h>

14262 double frexp(double *num*, int **exp*);

14263 float frexpf(float *value*, int **exp*);

14264 long double frexpl(long double *value*, int **exp*);

14265 **DESCRIPTION**

14266 CX The functionality described on this reference page is aligned with the ISO C standard. Any
14267 conflict between the requirements described here and the ISO C standard is unintentional. This
14268 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

14269 These functions breaks a floating-point number into a normalized fraction and an integral power
14270 of 2. It stores the integer exponent in the **int** object pointed to by *exp*.

14271 An application wishing to check for error situations should set *errno* to 0 before calling *frexp()*. If
14272 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

14273 **RETURN VALUE**

14274 These functions shall return the value *x*, such that *x* has a magnitude in the interval $[\frac{1}{2}, 1)$ or 0,
14275 and *num* equals *x* times 2 raised to the power **exp*.

14276 If *num* is 0, both parts of the result shall be 0.

14277 XSI If *num* is NaN, NaN shall be returned, *errno* may be set to [EDOM], and the value of **exp* shall be
14278 unspecified.

14279 If *num* is $\pm\text{Inf}$, *num* shall be returned, *errno* may be set to [EDOM], and the value of **exp* shall be
14280 unspecified.

14281 **ERRORS**

14282 These functions may fail if:

14283 XSI [EDOM] The value of *num* is NaN or $\pm\text{Inf}$.

14284 XSI No other errors shall occur.

14285 **EXAMPLES**

14286 None.

14287 **APPLICATION USAGE**

14288 None.

14289 **RATIONALE**

14290 None.

14291 **FUTURE DIRECTIONS**

14292 None.

14293 **SEE ALSO**

14294 *isnan()*, *ldexp()*, *modf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

14295 **CHANGE HISTORY**

14296 First released in Issue 1. Derived from Issue 1 of the SVID.

14297 **Issue 4**

14298 References to *matherr()* are removed.

14299 The name of the first argument is changed from *value* to *num*.

14300 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the
14301 ISO C standard and to rationalize error handling in the mathematics functions.

14302 The return value specified for [EDOM] is marked as an extension.

14303 **Issue 5**

14304 The DESCRIPTION is updated to indicate how an application should check for an error. This
14305 text was previously published in the APPLICATION USAGE section.

14306 **Issue 6**

14307 The *frexpf()* and *frexpl()* functions are added for alignment with the ISO/IEC 9899:1999
14308 standard.

14309 NAME

14310 fscanf, scanf, sscanf — convert formatted input

14311 SYNOPSIS

14312 #include <stdio.h>

14313 int fscanf(FILE *restrict *stream*, const char *restrict *format*, ...);14314 int scanf(const char *restrict *format*, ...);14315 int sscanf(const char *restrict *s*, const char *restrict *format*, ...);

14316 DESCRIPTION

14317 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 14318 conflict between the requirements described here and the ISO C standard is unintentional. This
 14319 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

14320 The *fscanf()* function reads from the named input *stream*. The *scanf()* function reads from the
 14321 standard input stream *stdin*. The *sscanf()* function reads from the string *s*. Each function reads
 14322 bytes, interprets them according to a format, and stores the results in its arguments. Each
 14323 expects, as arguments, a control string *format* described below, and a set of *pointer* arguments
 14324 indicating where the converted input should be stored. The result is undefined if there are
 14325 insufficient arguments for the format. If the format is exhausted while arguments remain, the
 14326 excess arguments are evaluated but are otherwise ignored.

14327 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
 14328 to the next unused argument. In this case, the conversion character '*%*' (see below) is replaced
 14329 by the sequence "*%n\$*", where *n* is a decimal integer in the range [1,{NL_ARGMAX}]. This
 14330 feature provides for the definition of format strings that select arguments in an order
 14331 appropriate to specific languages. In format strings containing the "*%n\$*" form of conversion
 14332 specifications, it is unspecified whether numbered arguments in the argument list can be
 14333 referenced from the format string more than once.

14334 The *format* can contain either form of a conversion specification—that is, '*%*' or "*%n\$*"—but the
 14335 two forms cannot normally be mixed within a single *format* string. The only exception to this is
 14336 that "*%%*" or "*%**" can be mixed with the "*%n\$*" form.

14337 The *fscanf()* function in all its forms allows for detection of a language-dependent radix
 14338 character in the input string. The radix character is defined in the program's locale (category
 14339 *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the
 14340 radix character defaults to a period ('.').

14341 The format is a character string, beginning and ending in its initial shift state, if any, composed
 14342 of zero or more directives. Each directive is composed of one of the following: one or more
 14343 white-space characters (<space>, <tab>, <newline>, <vertical-tab>, or <form-feed> characters);
 14344 an ordinary character (neither '*%*' nor a white-space character); or a conversion specification.
 14345 XSI Each conversion specification is introduced by the character '*%*' or the character sequence
 14346 "*%n\$*", after which the following appear in sequence:

- 14347 • An optional assignment-suppressing character '***'.
- 14348 • An optional non-zero decimal integer that specifies the maximum field width.
- 14349 • An option length modifier that specifies the size of the receiving object.
- 14350 • A conversion character that specifies the type of conversion to be applied. The valid
 14351 conversion characters are described below.

14352 The *fscanf()* functions execute each directive of the format in turn. If a directive fails, as detailed
 14353 below, the function shall return. Failures are described as input failures (due to the
 14354 unavailability of input bytes) or matching failures (due to inappropriate input).

- 14355 A directive composed of one or more white-space characters is executed by reading input until
14356 no more valid input can be read, or up to the first byte which is not a white-space character,
14357 which remains unread.
- 14358 A directive that is an ordinary character is executed as follows: the next byte is read from the
14359 input and compared with the byte that comprises the directive; if the comparison shows that
14360 they are not equivalent, the directive fails, and the differing and subsequent bytes remain
14361 unread. Similarly, if end-of-file, an encoding error, or a read error prevents a character from
14362 being read, the directive fails.
- 14363 A directive that is a conversion specification defines a set of matching input sequences, as
14364 described below for each conversion character. A conversion specification is executed in the
14365 following steps:
- 14366 Input white-space characters (as specified by *isspace()*) are skipped, unless the conversion
14367 specification includes a ' [', c, C, or n conversion character.
- 14368 An item is read from the input, unless the conversion specification includes an n conversion
14369 character. An input item is defined as the longest sequence of input bytes (up to any specified
14370 maximum field width, which may be measured in characters or bytes dependent on the
14371 conversion character) which is an initial subsequence of a matching sequence. The first byte, if
14372 any, after the input item remains unread. If the length of the input item is 0, the execution of the
14373 conversion specification fails; this condition is a matching failure, unless end-of-file, an encoding
14374 error, or a read error prevented input from the stream, in which case it is an input failure.
- 14375 Except in the case of a '%' conversion character, the input item (or, in the case of a %n
14376 conversion specification, the count of input bytes) is converted to a type appropriate to the
14377 conversion character. If the input item is not a matching sequence, the execution of the
14378 conversion specification fails; this condition is a matching failure. Unless assignment
14379 suppression was indicated by a '*' , the result of the conversion is placed in the object pointed
14380 to by the first argument following the *format* argument that has not already received a
14381 XSI conversion result if the conversion specification is introduced by '%', or in the *nth* argument if
14382 introduced by the character sequence "%n\$". If this object does not have an appropriate type, or
14383 if the result of the conversion cannot be represented in the space provided, the behavior is
14384 undefined.
- 14385 The length modifiers and their meanings are:
- 14386 *hh* Specifies that a following *d, i, o, u, x, X,* or *n* conversion specifier applies to an argument
14387 with type pointer to **signed char** or **unsigned char**.
- 14388 *h* Specifies that a following *d, i, o, u, x, X,* or *n* conversion specifier applies to an argument
14389 with type pointer to **short** or **unsigned short**.
- 14390 *l* (ell) Specifies that a following *d, i, o, u, x, X,* or *n* conversion specifier applies to an argument
14391 with type pointer to **long** or **unsigned long**; that a following *a, A, e, E, f, F, g,* or *G*
14392 conversion specifier applies to an argument with type pointer to **double**; or that a
14393 following *c, s,* or ' [' conversion specifier applies to an argument with type pointer to
14394 **wchar_t**.
- 14395 *ll* (ell-ell) Specifies that a following *d, i, o, u, x, X,* or *n* conversion specifier applies to an argument
14396 with type pointer to **long long** or **unsigned long long**.
- 14397 *j* Specifies that a following *d, i, o, u, x, X,* or *n* conversion specifier applies to an argument
14398 with type pointer to **intmax_t** or **uintmax_t**.
- 14399 *z* Specifies that a following *d, i, o, u, x, X,* or *n* conversion specifier applies to an argument
14400 with type pointer to **size_t** or the corresponding signed integer type.

14401	<i>t</i>	Specifies that a following <i>d</i> , <i>i</i> , <i>o</i> , <i>u</i> , <i>x</i> , <i>X</i> , or <i>n</i> conversion specifier applies to an argument with type pointer to ptrdiff_t or the corresponding unsigned type.
14402		
14403	<i>L</i>	Specifies that a following <i>a</i> , <i>A</i> , <i>e</i> , <i>E</i> , <i>f</i> , <i>F</i> , <i>g</i> , or <i>G</i> conversion specifier applies to an argument with type pointer to long double .
14404		
14405		If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.
14406		
14407		The following conversion characters are valid:
14408	<i>d</i>	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to int .
14409		
14410		
14411		
14412	<i>i</i>	Matches an optionally signed integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with 0 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to int .
14413		
14414		
14415		
14416	<i>o</i>	Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 8 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
14417		
14418		
14419		
14420	<i>u</i>	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
14421		
14422		
14423		
14424	<i>x</i>	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 16 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
14425		
14426		
14427		
14428	<i>a, e, f, g</i>	Matches an optionally signed floating-point number, infinity, or NaN, whose format is the same as expected for the subject sequence of <i>strtod()</i> . In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to float .
14429		
14430		
14431		
14432		If the <i>fprintf()</i> family of functions generates character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std. 754-1985, the <i>fscanf()</i> family of functions shall recognize them as input.
14433		
14434		
14435	<i>s</i>	Matches a sequence of bytes that are not white-space characters. The application shall ensure that the corresponding argument is a pointer to the initial byte of an array of char , signed char , or unsigned char large enough to accept the sequence and a terminating null character code, which shall be added automatically.
14436		
14437		
14438		
14439		If an <i>l</i> (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character is converted to a wide character as if by a call to the <i>mbrtowc()</i> function, with the conversion state described by an mbstate_t object initialized to zero before the first character is converted. The application shall ensure that the corresponding argument is a pointer to an array of wchar_t large enough to accept the sequence and the terminating null wide character, which shall be added automatically.
14440		
14441		
14442		
14443		
14444		
14445		

14446 [Matches a non-empty sequence of bytes from a set of expected bytes (the *scanset*). The
14447 normal skip over white-space characters is suppressed in this case. The application
14448 shall ensure that the corresponding argument is a pointer to the initial byte of an array
14449 of **char**, **signed char**, or **unsigned char** large enough to accept the sequence and a
14450 terminating null byte, which shall be added automatically.

14451 If an *l* (ell) qualifier is present, the input is a sequence of characters that begins in the
14452 initial shift state. Each character in the sequence is converted to a wide character as if
14453 by a call to the *mbrtowc*() function, with the conversion state described by an **mbstate_t**
14454 object initialized to zero before the first character is converted. The application shall
14455 ensure that the corresponding argument is a pointer to an array of **wchar_t** large
14456 enough to accept the sequence and the terminating null wide character, which shall be
14457 added automatically.

14458 The conversion specification includes all subsequent bytes in the *format* string up to
14459 and including the matching right square bracket (']'). The bytes between the square
14460 brackets (the *scanlist*) comprise the scanset, unless the byte after the left square bracket
14461 is a circumflex ('^'), in which case the scanset contains all bytes that do not appear in
14462 the scanlist between the circumflex and the right square bracket. If the conversion
14463 specification begins with "[]" or "[^]", the right square bracket is included in the
14464 scanlist and the next right square bracket is the matching right square bracket that ends
14465 the conversion specification; otherwise, the first right square bracket is the one that
14466 ends the conversion specification. If a '-' is in the scanlist and is not the first character,
14467 nor the second where the first character is a '^', nor the last character, the behavior is
14468 implementation-defined.

14469 *c* Matches a sequence of bytes of the number specified by the field width (1 if no field
14470 width is present in the conversion specification). The application shall ensure that the
14471 corresponding argument is a pointer to the initial byte of an array of **char**, **signed char**,
14472 or **unsigned char** large enough to accept the sequence. No null byte is added. The
14473 normal skip over white-space characters is suppressed in this case.

14474 If an *l* (ell) qualifier is present, the input is a sequence of characters that begins in the
14475 initial shift state. Each character in the sequence is converted to a wide character as if
14476 by a call to the *mbrtowc*() function, with the conversion state described by an **mbstate_t**
14477 object initialized to zero before the first character is converted. The application shall
14478 ensure that the corresponding argument is a pointer to an array of **wchar_t** large
14479 enough to accept the resulting sequence of wide characters. No null wide character is
14480 added.

14481 *p* Matches an implementation-defined set of sequences, which shall be the same as the set
14482 of sequences that is produced by the *%p* conversion of the corresponding *fprintf*()
14483 functions. The application shall ensure that the corresponding argument is a pointer to
14484 a pointer to **void**. The interpretation of the input item is implementation-defined. If the
14485 input item is a value converted earlier during the same program execution, the pointer
14486 that results shall compare equal to that value; otherwise, the behavior of the *%p*
14487 conversion is undefined.

14488 *n* No input is consumed. The application shall ensure that the corresponding argument is
14489 a pointer to the integer into which is to be written the number of bytes read from the
14490 input so far by this call to the *fscanf*() functions. Execution of a *%n* conversion
14491 specification does not increment the assignment count returned at the completion of
14492 execution of the function. No argument is converted, but one is consumed. If the
14493 conversion specification includes an assignment-suppressing character or a field width,
14494 the behavior is undefined.

14495	XSI	C	Same as <i>lc</i> .
14496	XSI	S	Same as <i>ls</i> .
14497		%	Matches a single '%'; no conversion or assignment occurs. The complete conversion specification shall be "%%".
14498			
14499			If a conversion specification is invalid, the behavior is undefined.
14500			The conversion characters <i>A</i> , <i>E</i> , <i>F</i> , <i>G</i> , and <i>X</i> are also valid and behave the same as, respectively, <i>a</i> ,
14501			<i>e</i> , <i>f</i> , <i>g</i> , and <i>x</i> .
14502			If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before
14503			any bytes matching the current conversion specification (except for % <i>n</i>) have been read (other
14504			than leading white-space characters, where permitted), execution of the current conversion
14505			specification terminates with an input failure. Otherwise, unless execution of the current
14506			conversion specification is terminated with a matching failure, execution of the following
14507			conversion specification (if any) is terminated with an input failure.
14508			Reaching the end of the string in <i>sscanf()</i> is equivalent to encountering end-of-file for <i>fscanf()</i> .
14509			If conversion terminates on a conflicting input, the offending input is left unread in the input.
14510			Any trailing white space (including newline characters) is left unread unless matched by a
14511			conversion specification. The success of literal matches and suppressed assignments is only
14512			directly determinable via the % <i>n</i> conversion specification.
14513			The <i>fscanf()</i> and <i>scanf()</i> functions may mark the <i>st_atime</i> field of the file associated with <i>stream</i>
14514			for update. The <i>st_atime</i> field shall be marked for update by the first successful execution of
14515			<i>fgetc()</i> , <i>fgets()</i> , <i>fread()</i> , <i>getc()</i> , <i>getchar()</i> , <i>gets()</i> , <i>fscanf()</i> , or <i>scanf()</i> using <i>stream</i> that returns data
14516			not supplied by a prior call to <i>ungetc()</i> .
14517			RETURN VALUE
14518			Upon successful completion, these functions shall return the number of successfully matched
14519			and assigned input items; this number can be 0 in the event of an early matching failure. If the
14520			input ends before the first matching failure or conversion, EOF shall be returned. If a read error
14521	CX		occurs, the error indicator for the stream is set, EOF shall be returned, and <i>errno</i> shall be set to
14522			indicate the error.
14523			ERRORS
14524			For the conditions under which the <i>fscanf()</i> functions fail and may fail, refer to <i>fgetc()</i> or
14525			<i>fgetwc()</i> .
14526			In addition, <i>fscanf()</i> may fail if:
14527	XSI	[EILSEQ]	Input byte sequence does not form a valid character.
14528	XSI	[EINVAL]	There are insufficient arguments.
14529			EXAMPLES
14530			The call:
14531			<pre>int i, n; float x; char name[50];</pre>
14532			<pre>n = scanf("%d%f%s", &i, &x, name);</pre>
14533			with the input line:
14534			<pre>25 54.32E-1 Hamster</pre>
14535			assigns to <i>n</i> the value 3, to <i>i</i> the value 25, to <i>x</i> the value 5.432, and <i>name</i> contains the string
14536			"Hamster".

14537 The call:

```
14538 int i; float x; char name[50];
14539 (void) scanf("%2d%f%d %[0123456789]", &i, &x, name);
```

14540 with input:

```
14541 56789 0123 56a72
```

14542 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to
14543 *getchar()* shall return the character 'a'.

14544 **Reading Data into an Array**

14545 The following call uses *fscanf()* to read three floating point numbers from standard input into
14546 the *input* array.

```
14547 float input[3];
14548 fscanf (stdin, "%f %f %f", input, input+1, input+2);
```

14549 **APPLICATION USAGE**

14550 If the application calling *fscanf()* has any objects of type **wint_t** or **wchar_t**, it must also include
14551 the **<wchar.h>** header to have these objects defined.

14552 **RATIONALE**

14553 None.

14554 **FUTURE DIRECTIONS**

14555 None.

14556 **SEE ALSO**

14557 *getc()*, *printf()*, *setlocale()*, *strtod()*, *strtol()*, *strtoul()*, *wcrtomb()*, the Base Definitions volume of
14558 IEEE Std. 1003.1-200x, **<langinfo.h>**, **<stdio.h>**, **<wchar.h>**, the Base Definitions volume of
14559 IEEE Std. 1003.1-200x, Chapter 7, Locale

14560 **CHANGE HISTORY**

14561 First released in Issue 1. Derived from Issue 1 of the SVID.

14562 **Issue 4**

14563 Use of the terms “byte” and “character” is rationalized to make it clear when single-byte and
14564 multi-byte values can be used. Similarly, use of the terms “conversion specification” and
14565 “conversion character” is now more precise.

14566 Various errors are corrected. For example, the description of the *d* conversion character
14567 contained an erroneous reference to *strtod()* in Issue 3. This is replaced in this issue by reference
14568 to *strtol()*.

14569 The DESCRIPTION is updated in a number of places to indicate further implications of the
14570 “%n\$” form of a conversion. All references to this functionality, which is not specified in the
14571 ISO C standard, are marked as extensions.

14572 The ERRORS section is changed to refer to the entries for *fgetc()* and *fgetwc()*, the [EINVAL]
14573 error is marked as an extension, and the [EILSEQ] error is added and marked as an extension.

14574 The detailed description of this function including the CHANGE HISTORY section for *scanf()* is
14575 provided here instead of under *scanf()*.

14576 The APPLICATION USAGE section is amended to record the need for **<sys/types.h>** or
14577 **<stddef.h>** if type **wchar_t** is required.

- 14578 The following changes are incorporated for alignment with the ISO C standard:
- 14579 • The type of the argument *format* for all functions, and the type of argument *s* for *sscanf()*, are
14580 changed from **char*** to **const char***.
 - 14581 • The description is updated in various places to align more closely with the text of the ISO C
14582 standard. In particular, this issue fully defines the *L* conversion character, allows for the
14583 support of multi-byte coded character sets (although these are not mandated by X/Open),
14584 and fills in a number of gaps in the definition (for example, by defining termination
14585 conditions for *sscanf()*).
 - 14586 • Following an ANSI interpretation, the effect of conversion specifications that consume no
14587 input is better defined, and is no longer marked as an extension.
- 14588 The following change is incorporated for alignment with the MSE working draft.
- 14589 • The *C* and *S* conversion characters are added, indicating a pointer in the argument list to the
14590 initial wide-character code of an array large enough to accept the input sequence.
- 14591 **Issue 5**
- 14592 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the *l* (ell) qualifier is now
14593 defined for the *c*, *s*, and *' ['* conversion characters.
- 14594 The DESCRIPTION is updated to indicate that if infinity and NaN can be generated by the
14595 *fprintf()* family of functions, then they are recognized by the *fscanf()* family.
- 14596 **Issue 6**
- 14597 The Open Group corrigenda items U021/7 and U028/10 have been applied. These correct
14598 several occurrences of “characters” in the text which have been replaced with the term “bytes”.
- 14599 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 14600 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
- 14601 • The prototypes for *fscanf()*, *scanf()*, and *sscanf()* are updated.
 - 14602 • The DESCRIPTION is updated.

14603 **NAME**

14604 fseek, fseeko — reposition a file-position indicator in a stream

14605 **SYNOPSIS**

14606 #include <stdio.h>

14607 int fseek(FILE *stream, long offset, int whence);

14608 CX int fseeko(FILE *stream, off_t offset, int whence);

14609

14610 **DESCRIPTION**

14611 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 14612 conflict between the requirements described here and the ISO C standard is unintentional. This
 14613 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

14614 The *fseek()* function shall set the file-position indicator for the stream pointed to by *stream*. If a
 14615 read or write error occurs, the error indicator for the stream shall be set and *fseek()* fails.

14616 The new position, measured in bytes from the beginning of the file, shall be obtained by adding
 14617 *offset* to the position specified by *whence*. The specified point is the beginning of the file for
 14618 {SEEK_SET}, the current value of the file-position indicator for {SEEK_CUR}, or end-of-file for
 14619 {SEEK_END}.

14620 If the stream is to be used with wide-character input/output functions, the application shall
 14621 ensure that *offset* is either 0 or a value returned by an earlier call to *ftell()* on the same stream and
 14622 *whence* is {SEEK_SET}.

14623 A successful call to *fseek()* shall clear the end-of-file indicator for the stream and undo any effects
 14624 of *ungetc()* and *ungetwc()* on the same stream. After an *fseek()* call, the next operation on an
 14625 update stream may be either input or output.

14626 If the most recent operation, other than *ftell()*, on a given stream is *flush()*, the file offset in the
 14627 underlying open file description shall be adjusted to reflect the location specified by *fseek()*.

14628 The *fseek()* function shall allow the file-position indicator to be set beyond the end of existing
 14629 data in the file. If data is later written at this point, subsequent reads of data in the gap shall
 14630 return bytes with the value 0 until data is actually written into the gap.

14631 CX The behavior of *fseek()* on devices which are incapable of seeking is implementation-defined.
 14632 The value of the file offset associated with such a device is undefined.

14633 If the stream is writable and buffered data had not been written to the underlying file, *fseek()*
 14634 shall cause the unwritten data to be written to the file and shall mark the *st_ctime* and *st_mtime*
 14635 fields of the file for update.

14636 In a locale with state-dependent encoding, whether *fseek()* restores the stream's shift state is
 14637 implementation-defined.

14638 The *fseeko()* function is equivalent to the *fseek()* function except that the *offset* argument is of
 14639 type **off_t**.

14640 **RETURN VALUE**14641 CX The *fseek()* and *fseeko()* functions shall return 0 if they succeed.14642 CX Otherwise, they shall return -1 and set *errno* to indicate the error.14643 **ERRORS**

14644 CX The *fseek()* and *fseeko()* functions shall fail if, either the *stream* is unbuffered or the *stream*'s
 14645 CX buffer needed to be flushed, and the call to *fseek()* or *fseeko()* causes an underlying *lseek()* or
 14646 *write()* to be invoked:

14647 CX 14648	[EAGAIN]	The O_NONBLOCK flag is set for the file descriptor and the process would be delayed in the write operation.
14649 CX 14650	[EBADF]	The file descriptor underlying the stream file is not open for writing or the stream's buffer needed to be flushed and the file is not open.
14651 CX	[EFBIG]	An attempt was made to write a file that exceeds the maximum file size.
14652 XSI	[EFBIG]	An attempt was made to write a file that exceeds the process' file size limit.
14653 CX 14654	[EFBIG]	The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.
14655 CX 14656	[EINTR]	The write operation was terminated due to the receipt of a signal, and no data was transferred.
14657 CX 14658	[EINVAL]	The <i>whence</i> argument is invalid. The resulting file-position indicator would be set to a negative value.
14659 CX 14660 14661 14662 14663	[EIO]	A physical I/O error has occurred, or the process is a member of a background process group attempting to perform a <i>write()</i> to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
14664 CX	[ENOSPC]	There was no free space remaining on the device containing the file.
14665 CX 14666	[EOVERFLOW]	For <i>fseek()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type long .
14667 CX 14668	[EOVERFLOW]	For <i>fseeko()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type off_t .
14669 CX	[EPIPE]	The file descriptor underlying <i>stream</i> is associated with a pipe or FIFO.
14670 CX 14671	[EPIPE]	An attempt was made to write to a pipe or FIFO that is not open for reading by any process; a SIGPIPE signal shall also be sent to the thread.
14672 CX 14673	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.

14674 **EXAMPLES**

14675 None.

14676 **APPLICATION USAGE**

14677 None.

14678 **RATIONALE**

14679 None.

14680 **FUTURE DIRECTIONS**

14681 None.

14682 **SEE ALSO**

14683 *fopen()*, *fsetpos()*, *ftell()*, *getrlimit()*, *rewind()*, *ulimit()*, *ungetc()*, the Base Definitions volume of
 14684 IEEE Std. 1003.1-200x, <stdio.h>

14685 **CHANGE HISTORY**

14686 First released in Issue 1. Derived from Issue 1 of the SVID.

14687 Issue 4

14688 In the DESCRIPTION, the words “The *seek()* function does not, by itself, extend the size of a
14689 file” are deleted.

14690 In the RETURN VALUE section, the value `-1` is marked as an extension. This is because the
14691 ISO POSIX-1 standard only requires that a non-zero value is returned.

14692 In the ERRORS section, text is added to indicate that error returns are only generated when
14693 either the stream is unbuffered, or if the stream buffer needs to be flushed.

14694 The “fail” and “may fail” parts of the ERRORS section are revised for consistency with *lseek()*
14695 and *write()*.

14696 Text associated with the [EIO] error is expanded and the [ENXIO] error is added.

14697 Text is added to explain how *fseek()* is used with wide-character input/output; this is marked as
14698 a WP extension.

14699 The [EFBIG] error is marked to show extensions.

14700 The APPLICATION USAGE section is added.

14701 The following change is incorporated for alignment with the ISO C standard:

- 14702 • The type of argument *offset* is now defined in full as **long** instead of **long**.

14703 The following change is incorporated for alignment with the FIPS requirements:

- 14704 • The [EINTR] error is no longer an indication that the implementation does not report partial
14705 transfers.

14706 Issue 4, Version 2

14707 In the ERRORS section, the description of [EIO] is updated to include the case where a physical
14708 I/O error occurs.

14709 Issue 5

14710 Normative text previously in the APPLICATION USAGE section is moved to the
14711 DESCRIPTION.

14712 Large File Summit extensions are added.

14713 Issue 6

14714 Extensions beyond the ISO C standard are now marked.

14715 The following new requirements on POSIX implementations derive from alignment with the
14716 Single UNIX Specification:

- 14717 • The *fseeko()* function is added.
- 14718 • The [EFBIG], [EOVERFLOW], and [ENXIO] mandatory error conditions are added.

14719 The following change is incorporated for alignment with the FIPS requirements:

- 14720 • The [EINTR] error is no longer an indication that the implementation does not report partial
14721 transfers.

14722 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

14723 The DESCRIPTION is updated to explicitly state that *fseek()* sets the file-position indicator, and
14724 then on error the error indicator is set and *fseek()* fails. This is for alignment with the
14725 ISO/IEC 9899:1999 standard.

14726 **NAME**

14727 fsetpos — set current file position

14728 **SYNOPSIS**

14729 #include <stdio.h>

14730 int fsetpos(FILE *stream, const fpos_t *pos);

14731 **DESCRIPTION**

14732 CX The functionality described on this reference page is aligned with the ISO C standard. Any
14733 conflict between the requirements described here and the ISO C standard is unintentional. This
14734 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

14735 The *fsetpos()* function shall set the file position and state indicators for the stream pointed to by
14736 *stream* according to the value of the object pointed to by *pos*, which the application shall ensure
14737 is a value obtained from an earlier call to *fgetpos()* on the same stream. If a read or write error
14738 occurs, the error indicator for the stream shall be set and *fsetpos()* fails.

14739 A successful call to the *fsetpos()* function shall clear the end-of-file indicator for the stream and
14740 undo any effects of *ungetc()* on the same stream. After an *fsetpos()* call, the next operation on an
14741 update stream may be either input or output.

14742 CX The behavior of *fsetpos()* on devices which are incapable of seeking is implementation-defined.
14743 The value of the file offset associated with such a device is undefined.

14744 **RETURN VALUE**

14745 The *fsetpos()* function shall return 0 if it succeeds; otherwise, it shall return a non-zero value and
14746 set *errno* to indicate the error.

14747 **ERRORS**14748 The *fsetpos()* function may fail if:

14749 CX [EBADF] The file descriptor underlying *stream* is not valid.

14750 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

14751

14752 **EXAMPLES**

14753 None.

14754 **APPLICATION USAGE**

14755 None.

14756 **RATIONALE**

14757 None.

14758 **FUTURE DIRECTIONS**

14759 None.

14760 **SEE ALSO**

14761 *fopen()*, *ftell()*, *rewind()*, *ungetc()*, the Base Definitions volume of IEEE Std. 1003.1-200x,
14762 <stdio.h>

14763 **CHANGE HISTORY**

14764 First released in Issue 4. Derived from the ISO C standard.

14765 **Issue 6**

14766 Extensions beyond the ISO C standard are now marked.

14767 An additional [ESPIPE] error condition is added for sockets.

14768	The DESCRIPTION is updated to avoid use of the term “must” for application requirements.	
14769	The DESCRIPTION is updated to clarify that the error indicator is set for the stream on a read or	
14770	write error. This is for alignment with the ISO/IEC 9899: 1999 standard.	

14771 NAME

14772 fstat — get file status

14773 SYNOPSIS

14774 #include <sys/stat.h>

14775 int fstat(int *fildev*, struct stat **buf*);

14776 DESCRIPTION

14777 The *fstat()* function obtains information about an open file associated with the file descriptor
14778 *fildev*, and writes it to the area pointed to by *buf*.14779 SHM If *fildev* references a shared memory object, the implementation need update in the **stat** structure
14780 pointed to by the *buf* argument only the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the
14781 S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be
14782 valid.14783 TYM If *fildev* references a typed memory object, the implementation need update in the **stat** structure
14784 pointed to by the *buf* argument only the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the
14785 S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be
14786 valid.14787 The *buf* argument is a pointer to a **stat** structure, as defined in <sys/stat.h>, into which
14788 information is placed concerning the file.14789 The structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atime*, *st_ctime*, and *st_mtime*
14790 shall have meaningful values for all file types defined in this volume of IEEE Std. 1003.1-200x.
14791 The value of the member *st_nlink* shall be set to the number of links to the file.14792 An implementation that provides additional or alternative file access control mechanisms may,
14793 under implementation-defined conditions, cause *fstat()* to fail.14794 The *fstat()* function updates any time-related fields as described in **File Times Update** (see the
14795 Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 6, Character Set), before writing into
14796 the **stat** structure.

14797 RETURN VALUE

14798 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
14799 indicate the error.

14800 ERRORS

14801 The *fstat()* function shall fail if:14802 [EBADF] The *fildev* argument is not a valid file descriptor.

14803 [EIO] An I/O error occurred while reading from the file system.

14804 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file
14805 serial number cannot be represented correctly in the structure pointed to by
14806 *buf*.14807 The *fstat()* function may fail if:14808 [EOVERFLOW] One of the values is too large to store into the structure pointed to by the *buf*
14809 argument.

14810 **EXAMPLES**14811 **Obtaining File Status Information**

14812 The following example shows how to obtain file status information for a file named
 14813 `/home/cnd/mod1`. The structure variable `buffer` is defined for the `stat` structure. The
 14814 `/home/cnd/mod1` file is opened with read/write privileges and is passed to the open file
 14815 descriptor `fildev`.

```
14816 #include <sys/types.h>
14817 #include <sys/stat.h>
14818 #include <fcntl.h>

14819 struct stat buffer;
14820 int      status;
14821 ...
14822 fildev = open("/home/cnd/mod1", O_RDWR);
14823 status = fstat(fildev, &buffer);
```

14824 **APPLICATION USAGE**

14825 None.

14826 **RATIONALE**

14827 None.

14828 **FUTURE DIRECTIONS**

14829 None.

14830 **SEE ALSO**

14831 `lstat()`, `stat()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<sys/stat.h>`, `<sys/types.h>`

14832 **CHANGE HISTORY**

14833 First released in Issue 1. Derived from Issue 1 of the SVID.

14834 **Issue 4**

14835 The `<sys/types.h>` header is now marked as optional (OH); this header need not be included on
 14836 XSI-conformant systems.

14837 The following changes are incorporated in the DESCRIPTION for alignment with the
 14838 ISO POSIX-1 standard:

- 14839 • A paragraph defining the contents of `stat` structure members is added.
- 14840 • The words “extended security controls” are replaced by “additional or alternative file access
 14841 control mechanisms”.

14842 **Issue 4, Version 2**

14843 The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- 14844 • The [EIO] error is added as a mandatory error indicated the occurrence of an I/O error.
- 14845 • The [EOVERFLOW] error is added as an optional error indicating that one of the values is
 14846 too large to store in the area pointed to by `buf`.

14847 **Issue 5**

14848 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

14849 Large File Summit extensions are added.

14850 **Issue 6**

14851 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

14852 The following new requirements on POSIX implementations derive from alignment with the
14853 Single UNIX Specification:

14854 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
14855 required for conforming implementations of previous POSIX specifications, it was not
14856 required for UNIX applications.

14857 • The [EIO] mandatory error condition is added.

14858 • The [EOVERFLOW] mandatory error condition is added. This change is to support large
14859 files.

14860 • The [EOVERFLOW] optional error condition is added.

14861 The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by specifying that
14862 shared memory object semantics apply to typed memory objects.

14863 **NAME**

14864 fstatvfs, statvfs — get file system information

14865 **SYNOPSIS**14866 XSI

```
#include <sys/statvfs.h>
```

14867

```
int fstatvfs(int fildev, struct statvfs *buf);
```

14868

```
int statvfs(const char *restrict path, struct statvfs *restrict buf);
```

14869

14870 **DESCRIPTION**14871 The *fstatvfs()* function obtains information about the file system containing the file referenced by
14872 *fildev*.14873 The following flags can be returned in the *f_flag* member:

14874 ST_RDONLY Read-only file system.

14875 ST_NOSUID Setuid/setgid bits ignored by *exec*.14876 The *statvfs()* function obtains descriptive information about the file system containing the file
14877 named by *path*.14878 For both functions, the *buf* argument is a pointer to a **statvfs** structure that shall be filled. Read,
14879 write, or execute permission of the named file is not required.14880 It is unspecified whether all members of the **statvfs** structure have meaningful values on all file
14881 systems.14882 **RETURN VALUE**14883 Upon successful completion, *statvfs()* shall return 0. Otherwise, it shall return -1 and set *errno* to
14884 indicate the error.14885 **ERRORS**14886 The *fstatvfs()* and *statvfs()* functions shall fail if:

14887 [EIO] An I/O error occurred while reading the file system.

14888 [EINTR] A signal was caught during execution of the function.

14889 [EOVERFLOW] One of the values to be returned cannot be represented correctly in the
14890 structure pointed to by *buf*.14891 The *fstatvfs()* function shall fail if:14892 [EBADF] The *fildev* argument is not an open file descriptor.14893 The *statvfs()* function shall fail if:

14894 [EACCES] Search permission is denied on a component of the path prefix.

14895 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
14896 argument.

14897 [ENAMETOOLONG]

14898 The length of a path name exceeds {PATH_MAX} or a path name component
14899 is longer than {NAME_MAX}.14900 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.14901 [ENOTDIR] A component of the path prefix of *path* is not a directory.14902 The *statvfs()* function may fail if:

14903 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 14904 resolution of the *path* argument.

14905 [ENAMETOOLONG]
 14906 Path name resolution of a symbolic link produced an intermediate result
 14907 whose length exceeds {PATH_MAX}.

14908 EXAMPLES

14909 Obtaining File System Information Using *fstatvfs()*

14910 The following example shows how to obtain file system information for the file system upon
 14911 which the file named */home/cnd/mod1* resides, using the *fstatvfs()* function. The
 14912 */home/cnd/mod1* file is opened with read/write privileges and the open file descriptor is passed
 14913 to the *fstatvfs()* function.

```
14914 #include <statvfs.h>
14915 #include <fcntl.h>
14916 struct statvfs buffer;
14917 int status;
14918 ...
14919 fildes = open("/home/cnd/mod1", O_RDWR);
14920 status = fstatvfs(fildes, &buffer);
```

14921 Obtaining File System Information Using *statvfs()*

14922 The following example shows how to obtain file system information for the file system upon
 14923 which the file named */home/cnd/mod1* resides, using the *statvfs()* function.

```
14924 #include <statvfs.h>
14925 struct statvfs buffer;
14926 int status;
14927 ...
14928 status = statvfs("/home/cnd/mod1", &buffer);
```

14929 APPLICATION USAGE

14930 None.

14931 RATIONALE

14932 None.

14933 FUTURE DIRECTIONS

14934 None.

14935 SEE ALSO

14936 *chmod()*, *chown()*, *creat()*, *dup()*, *exec*, *fcntl()*, *link()*, *mknod()*, *open()*, *pipe()*, *read()*, *time()*,
 14937 *unlink()*, *utime()*, *write()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/statvfs.h>

14938 CHANGE HISTORY

14939 First released in Issue 4, Version 2.

14940 Issue 5

14941 Moved from X/OPEN UNIX extension to BASE.

14942 Large File Summit extensions are added.

14943 **Issue 6**

- 14944 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 14945 The **restrict** keyword is added to the *statvfs()* prototype for alignment with the
14946 ISO/IEC 9899:1999 standard.
- 14947 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
14948 [ELOOP] error condition is added.

14949 **NAME**

14950 fsync — synchronize changes to a file

14951 **SYNOPSIS**

14952 FSC #include <unistd.h>

14953 int fsync(int *fildev*);

14954

14955 **DESCRIPTION**

14956 The *fsync()* function can be used by an application to indicate that all data for the open file
 14957 description named by *fildev* is to be transferred to the storage device associated with the file
 14958 described by *fildev* in an implementation-defined manner. The *fsync()* function shall not return
 14959 until the system has completed that action or until an error is detected.

14960 SIO If `_POSIX_SYNCHRONIZED_IO` is defined, the *fsync()* function shall force all currently queued
 14961 I/O operations associated with the file indicated by file descriptor *fildev* to the synchronized I/O
 14962 completion state. All I/O operations shall be completed as defined for synchronized I/O file
 14963 integrity completion.

14964 **RETURN VALUE**

14965 Upon successful completion, *fsync()* shall return 0. Otherwise, `-1` shall be returned and *errno* set
 14966 to indicate the error. If the *fsync()* function fails, outstanding I/O operations are not guaranteed
 14967 to have been completed.

14968 **ERRORS**14969 The *fsync()* function shall fail if:14970 [EBADF] The *fildev* argument is not a valid descriptor.14971 [EINTR] The *fsync()* function was interrupted by a signal.14972 [EINVAL] The *fildev* argument does not refer to a file on which this operation is possible.

14973 [EIO] An I/O error occurred while reading from or writing to the file system.

14974 In the event that any of the queued I/O operations fail, *fsync()* shall return the error conditions
 14975 defined for *read()* and *write()*.

14976 **EXAMPLES**

14977 None.

14978 **APPLICATION USAGE**

14979 The *fsync()* function should be used by programs which require modifications to a file to be
 14980 completed before continuing; for example, a program which contains a simple transaction
 14981 facility might use it to ensure that all modifications to a file or files caused by a transaction are
 14982 recorded.

14983 **RATIONALE**

14984 The *fsync()* function is intended to force a physical write of data from the buffer cache, and to
 14985 assure that after a system crash or other failure that all data up to the time of the *fsync()* call is
 14986 recorded on the disk. Since the concepts of “buffer cache”, “system crash”, “physical write”, and
 14987 “non-volatile storage” are not defined here, the wording has to be more abstract.

14988 If `_POSIX_SYNCHRONIZED_IO` is not defined, the wording relies heavily on the conformance
 14989 document to tell the user what can be expected from the system. It is explicitly intended that a
 14990 null implementation is permitted. This could be valid in the case where the system cannot assure
 14991 non-volatile storage under any circumstances or when the system is highly fault-tolerant and the
 14992 functionality is not required. In the middle ground between these extremes, *fsync()* might or
 14993 might not actually cause data to be written where it is safe from a power failure. The

14994 conformance document should identify at least that one configuration exists (and how to obtain
14995 that configuration) where this can be assured for at least some files that the user can select to use
14996 for critical data. It is not intended that an exhaustive list is required, but rather sufficient
14997 information is provided to let the user determine that if he or she has critical data he or she can
14998 configure her system to allow it to be written to non-volatile storage.

14999 It is reasonable to assert that the key aspects of *fsync()* are unreasonable to test in a test suite.
15000 That does not make the function any less valuable, just more difficult to test. A formal
15001 conformance test should probably force a system crash (power shutdown) during the test for
15002 this condition, but it needs to be done in such a way that automated testing does not require this
15003 to be done except when a formal record of the results is being made. It would also not be
15004 unreasonable to omit testing for *fsync()*, allowing it to be treated as a quality-of-implementation
15005 issue.

15006 **FUTURE DIRECTIONS**

15007 None.

15008 **SEE ALSO**

15009 *sync()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>

15010 **CHANGE HISTORY**

15011 First released in Issue 3.

15012 **Issue 4**

15013 The <**unistd.h**> header is added to the SYNOPSIS section.

15014 In the APPLICATION USAGE section, the words “require a file to be in a known state” are
15015 replaced by “require modifications to a file to be completed before continuing”.

15016 **Issue 5**

15017 Aligned with *fsync()* in the POSIX Realtime Extension. Specifically, the DESCRIPTION and
15018 RETURN VALUE sections are much expanded, and the ERRORS section is updated to indicate
15019 that *fsync()* can return the error conditions defined for *read()* and *write()*.

15020 **Issue 6**

15021 This function is marked as part of the File Synchronization option.

15022 The following new requirements on POSIX implementations derive from alignment with the
15023 Single UNIX Specification:

- 15024 • The [EINVAL] and [EIO] mandatory error conditions are added.

15025 **NAME**

15026 ftell, ftello — return a file offset in a stream

15027 **SYNOPSIS**

15028 #include <stdio.h>

15029 long ftell(FILE *stream);

15030 CX off_t ftello(FILE *stream);

15031

15032 **DESCRIPTION**15033 CX The functionality described on this reference page is aligned with the ISO C standard. Any
15034 conflict between the requirements described here and the ISO C standard is unintentional. This
15035 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.15036 The *ftell()* function shall obtain the current value of the file-position indicator for the stream
15037 pointed to by *stream*.15038 CX The *ftello()* function is identical to *ftell()* except that the return value is of type **off_t**.15039 **RETURN VALUE**15040 CX Upon successful completion, *ftell()* and *ftello()* shall return the current value of the file-position
15041 indicator for the stream measured in bytes from the beginning of the file.15042 CX Otherwise, *ftell()* and *ftello()* shall return -1 , cast to **long** and **off_t** respectively, and set *errno* to
15043 indicate the error.15044 **ERRORS**15045 CX The *ftell()* and *ftello()* functions shall fail if:15046 CX [EBADF] The file descriptor underlying *stream* is not an open file descriptor.15047 CX [EOVERFLOW] For *ftell()*, the current file offset cannot be represented correctly in an object of
15048 type **long**.15049 CX [EOVERFLOW] For *ftello()*, the current file offset cannot be represented correctly in an object
15050 of type **off_t**.15051 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.15052 The *ftell()* function may fail if:15053 CX [ESPIPE] The file descriptor underlying *stream* is associated with a socket.15054 **EXAMPLES**

15055 None.

15056 **APPLICATION USAGE**

15057 None.

15058 **RATIONALE**

15059 None.

15060 **FUTURE DIRECTIONS**

15061 None.

15062 **SEE ALSO**15063 *fgetpos()*, *fopen()*, *fseek()*, *lseek()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>

15064 **CHANGE HISTORY**

15065 First released in Issue 1. Derived from Issue 1 of the SVID.

15066 **Issue 4**

15067 The following change is incorporated for alignment with the ISO C standard:

- 15068 • The function return value is now defined in full as **long**. It was previously defined as **long**.

15069 **Issue 5**

15070 Large File Summit extensions are added.

15071 **Issue 6**

15072 Extensions beyond the ISO C standard are now marked.

15073 The following new requirements on POSIX implementations derive from alignment with the
15074 Single UNIX Specification:

- 15075 • The *ftello()* function is added.
- 15076 • The [EOVERFLOW] error conditions are added.

15077 An additional [ESPIPE] error condition is added for sockets.

15078 **NAME**15079 ftime — get date and time (**LEGACY**)15080 **SYNOPSIS**

15081 xSI #include <sys/timeb.h>

15082 int ftime(struct timeb *tp);

15083

15084 **DESCRIPTION**

15085 The *ftime()* function shall set the *time* and *millitm* members of the **timeb** structure pointed to by
15086 *tp* to contain the seconds and milliseconds portions, respectively, of the current time in seconds
15087 since the Epoch. The contents of the *timezone* and *dstflag* members of *tp* after a call to *ftime()* are
15088 unspecified.

15089 The system clock need not have millisecond granularity. Depending on any granularity
15090 (particularly a granularity of one) renders code non-portable.

15091 **RETURN VALUE**15092 Upon successful completion, the *ftime()* function shall return 0; otherwise, -1 shall be returned.15093 **ERRORS**

15094 No errors are defined.

15095 **EXAMPLES**15096 **Getting the Current Time and Date**

15097 The following example shows how to get the current system time values using the *ftime()*
15098 function. The **timeb** structure pointed to by *tp* is filled with the current system time values for
15099 *time* and *millitm*.

15100 #include <sys/timeb.h>

15101 struct timeb tp;

15102 int status;

15103 ...

15104 status = ftime(&tp);

15105 **APPLICATION USAGE**

15106 For applications portability, the *time()* function should be used to determine the current time
15107 instead of *ftime()*.

15108 **RATIONALE**

15109 None.

15110 **FUTURE DIRECTIONS**

15111 This function may be withdrawn in a future version.

15112 **SEE ALSO**

15113 *ctime()*, *gettimeofday()*, *time()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |
15114 <sys/timeb.h>

15115 **CHANGE HISTORY**

15116 First released in Issue 4, Version 2.

15117 **Issue 5**

15118 Moved from X/OPEN UNIX extension to BASE.

15119 Normative text previously in the APPLICATION USAGE section is moved to the
15120 DESCRIPTION.

15121 **Issue 6**

15122 This function is marked LEGACY.

15123 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since
15124 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time* |
15125 functions.

15126 NAME

15127 ftok — generate an IPC key

15128 SYNOPSIS

15129 xSI #include <sys/ipc.h>

15130 key_t ftok(const char *path, int id);

15131

15132 DESCRIPTION

15133 The *ftok()* function shall return a key based on *path* and *id* that is usable in subsequent calls to
 15134 *msgget()*, *semget()*, and *shmget()*. The application shall ensure that the *path* argument is the path
 15135 name of an existing file that the process is able to *stat()*.

15136 The *ftok()* function shall return the same key value for all paths that name the same file, when
 15137 called with the same *id* value, and return different key values when called with different *id*
 15138 values or with paths that name different files existing on the same file system at the same time. It
 15139 is unspecified whether *ftok()* shall return the same key value when called again after the file
 15140 named by *path* is removed and recreated with the same name.

15141 Only the low order 8-bits of *id* are significant. The behavior of *ftok()* is unspecified if these bits
 15142 are 0.

15143 RETURN VALUE

15144 Upon successful completion, *ftok()* shall return a key. Otherwise, *ftok()* shall return (**key_t**)-1
 15145 and set *errno* to indicate the error.

15146 ERRORS

15147 The *ftok()* function shall fail if:

15148 [EACCES] Search permission is denied for a component of the path prefix. |

15149 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* |
 15150 argument. |

15151 [ENAMETOOLONG] |
 15152 The length of the *path* argument exceeds {PATH_MAX} or a path name |
 15153 component is longer than {NAME_MAX}. |

15154 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string. |

15155 [ENOTDIR] A component of the path prefix is not a directory. |

15156 The *ftok()* function may fail if:

15157 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during |
 15158 resolution of the *path* argument. |

15159 [ENAMETOOLONG] |
 15160 Path name resolution of a symbolic link produced an intermediate result |
 15161 whose length exceeds {PATH_MAX}. |

15162 **EXAMPLES**15163 **Getting an IPC Key**

15164 The following example gets a unique key that can be used by the IPC functions *semget()*,
 15165 *msgget()*, and *shmget()*. The key returned by *ftok()* for this example is based on the ID value *S*
 15166 and the path name */tmp*.

```
15167 #include <sys/ipc.h>
15168 ...
15169 key_t key;
15170 char *path = "/tmp";
15171 int id = 'S';
15172 key = ftok(path, id);
```

15173 **Saving an IPC Key**

15174 The following example gets a unique key based on the path name */tmp* and the ID value *a*. It
 15175 also assigns the value of the resulting key to the *semkey* variable so that it will be available to a
 15176 later call to *semget()*, *msgget()*, or *shmget()*.

```
15177 #include <sys/ipc.h>
15178 ...
15179 key_t semkey;
15180 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
15181     perror("IPC error: ftok"); exit(1);
15182 }
```

15183 **APPLICATION USAGE**

15184 For maximum portability, *id* should be a single-byte character.

15185 **RATIONALE**

15186 None.

15187 **FUTURE DIRECTIONS**

15188 None.

15189 **SEE ALSO**

15190 *msgget()*, *semget()*, *shmget()*, the Base Definitions volume of IEEE Std. 1003.1-200x, *<sys/ipc.h>*

15191 **CHANGE HISTORY**

15192 First released in Issue 4, Version 2.

15193 **Issue 5**

15194 Moved from X/OPEN UNIX extension to BASE.

15195 **Issue 6**

15196 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

15197 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
 15198 [ELOOP] error condition is added.

15199 **NAME**

15200 ftruncate — truncate a file to a specified length

15201 **SYNOPSIS**

15202 #include <unistd.h>

15203 int ftruncate(int *fildev*, off_t *length*);15204 **DESCRIPTION**15205 If *fildev* is not a valid file descriptor open for writing, the *ftruncate()* function shall fail.

15206 If *fildev* refers to a regular file, the *ftruncate()* function shall cause the size of the file to be truncated to *length*. If the size of the file previously exceeded *length*, the extra data shall no longer be available to reads on the file. If the file previously was smaller than this size, *ftruncate()* shall either increase the size of the file or fail. XSI-conformant systems shall increase the size of the file. If the file size is increased, the extended area shall appear as if it were zero-filled. The value of the seek pointer shall not be modified by a call to *ftruncate()*.

15212 Upon successful completion, if *fildev* refers to a regular file, the *ftruncate()* function shall mark for update the *st_ctime* and *st_mtime* fields of the file and the S_ISUID and S_ISGID bits of the file mode may be cleared. If the *ftruncate()* function is unsuccessful, the file is unaffected.

15215 XSI If the request would cause the file size to exceed the soft file size limit for the process, the request shall fail and the implementation shall generate the SIGXFSZ signal for the process.

15217 If *fildev* refers to a directory, *ftruncate()* shall fail.15218 If *fildev* refers to any other file type, except a shared memory object, the result is unspecified.

15219 SHM If *fildev* refers to a shared memory object, *ftruncate()* shall set the size of the shared memory object to *length*.

15221 MF|SHM If the effect of *ftruncate()* is to decrease the size of a shared memory object or memory mapped file and whole pages beyond the new end were previously mapped, then the whole pages beyond the new end shall be discarded.

15224 MPR If the Memory Protection option is supported, references to discarded pages shall result in the generation of a SIGBUS signal; otherwise, the result of such references is undefined.

15226 MF|SHM If the effect of *ftruncate()* is to increase the size of a shared memory object, it is unspecified if the contents of any mapped pages between the old end-of-file and the new are flushed to the underlying object.

15229 **RETURN VALUE**

15230 Upon successful completion, *ftruncate()* shall return 0; otherwise, -1 shall be returned and *errno* set to indicate the error.

15232 **ERRORS**15233 The *ftruncate()* function shall fail if:

15234 [EINTR] A signal was caught during execution.

15235 [EINVAL] The *length* argument was less than 0.

15236 [EFBIG] or [EINVAL]

15237 The *length* argument was greater than the maximum file size.

15238 XSI [EFBIG] The file is a regular file and *length* is greater than the offset maximum established in the open file description associated with *fildev*.

15240 [EIO] An I/O error occurred while reading from or writing to a file system.

- 15241 [EBADF] or [EINVAL] |
 15242 The *fildest* argument is not a file descriptor open for writing. |
 15243 [EINVAL] The *fildest* argument references a file that was opened without write |
 15244 permission. |
 15245 [EROFS] The named file resides on a read-only file system. |

15246 EXAMPLES

15247 None.

15248 APPLICATION USAGE

15249 None.

15250 RATIONALE

15251 The *ftruncate()* function is part of IEEE Std. 1003.1-200x as it was deemed to be more useful than |
 15252 *truncate()*. The *truncate()* function is provided as an XSI extension. |

15253 FUTURE DIRECTIONS

15254 None.

15255 SEE ALSO

15256 *open()*, *truncate()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**> |

15257 CHANGE HISTORY

15258 First released in Issue 4, Version 2.

15259 Issue 5

15260 Moved from X/OPEN UNIX extension to BASE and aligned with *ftruncate()* in the POSIX |
 15261 Realtime Extension. Specifically, the DESCRIPTION is extensively reworded and [EROFS] is |
 15262 added to the list of mandatory errors that can be returned by *ftruncate()*.

15263 Large File Summit extensions are added.

15264 Issue 6

15265 The *truncate()* function has been split out into a separate reference page.

15266 The following new requirements on POSIX implementations derive from alignment with the |
 15267 Single UNIX Specification:

- 15268 • The DESCRIPTION is change to indicate that if the file size is changed, and if the file is a |
 15269 regular file, the S_ISUID and S_ISGID bits in the file mode may be cleared.

15270 The following changes were made to align with the IEEE P1003.1a draft standard:

- 15271 • The DESCRIPTION text is updated.

15272 XSI-conformant systems are required to increase the size of the file if the file was previously |
 15273 smaller than the size requested. |

15274 **NAME**

15275 ftrylockfile — stdio locking functions

15276 **SYNOPSIS**

15277 TSF #include <stdio.h>

15278 int ftrylockfile(FILE *file);

15279

15280 **DESCRIPTION**

15281 Refer to *flockfile()*.

15282 **NAME**

15283 ftw — traverse (walk) a file tree

15284 **SYNOPSIS**

15285 XSI #include <ftw.h>

```
15286 int ftw(const char *path, int (*fn)(const char *,
15287     const struct stat *ptr, int flag), int ndirs);
15288
```

15289 **DESCRIPTION**

15290 The *ftw()* function recursively descends the directory hierarchy rooted in *path*. For each object in
 15291 the hierarchy, *ftw()* shall call the function pointed to by *fn*, passing it a pointer to a null-
 15292 terminated character string containing the name of the object, a pointer to a **stat** structure
 15293 containing information about the object, and an integer. Possible values of the integer, defined
 15294 in the <ftw.h> header, are:

15295 FTW_D For a directory.

15296 FTW_DNR For a directory that cannot be read.

15297 FTW_F For a file.

15298 FTW_SL For a symbolic link (but see also FTW_NS below).

15299 FTW_NS For an object other than a symbolic link on which *stat()* could not successfully be
 15300 executed. If the object is a symbolic link and *stat()* failed, it is unspecified whether
 15301 *ftw()* passes FTW_SL or FTW_NS to the user-supplied function.

15302 If the integer is FTW_DNR, descendants of that directory shall not be processed. If the integer is
 15303 FTW_NS, the **stat** structure contains undefined values. An example of an object that would
 15304 cause FTW_NS to be passed to the function pointed to by *fn* would be a file in a directory with
 15305 read but without execute (search) permission.

15306 The *ftw()* function shall visit a directory before visiting any of its descendants.15307 The *ftw()* function shall use at most one file descriptor for each level in the tree.15308 The argument *ndirs* should be in the range of 1 to {OPEN_MAX}.

15309 The tree traversal shall continue until the tree is exhausted, an invocation of *fn* returns a non-
 15310 zero value, or some error, other than [EACCES], is detected within *ftw()*.

15311 The *ndirs* argument shall specify the maximum number of directory streams or file descriptors
 15312 or both available for use by *ftw()* while traversing the tree. When *ftw()* returns it shall close any
 15313 directory streams and file descriptors it uses not counting any opened by the application-
 15314 supplied *fn* function.

15315 **RETURN VALUE**

15316 If the tree is exhausted, *ftw()* shall return 0. If the function pointed to by *fn* returns a non-zero
 15317 value, *ftw()* shall stop its tree traversal and return whatever value was returned by the function
 15318 pointed to by *fn*. If *ftw()* detects an error, it shall return -1 and set *errno* to indicate the error.

15319 If *ftw()* encounters an error other than [EACCES] (see FTW_DNR and FTW_NS above), it shall
 15320 return -1 and set *errno* to indicate the error. The external variable *errno* may contain any error
 15321 value that is possible when a directory is opened or when one of the *stat* functions is executed on
 15322 a directory or file.

15323 **ERRORS**15324 The *ftw()* function shall fail if:15325 [EACCES] Search permission is denied for any component of *path* or read permission is
15326 denied for *path*.15327 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
15328 argument.

15329 [ENAMETOOLONG]

15330 The length of the *path* argument exceeds {PATH_MAX} or a path name
15331 component is longer than {NAME_MAX}.15332 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.15333 [ENOTDIR] A component of *path* is not a directory.15334 The *ftw()* function may fail if:15335 [EINVAL] The value of the *ndirs* argument is invalid.15336 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
15337 resolution of the *path* argument.

15338 [ENAMETOOLONG]

15339 Path name resolution of a symbolic link produced an intermediate result
15340 whose length exceeds {PATH_MAX}.15341 In addition, if the function pointed to by *fn* encounters system errors, *errno* may be set
15342 accordingly.15343 **EXAMPLES**15344 **Walking a Directory Structure**15345 The following example walks the current directory structure, calling the *fn* function for every
15346 directory entry, using at most 10 file descriptors:

```

15347 #include <ftw.h>
15348 ...
15349 if (ftw(".", fn, 10) != 0) {
15350     perror("ftw"); exit(2);
15351 }
```

15352 **APPLICATION USAGE**

15353 The *ftw()* function may allocate dynamic storage during its operation. If *ftw()* is forcibly
 15354 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *fn*
 15355 or an interrupt routine, *ftw()* does not have a chance to free that storage, so it remains
 15356 permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has
 15357 occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next
 15358 invocation.

15359 **RATIONALE**

15360 None.

15361 **FUTURE DIRECTIONS**

15362 None.

15363 **SEE ALSO**

15364 *longjmp()*, *lstat()*, *malloc()*, *nftw()*, *opendir()*, *siglongjmp()*, *stat()*, the Base Definitions volume of
 15365 IEEE Std. 1003.1-200x, <ftw.h>, <sys/stat.h>

15366 **CHANGE HISTORY**

15367 First released in Issue 1. Derived from Issue 1 of the SVID.

15368 **Issue 4**

15369 The type of argument *path* is changed from **char*** to **const char***. The argument list for *fn* has
 15370 also been defined.

15371 In the DESCRIPTION, the words “other than [EACCES]” are added to the paragraph describing
 15372 termination conditions for tree traversal.

15373 The following change is incorporated for alignment with the FIPS requirements:

- 15374 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path
 15375 name component is larger than {NAME_MAX} is now defined as mandatory and marked as
 15376 an extension.

15377 **Issue 4, Version 2**

15378 The following changes are incorporated for X/OPEN UNIX conformance:

- 15379 • The DESCRIPTION is updated to describe the use of the FTW_SL and FTW_NS values for a
 15380 symbolic link.
- 15381 • The DESCRIPTION states that *ftw()* uses at most one file descriptor for each level in the tree.
- 15382 • The DESCRIPTION constrains *ndirs* to the range from 1 to {OPEN_MAX}.
- 15383 • The RETURN VALUE section is updated to describe the case where *ftw()* encounters an error
 15384 other than [EACCES].
- 15385 • In the ERRORS section, a second [ENAMETOOLONG] condition is defined that may report
 15386 excessive length of an intermediate result of path name resolution of a symbolic link.

15387 **Issue 5**

15388 UX codings in the DESCRIPTION, RETURN VALUE, and ERRORS sections have been changed
 15389 to EX.

15390 **Issue 6**

15391 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 15392 • The [ENAMETOOLONG] error is restored as an error dependent on _POSIX_NO_TRUNC.
 15393 This is since behavior may vary from one file system to another.

15394 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
 15395 [ELOOP] error condition is added.

15396 **NAME**

15397 funlockfile — stdio locking functions

15398 **SYNOPSIS**

15399 TSF #include <stdio.h>

15400 void funlockfile(FILE *file);

15401

15402 **DESCRIPTION**

15403 Refer to *flockfile()*.

15404 **NAME**

15405 fwide — set stream orientation

15406 **SYNOPSIS**

15407 #include <stdio.h>

15408 #include <wchar.h>

15409 int fwide(FILE **stream*, int *mode*);15410 **DESCRIPTION**

15411 CX The functionality described on this reference page is aligned with the ISO C standard. Any
15412 conflict between the requirements described here and the ISO C standard is unintentional. This
15413 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

15414 The *fwide()* function shall determine the orientation of the stream pointed to by *stream*. If *mode* is
15415 greater than zero, the function first attempts to make the stream wide-oriented. If *mode* is less
15416 than zero, the function first attempts to make the stream byte-oriented. Otherwise, *mode* is zero
15417 and the function does not alter the orientation of the stream.

15418 If the orientation of the stream has already been determined, *fwide()* shall not change it.

15419 Because no return value is reserved to indicate an error, an application wishing to check for error
15420 situations should set *errno* to 0, then call *fwide()*, then check *errno*, and if it is non-zero, assume
15421 an error has occurred.

15422 **RETURN VALUE**

15423 The *fwide()* function shall return a value greater than zero if, after the call, the stream has wide-
15424 orientation, a value less than zero if the stream has byte-orientation, or zero if the stream has no
15425 orientation.

15426 **ERRORS**15427 The *fwide()* function may fail if:

15428 CX [EBADF] The *stream* argument is not a valid stream.

15429 **EXAMPLES**

15430 None.

15431 **APPLICATION USAGE**

15432 A call to *fwide()* with *mode* set to zero can be used to determine the current orientation of a
15433 stream.

15434 **RATIONALE**

15435 None.

15436 **FUTURE DIRECTIONS**

15437 None.

15438 **SEE ALSO**

15439 The Base Definitions volume of IEEE Std. 1003.1-200x, <wchar.h>

15440 **CHANGE HISTORY**

15441 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
15442 (E).

15443 **Issue 6**

15444 Extensions beyond the ISO C standard are now marked.

15445 NAME

15446 fwprintf, swprintf, wprintf — print formatted wide-character output

15447 SYNOPSIS

15448 #include <stdio.h>

15449 #include <wchar.h>

15450 int fwprintf(FILE *restrict *stream*, const wchar_t *restrict *format*, ...);15451 int swprintf(wchar_t *restrict *ws*, size_t *n*, const wchar_t *restrict *format*, ...);15452 int wprintf(const wchar_t *restrict *format*, ...);

15453 DESCRIPTION

15454 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 15455 conflict between the requirements described here and the ISO C standard is unintentional. This
 15456 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

15457 The *fwprintf()* function places output on the named output *stream*. The *wprintf()* function places
 15458 output on the standard output stream *stdout*. The *swprintf()* function places output followed by
 15459 the null wide character in consecutive wide characters starting at **ws*; no more than *n* wide
 15460 characters are written, including a terminating null wide character, which is always added
 15461 (unless *n* is zero).

15462 Each of these functions converts, formats, and prints its arguments under control of the *format*
 15463 wide-character string. The *format* is composed of zero or more directives: *ordinary wide-*
 15464 *characters*, which are simply copied to the output stream, and *conversion specifications*, each of
 15465 which results in the fetching of zero or more arguments. The results are undefined if there are
 15466 insufficient arguments for the *format*. If the *format* is exhausted while arguments remain, the
 15467 excess arguments are evaluated but are otherwise ignored.

15468 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
 15469 to the next unused argument. In this case, the conversion wide character '%' (see below) is
 15470 replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL_ARGMAX}],
 15471 giving the position of the argument in the argument list. This feature provides for the definition
 15472 of *format* wide-character strings that select arguments in an order appropriate to specific
 15473 languages (see the EXAMPLES section).

15474 In *format* wide-character strings containing the "%n\$" form of conversion specifications,
 15475 numbered arguments in the argument list can be referenced from the *format* wide-character
 15476 string as many times as required.

15477 In *format* wide-character strings containing the '%' form of conversion specifications, each
 15478 argument in the argument list is used exactly once.

15479 All forms of the *fwprintf()* function allow for the insertion of a locale-dependent radix character
 15480 in the output string, output as a wide-character value. The radix character is defined in the
 15481 program's locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix
 15482 character is not defined, the radix character defaults to a period ('.').

15483 XSI Each conversion specification is introduced by the '%' wide character or by the wide-character
 15484 sequence "%n\$", after which the following appear in sequence:

- 15485 • Zero or more *flags* (in any order), which modify the meaning of the conversion specification.
- 15486 • An optional minimum *field width*. If the converted value has fewer wide characters than the
 15487 field width, it shall be padded with spaces by default on the left; it shall be padded on the
 15488 right, if the left-adjustment flag ('-'), described below, is given to the field width. The field
 15489 width takes the form of an asterisk ('*'), described below, or a decimal integer.

- 15490 • An optional *precision* that gives the minimum number of digits to appear for the *d*, *i*, *o*, *u*, *x*,
 15491 and *X* conversions; the number of digits to appear after the radix character for the *e*, *E*, and *f*
 15492 conversions; the maximum number of significant digits for the *g* and *G* conversions; or the
 15493 maximum number of wide characters to be printed from a string in *s* conversions. The
 15494 precision takes the form of a period (*'.'*) followed either by an asterisk (*'*'*), described
 15495 below, or an optional decimal digit string, where a null digit string is treated as 0. If a
 15496 precision appears with any other conversion wide character, the behavior is undefined.
- 15497 • An optional length modifier that specifies the size of the argument.
- 15498 • A *conversion wide character* that indicates the type of conversion to be applied.
- 15499 A field width, or precision, or both, may be indicated by an asterisk (*'*'*). In this case an
 15500 argument of type `int` supplies the field width or precision. The application shall ensure that
 15501 arguments specifying field width, or precision, or both appear in that order before the argument,
 15502 if any, to be converted. A negative field width is taken as a *'-'* flag followed by a positive field
 15503 XSI width. A negative precision is taken as if the precision were omitted. In format wide-character
 15504 strings containing the "`%n$`" form of a conversion specification, a field width or precision may
 15505 be indicated by the sequence "`*m$`", where *m* is a decimal integer in the range
 15506 [1,{NL_ARGMAX}] giving the position in the argument list (after the format argument) of an
 15507 integer argument containing the field width or precision, for example:
- ```
15508 wprintf(L"%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```
- 15509 The *format* can contain either numbered argument specifications (that is, "`%n$`" and "`*m$`"), or  
 15510 unnumbered argument specifications (that is, *'%'* and *'\*'*), but normally not both. The only  
 15511 exception to this is that "`%%`" can be mixed with the "`%n$`" form. The results of mixing  
 15512 numbered and unnumbered argument specifications in a *format* wide-character string are  
 15513 undefined. When numbered argument specifications are used, specifying the *N*th argument  
 15514 requires that all the leading arguments, from the first to the (*N*-1)th, are specified in the format  
 15515 wide-character string.
- 15516 The flag wide characters and their meanings are:
- 15517 XSI *'* The integer portion of the result of a decimal conversion (`%i`, `%d`, `%u`, `%f`, `%g`, or `%G`)  
 15518 shall be formatted with thousands' grouping wide characters. For other conversions,  
 15519 the behavior is undefined. The numeric grouping wide character is used.
- 15520 *-* The result of the conversion shall be left-justified within the field. The conversion shall  
 15521 be right-justified if this flag is not specified.
- 15522 *+* The result of a signed conversion shall always begin with a sign (*'+' or '-'*). The  
 15523 conversion shall begin with a sign only when a negative value is converted if this flag is  
 15524 not specified.
- 15525 <space> If the first wide character of a signed conversion is not a sign, or if a signed conversion  
 15526 results in no wide characters, a space shall be prefixed to the result. This means that if  
 15527 the <space> and *'+'* flags both appear, the space flag shall be ignored.
- 15528 *#* This flag specifies that the value is to be converted to an alternative form. For *o*  
 15529 conversion, it increases the precision (if necessary) to force the first digit of the result to  
 15530 be 0. For *x* or *X* conversions, a non-zero result shall have 0*x* (or 0*X*) prefixed to it. For *e*,  
 15531 *E*, *f*, *g*, or *G* conversions, the result shall always contain a radix character, even if no  
 15532 digits follow it. Without this flag, a radix character appears in the result of these  
 15533 conversions only if a digit follows it. For *g* and *G* conversions, trailing zeros shall *not* be  
 15534 removed from the result as they normally are. For other conversions, the behavior is  
 15535 undefined.

|       |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 15536 | 0                   | For <i>d, i, o, u, x, X, e, E, f, g,</i> and <i>G</i> conversions, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the 0 and '-' flags both appear, the 0 flag shall be ignored. For <i>d, i, o, u, x,</i> and <i>X</i> conversions, if a precision is specified, the 0 flag shall be ignored. If the 0 and ''' flags both appear, the grouping wide characters are inserted before zero padding. For other conversions, the behavior is undefined. |
| 15537 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15538 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15539 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15540 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15541 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15542 |                     | The length modifiers and their meanings are:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 15543 | <i>hh</i>           | Specifies that a following <i>d, i, o, u, x,</i> or <i>X</i> conversion specifier applies to a <b>signed char</b> or <b>unsigned char</b> argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to <b>signed char</b> or <b>unsigned char</b> before printing); or that a following <i>n</i> conversion specifier applies to a pointer to a <b>signed char</b> argument.                                                                                            |
| 15544 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15545 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15546 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15547 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15548 | <i>h</i>            | Specifies that a following <i>d, i, o, u, x,</i> or <i>X</i> conversion specifier applies to a <b>short</b> or <b>unsigned short</b> argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to <b>short</b> or <b>unsigned short</b> before printing); or that a following <i>n</i> conversion specifier applies to a pointer to a <b>short</b> argument.                                                                                                            |
| 15549 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15550 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15551 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15552 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15553 | <i>l</i> (ell)      | Specifies that a following <i>d, i, o, u, x,</i> or <i>X</i> conversion specifier applies to a <b>long</b> or <b>unsigned long</b> argument; that a following <i>n</i> conversion specifier applies to a pointer to a <b>long</b> argument; that a following <i>c</i> conversion specifier applies to a <b>wint_t</b> argument; that a following <i>s</i> conversion specifier applies to a pointer to a <b>wchar_t</b> argument; or has no effect on a following <i>a, A, e, E, f, F, g,</i> or <i>G</i> conversion specifier.    |
| 15554 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15555 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15556 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15557 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15558 | <i>ll</i> (ell-ell) | Specifies that a following <i>d, i, o, u, x,</i> or <i>X</i> conversion specifier applies to a <b>long long</b> or <b>unsigned long long</b> argument; or that a following <i>n</i> conversion specifier applies to a pointer to a <b>long long</b> argument.                                                                                                                                                                                                                                                                      |
| 15559 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15560 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15561 | <i>j</i>            | Specifies that a following <i>d, i, o, u, x,</i> or <i>X</i> conversion specifier applies to an <b>intmax_t</b> or <b>uintmax_t</b> argument; or that a following <i>n</i> conversion specifier applies to a pointer to an <b>intmax_t</b> argument.                                                                                                                                                                                                                                                                               |
| 15562 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15563 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15564 | <i>z</i>            | Specifies that a following <i>d, i, o, u, x,</i> or <i>X</i> conversion specifier applies to a <b>size_t</b> or the corresponding signed integer type argument; or that a following <i>n</i> conversion specifier applies to a pointer to a signed integer type corresponding to <b>size_t</b> argument.                                                                                                                                                                                                                           |
| 15565 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15566 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15567 | <i>t</i>            | Specifies that a following <i>d, i, o, u, x,</i> or <i>X</i> conversion specifier applies to a <b>ptrdiff_t</b> or the corresponding <b>unsigned</b> type argument; or that a following <i>n</i> conversion specifier applies to a pointer to a <b>ptrdiff_t</b> argument.                                                                                                                                                                                                                                                         |
| 15568 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15569 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15570 | <i>L</i>            | Specifies that a following <i>a, A, e, E, f, F, g,</i> or <i>G</i> conversion specifier applies to a <b>long double</b> argument.                                                                                                                                                                                                                                                                                                                                                                                                  |
| 15571 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15572 |                     | If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                                               |
| 15573 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15574 |                     | The conversion wide characters and their meanings are:                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 15575 | <i>d, i</i>         | The <b>int</b> argument is converted to a signed decimal in the style <code>[-]ddd</code> . The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no wide characters.                                                                                                                                                |
| 15576 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15577 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15578 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15579 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 15580 | <i>o</i>            | The <b>unsigned</b> argument is converted to unsigned octal format in the style <code>ddd</code> . The precision specifies the minimum number of digits to appear; if the value being                                                                                                                                                                                                                                                                                                                                              |
| 15581 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

|       |             |                                                                                                                               |
|-------|-------------|-------------------------------------------------------------------------------------------------------------------------------|
| 15582 |             | converted can be represented in fewer digits, it shall be expanded with leading zeros.                                        |
| 15583 |             | The default precision is 1. The result of converting 0 with an explicit precision of 0 is no                                  |
| 15584 |             | wide characters.                                                                                                              |
| 15585 | <i>u</i>    | The <b>unsigned</b> argument is converted to unsigned decimal format in the style <i>ddd</i> . The                            |
| 15586 |             | precision specifies the minimum number of digits to appear; if the value being                                                |
| 15587 |             | converted can be represented in fewer digits, it shall be expanded with leading zeros.                                        |
| 15588 |             | The default precision is 1. The result of converting 0 with an explicit precision of 0 is no                                  |
| 15589 |             | wide characters.                                                                                                              |
| 15590 | <i>x</i>    | The <b>unsigned</b> argument is converted to unsigned hexadecimal format in the style <i>ddd</i> ;                            |
| 15591 |             | the letters "abcdef" are used. The precision specifies the minimum number of digits                                           |
| 15592 |             | to appear; if the value being converted can be represented in fewer digits, it shall be                                       |
| 15593 |             | expanded with leading zeros. The default precision is 1. The result of converting 0 with                                      |
| 15594 |             | an explicit precision of 0 is no wide characters.                                                                             |
| 15595 | <i>X</i>    | Behaves the same as the <i>x</i> conversion wide character except that letters "ABCDEF" are                                   |
| 15596 |             | used instead of "abcdef".                                                                                                     |
| 15597 | <i>f, F</i> | The <b>double</b> argument is converted to decimal notation in the style <i>[-]ddd.ddd</i> , where                            |
| 15598 |             | the number of digits after the radix character is equal to the precision specification. If                                    |
| 15599 |             | the precision is missing, it is taken as 6; if the precision is explicitly 0 and no '#' flag is                               |
| 15600 |             | present, no radix character appears. If a radix character appears, at least one digit                                         |
| 15601 |             | appears before it. The value is rounded to the appropriate number of digits.                                                  |
| 15602 |             | A <b>double</b> argument representing an infinity is converted in one of the styles <i>[-]inf</i> or                          |
| 15603 |             | <i>[-]infinity</i> ; which style is implementation-defined. A <b>double</b> argument representing a                           |
| 15604 |             | NaN is converted in one of the styles <i>[-]nan</i> or <i>[-]nan(n-char-sequence)</i> ; which style, and                      |
| 15605 |             | the meaning of any <i>n-char-sequence</i> , is implementation-defined. The <i>F</i> conversion                                |
| 15606 |             | specifier produces INF, INFINITY, or NAN instead of inf, infinity, or nan, respectively.                                      |
| 15607 | <i>e, E</i> | The <b>double</b> argument is converted in the style <i>[-]d.ddde±dd</i> , where there is one digit                           |
| 15608 |             | before the radix character (which is non-zero if the argument is non-zero) and the                                            |
| 15609 |             | number of digits after it is equal to the precision; if the precision is missing, it is taken                                 |
| 15610 |             | as 6; if the precision is 0 and no '#' flag is present, no radix character appears. The                                       |
| 15611 |             | value is rounded to the appropriate number of digits. The <i>E</i> conversion wide character                                  |
| 15612 |             | shall produce a number with <i>E</i> instead of <i>e</i> introducing the exponent. The exponent                               |
| 15613 |             | always contains at least two digits. If the value is 0, the exponent is 0.                                                    |
| 15614 |             | A <b>double</b> argument representing an infinity or NaN is converted in the style of an <i>f</i> or <i>F</i>                 |
| 15615 |             | conversion specifier.                                                                                                         |
| 15616 | <i>g, G</i> | The <b>double</b> argument is converted in the style <i>f</i> or <i>e</i> (or in the style <i>E</i> in the case of a <i>G</i> |
| 15617 |             | conversion wide character), with the precision specifying the number of significant                                           |
| 15618 |             | digits. If an explicit precision is 0, it is taken as 1. The style used depends on the value                                  |
| 15619 |             | converted; style <i>e</i> (or <i>E</i> ) shall be used only if the exponent resulting from such a                             |
| 15620 |             | conversion is less than -4 or greater than or equal to the precision. Trailing zeros are                                      |
| 15621 |             | removed from the fractional portion of the result; a radix character appears only if it is                                    |
| 15622 |             | followed by a digit.                                                                                                          |
| 15623 |             | A <b>double</b> argument representing an infinity or NaN is converted in the style of an <i>f</i> or <i>F</i>                 |
| 15624 |             | conversion specifier.                                                                                                         |
| 15625 | <i>a, A</i> | A <b>double</b> argument representing a floating-point number is converted in the style                                       |
| 15626 |             | <i>[-]0xh.hhhh p1d</i> , where there is one hexadecimal digit (which is non-zero if the                                       |
| 15627 |             | argument is a normalized floating-point number and is otherwise unspecified) before                                           |
| 15628 |             | the decimal-point character 235) and the number of hexadecimal digits after it is equal                                       |

15629 to the precision; if the precision is missing and FLT\_RADIX is a power of 2, then the  
15630 precision is sufficient for an exact representation of the value; if the precision is missing  
15631 and FLT\_RADIX is not a power of 2, then the precision is sufficient to distinguish 236  
15632 values of type **double**, except that trailing zeros may be omitted; if the precision is zero  
15633 and the '#' flag is not specified, no decimal-point character appears. The letters  
15634 "abcdef" are used for a conversion and the letters "ABCDEF" for A conversion. The A  
15635 conversion specifier produces a number with 'X' and 'P' instead of 'x' and 'p'.  
15636 The exponent always contains at least one digit, and only as many more digits as  
15637 necessary to represent the decimal exponent of 2. If the value is zero, the exponent is  
15638 zero.

15639 A **double** argument representing an infinity or NaN is converted in the style of an *f* or *F*  
15640 conversion specifier.

15641 *c* If no *l* (ell) qualifier is present, the **int** argument is converted to a wide character as if by  
15642 calling the *btowc*() function and the resulting wide character is written. Otherwise, the  
15643 **wint\_t** argument is converted to **wchar\_t**, and written.

15644 *s* If no *l* (ell) qualifier is present, the application shall ensure that the argument is a  
15645 pointer to a character array containing a character sequence beginning in the initial  
15646 shift state. Characters from the array are converted as if by repeated calls to the  
15647 *mbrtowc*() function, with the conversion state described by an **mbstate\_t** object  
15648 initialized to zero before the first character is converted, and written up to (but not  
15649 including) the terminating null wide character. If the precision is specified, no more  
15650 than that many wide characters are written. If the precision is not specified, or is  
15651 greater than the size of the array, the application shall ensure that the array contains a  
15652 null wide character.

15653 If an *l* (ell) qualifier is present, the application shall ensure that the argument is a  
15654 pointer to an array of type **wchar\_t**. Wide characters from the array are written up to  
15655 (but not including) a terminating null wide character. If no precision is specified, or is  
15656 greater than the size of the array, the application shall ensure that the array contains a  
15657 null wide character. If a precision is specified, no more than that many wide characters  
15658 are written.

15659 *p* The application shall ensure that the argument is a pointer to **void**. The value of the  
15660 pointer is converted to a sequence of printable wide characters, in an implementation-  
15661 defined manner.

15662 *n* The application shall ensure that the argument is a pointer to an integer into which is  
15663 written the number of wide characters written to the output so far by this call to one of  
15664 the *fwprintf*() functions. No argument is converted, but one is consumed. If the  
15665 conversion specification includes any flags, a field width, or a precision, the behavior is  
15666 undefined.

15667 XSI *C* Same as *lc*.

15668 XSI *S* Same as *ls*.

15669 % Output a '%' wide character; no argument is converted. The entire conversion  
15670 specification shall be "%%".

15671 If a conversion specification does not match one of the above forms, the behavior is undefined.

15672 In no case does a nonexistent or small field width cause truncation of a field; if the result of a  
15673 conversion is wider than the field width, the field is simply expanded to contain the conversion  
15674 result. Characters generated by *fwprintf*() and *wprintf*() are printed as if *fputwc*() had been  
15675 called.

15676 For *a* and *A* conversions, if `FLT_RADIX` is a power of 2, the value is correctly rounded to a  
15677 hexadecimal floating number with the given precision.

15678 If `FLT_RADIX` is not a power of 2, the result should be one of the two adjacent numbers in  
15679 hexadecimal floating style with the given precision, with the extra stipulation that the error  
15680 should have a correct sign for the current rounding direction.

15681 For *e*, *E*, *f*, *F*, *g*, and *G* conversions, if the number of significant decimal digits is at most  
15682 `DECIMAL_DIG`, then the result should be correctly rounded. If the number of significant  
15683 decimal digits is more than `DECIMAL_DIG` but the source value is exactly representable with  
15684 `DECIMAL_DIG` digits, then the result should be an exact representation with trailing zeros.  
15685 Otherwise, the source value is bounded by two adjacent decimal strings "*L* < *U*", both having  
15686 `DECIMAL_DIG` significant digits; the value of the resultant decimal string "*D*" should satisfy "*L*  
15687 <= *D* <= *U*", with the extra stipulation that the error should have a correct sign for the current  
15688 rounding direction.

15689 CX The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the call to a  
15690 successful execution of `fwprintf()` or `wprintf()` and the next successful completion of a call to  
15691 `fflush()` or `fclose()` on the same stream, or a call to `exit()` or `abort()`.

#### 15692 RETURN VALUE

15693 Upon successful completion, these functions shall return the number of wide characters  
15694 transmitted, excluding the terminating null wide character in the case of `swprintf()`, or a negative  
15695 CX value if an output error was encountered, and set *errno* to indicate the error.

15696 If *n* or more wide characters were requested to be written, `swprintf()` shall return a negative  
15697 CX value, and set *errno* to indicate the error.

#### 15698 ERRORS

15699 For the conditions under which `fwprintf()` and `wprintf()` fail and may fail, refer to `fputwc()`.

15700 In addition, all forms of `wprintf()` may fail if:

15701 XSI [EILSEQ] A wide-character code that does not correspond to a valid character has been  
15702 detected.

15703 XSI [EINVAL] There are insufficient arguments.

15704 In addition, `wprintf()` and `fwprintf()` may fail if:

15705 XSI [ENOMEM] Insufficient storage space is available.

#### 15706 EXAMPLES

15707 To print the language-independent date and time format, the following statement could be used:

```
15708 wprintf(format, weekday, month, day, hour, min);
```

15709 For American usage, *format* could be a pointer to the wide-character string:

```
15710 L"%s, %s %d, %d:%.2d\n"
```

15711 producing the message:

```
15712 Sunday, July 3, 10:02
```

15713 whereas for German usage, *format* could be a pointer to the wide-character string:

```
15714 L"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

15715 producing the message:

```
15716 Sonntag, 3. Juli, 10:02
```

15717 **APPLICATION USAGE**

15718 None.

15719 **RATIONALE**

15720 None.

15721 **FUTURE DIRECTIONS**

15722 None.

15723 **SEE ALSO**

15724 *btowc()*, *fputwc()*, *fwscanf()*, *mbrtowc()*, *setlocale()*, the Base Definitions volume of  
15725 IEEE Std. 1003.1-200x, `<stdio.h>`, `<wchar.h>`, the Base Definitions volume of  
15726 IEEE Std. 1003.1-200x, Chapter 7, Locale

15727 **CHANGE HISTORY**

15728 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
15729 (E).

15730 **Issue 6**

15731 The Open Group corrigenda item U040/1 has been applied to the RETURN VALUE section,  
15732 describing the case if *n* or more wide characters are requested to be written using *swprintf()*.

15733 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

15734 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15735 • The prototypes for *fwprintf()*, *swprintf()*, and *wprintf()* are updated.
- 15736 • The DESCRIPTION is updated.



15737 **NAME**

15738 fwrite — binary output

15739 **SYNOPSIS**

15740 #include &lt;stdio.h&gt;

```
15741 size_t fwrite(const void *restrict ptr, size_t size, size_t nitems,
15742 FILE *restrict stream);
```

15743 **DESCRIPTION**

15744 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 15745 conflict between the requirements described here and the ISO C standard is unintentional. This  
 15746 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

15747 The *fwrite()* function shall write, from the array pointed to by *ptr*, up to *nitems* members whose  
 15748 size is specified by *size*, to the stream pointed to by *stream*. For each object, *size* calls are made to  
 15749 the *fputc()* function, taking the values (in order) from an array of **unsigned char** exactly  
 15750 overlaying the object. The file-position indicator for the stream (if defined) shall be advanced by  
 15751 the number of bytes successfully written. If an error occurs, the resulting value of the file-  
 15752 position indicator for the stream is indeterminate.

15753 CX The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the successful  
 15754 execution of *fwrite()* and the next successful completion of a call to *fflush()* or *fclose()* on the  
 15755 same stream, or a call to *exit()* or *abort()*.

15756 **RETURN VALUE**

15757 The *fwrite()* function shall return the number of members successfully written, which may be  
 15758 less than *nitems* if a write error is encountered. If *size* or *nitems* is 0, *fwrite()* shall return 0 and the  
 15759 state of the stream remains unchanged. Otherwise, if a write error occurs, the error indicator for  
 15760 CX the stream shall be set, and *errno* shall be set to indicate the error.

15761 **ERRORS**15762 Refer to *fputc()*.15763 **EXAMPLES**

15764 None.

15765 **APPLICATION USAGE**

15766 Because of possible differences in member length and byte ordering, files written using *fwrite()*  
 15767 are application-dependent, and possibly cannot be read using *fread()* by a different application  
 15768 or by the same application on a different processor.

15769 **RATIONALE**

15770 None.

15771 **FUTURE DIRECTIONS**

15772 None.

15773 **SEE ALSO**

15774 *ferror()*, *fopen()*, *printf()*, *putc()*, *puts()*, *write()*, the Base Definitions volume of  
 15775 IEEE Std. 1003.1-200x, <stdio.h>

15776 **CHANGE HISTORY**

15777 First released in Issue 1. Derived from Issue 1 of the SVID.

15778 **Issue 4**

15779 In the DESCRIPTION, the text is changed to make it clear that the function advances the file-  
 15780 position indicator by the number of bytes successfully written rather than the number of  
 15781 characters, which could include multi-byte sequences.

15782 The following change is incorporated for alignment with the ISO C standard:

- 15783
- The type of argument *ptr* is changed from **void\*** to **const void\***.

15784 **Issue 6**

15785 Extensions beyond the ISO C standard are now marked.

15786 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15787
- The *fwrite()* prototype is updated.
- 15788
- The DESCRIPTION is updated to clarify how the data is written out using *fputc()*.

15789 **NAME**

15790 fwscanf, swscanf, wscanf — convert formatted wide-character input

15791 **SYNOPSIS**

15792 #include &lt;stdio.h&gt;

15793 #include &lt;wchar.h&gt;

15794 int fwscanf(FILE \*restrict *stream*, const wchar\_t \*restrict *format*, ... );15795 int swscanf(const wchar\_t \*restrict *ws*,15796 const wchar\_t \*restrict *format*, ... );15797 int wscanf(const wchar\_t \**format*, ... );15798 **DESCRIPTION**

15799 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 15800 conflict between the requirements described here and the ISO C standard is unintentional. This  
 15801 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

15802 The *fwscanf()* function reads from the named input *stream*. The *wscanf()* function reads from the  
 15803 standard input stream *stdin*. The *swscanf()* function reads from the wide-character string *ws*.  
 15804 Each function reads wide characters, interprets them according to a format, and stores the  
 15805 results in its arguments. Each expects, as arguments, a control wide-character string *format*  
 15806 described below, and a set of *pointer* arguments indicating where the converted input should be  
 15807 stored. The result is undefined if there are insufficient arguments for the format. If the format is  
 15808 exhausted while arguments remain, the excess arguments are evaluated but are otherwise  
 15809 ignored.

15810 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 15811 to the next unused argument. In this case, the conversion wide character '%' (see below) is  
 15812 replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL\_ARGMAX}].  
 15813 This feature provides for the definition of format wide-character strings that select arguments in  
 15814 an order appropriate to specific languages. In format wide-character strings containing the  
 15815 "%n\$" form of conversion specifications, it is unspecified whether numbered arguments in the  
 15816 argument list can be referenced from the format wide-character string more than once.

15817 The *format* can contain either form of a conversion specification—that is, '%' or "%n\$"—but the  
 15818 two forms cannot normally be mixed within a single *format* wide-character string. The only  
 15819 exception to this is that "%%" or "%\*" can be mixed with the "%n\$" form.

15820 The *fwscanf()* function in all its forms allows for detection of a language-dependent radix  
 15821 character in the input string, encoded as a wide-character value. The radix character is defined in  
 15822 the program's locale (category *LC\_NUMERIC*). In the POSIX locale, or in a locale where the  
 15823 radix character is not defined, the radix character defaults to a period ('.').

15824 The *format* is a wide-character string composed of zero or more directives. Each directive is  
 15825 composed of one of the following: one or more white-space wide characters (<space>, <tab>,  
 15826 <newline>, <vertical-tab>, or <form-feed> characters); an ordinary wide character (neither '%'   
 15827 nor a white-space character); or a conversion specification. Each conversion specification is  
 15828 XSI introduced by a '%' or the sequence "%n\$" after which the following appear in sequence:

- 15829 • An optional assignment-suppressing character '\*'.
- 15830 • An optional non-zero decimal integer that specifies the maximum field width.
- 15831 • An optional length modifier that specifies the size of the receiving object.
- 15832 • A conversion wide character that specifies the type of conversion to be applied. The valid  
 15833 conversion wide characters are described below.

15834 The *fwscanf()* functions execute each directive of the format in turn. If a directive fails, as  
 15835 detailed below, the function shall return. Failures are described as input failures (due to the  
 15836 unavailability of input bytes) or matching failures (due to inappropriate input).

15837 A directive composed of one or more white-space wide characters is executed by reading input  
 15838 until no more valid input can be read, or up to the first wide character which is not a white-  
 15839 space wide character, which remains unread.

15840 A directive that is an ordinary wide character is executed as follows. The next wide character is  
 15841 read from the input and compared with the wide character that comprises the directive; if the  
 15842 comparison shows that they are not equivalent, the directive fails, and the differing and  
 15843 subsequent wide characters remain unread. Similarly, if end-of-file, an encoding error, or a read  
 15844 error prevents a wide character from being read, the directive fails.

15845 A directive that is a conversion specification defines a set of matching input sequences, as  
 15846 described below for each conversion wide character. A conversion specification is executed in  
 15847 the following steps.

15848 Input white-space wide characters (as specified by *iswspace()*) are skipped, unless the conversion  
 15849 specification includes a ' [ ', c, or n conversion character.

15850 An item is read from the input, unless the conversion specification includes an n conversion  
 15851 wide character. An input item is defined as the longest sequence of input wide characters, not  
 15852 exceeding any specified field width, which is an initial subsequence of a matching sequence. The  
 15853 first wide character, if any, after the input item remains unread. If the length of the input item is  
 15854 0, the execution of the conversion specification fails; this condition is a matching failure, unless  
 15855 end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is  
 15856 an input failure.

15857 Except in the case of a ' % ' conversion wide character, the input item (or, in the case of a %n  
 15858 conversion specification, the count of input wide characters) is converted to a type appropriate  
 15859 to the conversion wide character. If the input item is not a matching sequence, the execution of  
 15860 the conversion specification fails; this condition is a matching failure. Unless assignment  
 15861 suppression was indicated by a ' \* ', the result of the conversion is placed in the object pointed  
 15862 to by the first argument following the *format* argument that has not already received a  
 15863 XSI conversion result if the conversion specification is introduced by ' % ', or in the *n*th argument if  
 15864 introduced by the wide-character sequence "%n\$". If this object does not have an appropriate  
 15865 type, or if the result of the conversion cannot be represented in the space provided, the behavior  
 15866 is undefined.

15867 The length modifiers and their meanings are:

15868 *hh* Specifies that a following *d, i, o, u, x, X,* or *n* conversion specifier applies to an argument  
 15869 with type pointer to **signed char** or **unsigned char**.

15870 *h* Specifies that a following *d, i, o, u, x, X,* or *n* conversion specifier applies to an argument  
 15871 with type pointer to **short** or **unsigned short**.

15872 *l* (ell) Specifies that a following *d, i, o, u, x, X,* or *n* conversion specifier applies to an argument  
 15873 with type pointer to **long** or **unsigned long**; that a following *a, A, e, E, f, F, g,* or *G*  
 15874 conversion specifier applies to an argument with type pointer to **double**; or that a  
 15875 following *c, s,* or ' [ ' conversion specifier applies to an argument with type pointer to  
 15876 **wchar\_t**.

15877 *ll* (ell-ell) Specifies that a following *d, i, o, u, x, X,* or *n* conversion specifier applies to an argument  
 15878 with type pointer to **long long** or **unsigned long long**.

|       |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 15879 | <i>j</i>                                  | Specifies that a following <i>d</i> , <i>i</i> , <i>o</i> , <i>u</i> , <i>x</i> , <i>X</i> , or <i>n</i> conversion specifier applies to an argument with type pointer to <b>intmax_t</b> or <b>uintmax_t</b> .                                                                                                                                                                                                                                                                                                                                          |
| 15880 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15881 | <i>z</i>                                  | Specifies that a following <i>d</i> , <i>i</i> , <i>o</i> , <i>u</i> , <i>x</i> , <i>X</i> , or <i>n</i> conversion specifier applies to an argument with type pointer to <b>size_t</b> or the corresponding signed integer type.                                                                                                                                                                                                                                                                                                                        |
| 15882 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15883 | <i>t</i>                                  | Specifies that a following <i>d</i> , <i>i</i> , <i>o</i> , <i>u</i> , <i>x</i> , <i>X</i> , or <i>n</i> conversion specifier applies to an argument with type pointer to <b>ptrdiff_t</b> or the corresponding <b>unsigned</b> type.                                                                                                                                                                                                                                                                                                                    |
| 15884 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15885 | <i>L</i>                                  | Specifies that a following <i>a</i> , <i>A</i> , <i>e</i> , <i>E</i> , <i>f</i> , <i>F</i> , <i>g</i> , or <i>G</i> conversion specifier applies to an argument with type pointer to <b>long double</b> .                                                                                                                                                                                                                                                                                                                                                |
| 15886 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15887 |                                           | If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 15888 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15889 |                                           | The following conversion wide characters are valid:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 15890 | <i>d</i>                                  | Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>wcstol()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>int</b> .                                                                                                                                                                                                                                              |
| 15891 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15892 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15893 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15894 | <i>i</i>                                  | Matches an optionally signed integer, whose format is the same as expected for the subject sequence of <i>wcstol()</i> with 0 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>int</b> .                                                                                                                                                                                                                                                                 |
| 15895 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15896 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15897 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15898 | <i>o</i>                                  | Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of <i>wcstoul()</i> with the value 8 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>unsigned</b> .                                                                                                                                                                                                                                           |
| 15899 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15900 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15901 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15902 | <i>u</i>                                  | Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>wcstoul()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>unsigned</b> .                                                                                                                                                                                                                                        |
| 15903 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15904 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15905 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15906 | <i>x</i>                                  | Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <i>wcstoul()</i> with the value 16 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>unsigned</b> .                                                                                                                                                                                                                                    |
| 15907 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15908 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15909 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15910 | <i>a</i> , <i>e</i> , <i>f</i> , <i>g</i> | Matches an optionally signed floating-point number, infinity, or NaN whose format is the same as expected for the subject sequence of <i>wcstod()</i> . In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>float</b> .                                                                                                                                                                                                                                                                   |
| 15911 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15912 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15913 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15914 |                                           | If the <i>fwprintf()</i> family of functions generates character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std. 754-1985, the <i>fwscanf()</i> family of functions shall recognize them as input.                                                                                                                                                                                                                                                                                 |
| 15915 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15916 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15917 | <i>s</i>                                  | Matches a sequence of non white-space wide characters. If no <i>l</i> (ell) qualifier is present, characters from the input field are converted as if by repeated calls to the <i>wcrtomb()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first wide character is converted. The application shall ensure that the corresponding argument is a pointer to a character array large enough to accept the sequence and the terminating null character, which shall be added automatically. |
| 15918 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15919 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15920 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15921 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15922 |                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

|       |   |                                                                                                             |
|-------|---|-------------------------------------------------------------------------------------------------------------|
| 15923 |   | Otherwise, the application shall ensure that the corresponding argument is a pointer to                     |
| 15924 |   | an array of <b>wchar_t</b> large enough to accept the sequence and the terminating null wide                |
| 15925 |   | character, which shall be added automatically.                                                              |
| 15926 | [ | Matches a non-empty sequence of wide characters from a set of expected wide                                 |
| 15927 |   | characters (the <i>scanset</i> ). If no <i>l</i> (ell) qualifier is present, wide characters from the input |
| 15928 |   | field are converted as if by repeated calls to the <i>wcrtomb()</i> function, with the conversion           |
| 15929 |   | state described by an <b>mbstate_t</b> object initialized to zero before the first wide character           |
| 15930 |   | is converted. The application shall ensure that the corresponding argument is a pointer                     |
| 15931 |   | to a character array large enough to accept the sequence and the terminating null                           |
| 15932 |   | character, which shall be added automatically.                                                              |
| 15933 |   | If an <i>l</i> (ell) qualifier is present, the application shall ensure that the corresponding              |
| 15934 |   | argument is a pointer to an array of <b>wchar_t</b> large enough to accept the sequence and                 |
| 15935 |   | the terminating null wide character, which shall be added automatically.                                    |
| 15936 |   | The conversion specification includes all subsequent wide characters in the <i>format</i>                   |
| 15937 |   | string up to and including the matching right square bracket (']'). The wide                                |
| 15938 |   | characters between the square brackets (the <i>scanlist</i> ) comprise the scanset, unless the              |
| 15939 |   | wide character after the left square bracket is a circumflex ('^'), in which case the                       |
| 15940 |   | scanset contains all wide characters that do not appear in the scanlist between the                         |
| 15941 |   | circumflex and the right square bracket. If the conversion specification begins with                        |
| 15942 |   | "[ ]" or "[ ^ ]", the right square bracket is included in the scanlist and the next right                   |
| 15943 |   | square bracket is the matching right square bracket that ends the conversion                                |
| 15944 |   | specification; otherwise, the first right square bracket is the one that ends the                           |
| 15945 |   | conversion specification. If a '-' is in the scanlist and is not the first wide character,                  |
| 15946 |   | nor the second where the first wide character is a '^', nor the last wide character, the                    |
| 15947 |   | behavior is implementation-defined.                                                                         |
| 15948 | c | Matches a sequence of wide characters of exactly the number specified by the field                          |
| 15949 |   | width (1 if no field width is present in the conversion specification).                                     |
| 15950 |   | If no <i>l</i> (ell) length modifier is present, characters from the input field are converted as if        |
| 15951 |   | by repeated calls to the <i>wcrtomb()</i> function, with the conversion state described by an               |
| 15952 |   | <b>mbstate_t</b> object initialized to zero before the first wide character is converted. The               |
| 15953 |   | corresponding argument shall be a pointer to the initial element of a character array                       |
| 15954 |   | large enough to accept the sequence. No null character is added.                                            |
| 15955 |   | If an <i>l</i> (ell) length modifier is present, the corresponding argument shall be a pointer to           |
| 15956 |   | the initial element of an array of <b>wchar_t</b> large enough to accept the sequence. No null              |
| 15957 |   | wide character is added.                                                                                    |
| 15958 |   | Otherwise, the application shall ensure that the corresponding argument is a pointer to                     |
| 15959 |   | an array of <b>wchar_t</b> large enough to accept the sequence. No null wide character is                   |
| 15960 |   | added.                                                                                                      |
| 15961 | p | Matches an implementation-defined set of sequences, which shall be the same as the set                      |
| 15962 |   | of sequences that is produced by the <i>%p</i> conversion of the corresponding <i>fwprintf()</i>            |
| 15963 |   | functions. The application shall ensure that the corresponding argument is a pointer to                     |
| 15964 |   | a pointer to <b>void</b> . The interpretation of the input item is implementation-defined. If the           |
| 15965 |   | input item is a value converted earlier during the same program execution, the pointer                      |
| 15966 |   | that results shall compare equal to that value; otherwise, the behavior of the <i>%p</i>                    |
| 15967 |   | conversion is undefined.                                                                                    |
| 15968 | n | No input is consumed. The application shall ensure that the corresponding argument is                       |
| 15969 |   | a pointer to the integer into which is to be written the number of wide characters read                     |
| 15970 |   | from the input so far by this call to the <i>fwscanf()</i> functions. Execution of a <i>%n</i>              |

- 15971 conversion specification does not increment the assignment count returned at the  
 15972 completion of execution of the function. No argument is converted, but one is  
 15973 consumed. If the conversion specification includes an assignment-suppressing wide  
 15974 character or a field width, the behavior is undefined.
- 15975 XSI **C** Same as *lc*.
- 15976 **S** Same as *ls*.
- 15977 % Matches a single '%'; no conversion or assignment occurs. The complete conversion  
 15978 specification shall be "%%".
- 15979 If a conversion specification is invalid, the behavior is undefined.
- 15980 The conversion characters *A*, *E*, *F*, *G*, and *X* are also valid and behave the same as, respectively, *a*,  
 15981 *e*, *f*, *g*, and *x*.
- 15982 If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before  
 15983 any wide characters matching the current conversion specification (except for %*n*) have been  
 15984 read (other than leading white-space, where permitted), execution of the current conversion  
 15985 specification terminates with an input failure. Otherwise, unless execution of the current  
 15986 conversion specification is terminated with a matching failure, execution of the following  
 15987 conversion specification (if any) is terminated with an input failure.
- 15988 Reaching the end of the string in *swscanf()* is equivalent to encountering end-of-file for *fwscanf()*.
- 15989 If conversion terminates on a conflicting input, the offending input is left unread in the input.  
 15990 Any trailing white space (including <newline>) is left unread unless matched by a conversion  
 15991 specification. The success of literal matches and suppressed assignments is only directly  
 15992 determinable via the %*n* conversion specification.
- 15993 The *fwscanf()* and *wscanf()* functions may mark the *st\_atime* field of the file associated with  
 15994 *stream* for update. The *st\_atime* field shall be marked for update by the first successful execution  
 15995 of *fgetc()*, *fgetwc()*, *fgets()*, *fgetws()*, *fread()*, *getc()*, *getwc()*, *getchar()*, *getwchar()*, *gets()*, *fscanf()*,  
 15996 or *fwscanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.
- 15997 **RETURN VALUE**
- 15998 Upon successful completion, these functions shall return the number of successfully matched  
 15999 and assigned input items; this number can be 0 in the event of an early matching failure. If the  
 16000 input ends before the first matching failure or conversion, EOF shall be returned. If a read error  
 16001 CX occurs the error indicator for the stream is set, EOF shall be returned, and *errno* shall be set to  
 16002 indicate the error.
- 16003 **ERRORS**
- 16004 For the conditions under which the *fwscanf()* functions shall fail and may fail, refer to *fgetwc()*.
- 16005 In addition, *fwscanf()* may fail if:
- 16006 XSI **[EILSEQ]** Input byte sequence does not form a valid character.
- 16007 XSI **[EINVAL]** There are insufficient arguments.

16008 **EXAMPLES**

16009 The call:

```
16010 int i, n; float x; char name[50];
16011 n = wscanf(L"%d%f%s", &i, &x, name);
```

16012 with the input line:

16013 25 54.32E-1 Hamster

16014 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string  
16015 "Hamster".

16016 The call:

```
16017 int i; float x; char name[50];
16018 (void) wscanf(L"%2d%f*d %[0123456789]", &i, &x, name);
```

16019 with input:

16020 56789 0123 56a72

16021 assigns 56 to *i*, 789.0 to *x*, skip 0123, and place the string "56\0" in *name*. The next call to  
16022 *getchar()* shall return the character 'a'.16023 **APPLICATION USAGE**16024 In format strings containing the '%' form of conversion specifications, each argument in the  
16025 argument list is used exactly once.16026 **RATIONALE**

16027 None.

16028 **FUTURE DIRECTIONS**

16029 None.

16030 **SEE ALSO**

16031 *getwc()*, *fwprintf()*, *setlocale()*, *wcstod()*, *wcstol()*, *wcstoul()*, *wcrtomb()*, the Base Definitions  
16032 volume of IEEE Std. 1003.1-200x, <langinfo.h>, <stdio.h>, <wchar.h>, the Base Definitions  
16033 volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

16034 **CHANGE HISTORY**16035 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
16036 (E).16037 **Issue 6**

16038 The DESCRIPTION is updated to avoid use of the term "must" for application requirements.

16039 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 16040 • The prototypes for *fwscanf()* and *swscanf()* are updated.
- 16041 • The DESCRIPTION is updated.



16042 **NAME**

16043        *gai\_strerror* — address and name information error description

16044 **SYNOPSIS**

16045        #include <netdb.h>

16046        char \**gai\_strerror*(int *ecode*);

16047 **DESCRIPTION**

16048        The *gai\_strerror*() function shall return a text string describing an error that is listed in the  
16049        <netdb.h> header.

16050        When the *ecode* argument is one of the values listed in the <netdb.h> header, the function return  
16051        value points to a string describing the error. If the argument is not one of those values, the  
16052        function shall return a pointer to a string whose contents indicate an unknown error.

16053 **RETURN VALUE**

16054        Upon successful completion, *gai\_strerror*() shall return a pointer to an implementation-defined  
16055        string.

16056 **ERRORS**

16057        No errors are defined.

16058 **EXAMPLES**

16059        None.

16060 **APPLICATION USAGE**

16061        None.

16062 **RATIONALE**

16063        None.

16064 **FUTURE DIRECTIONS**

16065        None.

16066 **SEE ALSO**

16067        *getaddrinfo*(), the Base Definitions volume of IEEE Std. 1003.1-200x, <netdb.h>

16068 **CHANGE HISTORY**

16069        First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

16070 **NAME**

16071           gcvt — convert a floating-point number to a string (**LEGACY**)

16072 **SYNOPSIS**

16073 XSI       #include <stdlib.h>

16074       char \*gcvt(double *value*, int *ndigit*, char \**buf*);

16075

16076 **DESCRIPTION**

16077       Refer to *ecvt()*.

16078 **NAME**

16079           getaddrinfo — get address information

16080 **SYNOPSIS**

16081           #include &lt;sys/socket.h&gt;

16082           #include &lt;netdb.h&gt;

16083           int getaddrinfo(const char \*restrict *nodename*,16084                           const char \*restrict *servname*,16085                           const struct addrinfo \*restrict *hints*,16086                           struct addrinfo \*\*restrict *res*);16087 **DESCRIPTION**16088           Refer to *freeaddrinfo()*.

16089 **NAME**

16090           getc — get a byte from a stream

16091 **SYNOPSIS**

16092           #include <stdio.h>

16093           int getc(FILE \**stream*);

16094 **DESCRIPTION**

16095 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
16096 conflict between the requirements described here and the ISO C standard is unintentional. This  
16097 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

16098       The *getc()* function shall be equivalent to *fgetc()*, except that if it is implemented as a macro it  
16099 may evaluate *stream* more than once, so the argument should never be an expression with side  
16100 effects.

16101 **RETURN VALUE**

16102       Refer to *fgetc()*.

16103 **ERRORS**

16104       Refer to *fgetc()*.

16105 **EXAMPLES**

16106       None.

16107 **APPLICATION USAGE**

16108       If the integer value returned by *getc()* is stored into a variable of type **char** and then compared  
16109 against the integer constant EOF, the comparison may never succeed, because sign-extension of  
16110 a variable of type **char** on widening to integer is implementation-defined.

16111       Because it may be implemented as a macro, *getc()* may treat incorrectly a *stream* argument with  
16112 side effects. In particular, *getc(\*f++)* does not necessarily work as expected. Therefore, use of this  
16113 function should be preceded by "#undef getc" in such situations; *fgetc()* could also be used.

16114 **RATIONALE**

16115       None.

16116 **FUTURE DIRECTIONS**

16117       None.

16118 **SEE ALSO**

16119       *fgetc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>

16120 **CHANGE HISTORY**

16121       First released in Issue 1. Derived from Issue 1 of the SVID.

16122 **Issue 4**

16123       The words “a character variable” are replaced by “a variable of type **char**”, to emphasize the fact  
16124 that this function deals with byte values.

16125       The APPLICATION USAGE section now states that the use of this function is not recommended.

16126 **NAME**

16127        getc\_unlocked, getchar\_unlocked, putc\_unlocked, putchar\_unlocked — stdio with explicit client  
16128        locking

16129 **SYNOPSIS**

```
16130 TSF #include <stdio.h>

16131 int getc_unlocked(FILE *stream);
16132 int getchar_unlocked(void);
16133 int putc_unlocked(int c, FILE *stream);
16134 int putchar_unlocked(int c);
16135
```

16136 **DESCRIPTION**

16137        Versions of the functions *getc()*, *getchar()*, *putc()*, and *putchar()* respectively named  
16138        *getc\_unlocked()*, *getchar\_unlocked()*, *putc\_unlocked()*, and *putchar\_unlocked()* shall be provided  
16139        which are functionally identical to the original versions, with the exception that they are not  
16140        required to be implemented in a thread-safe manner. They may only safely be used within a  
16141        scope protected by *flockfile()* (or *ftrylockfile()*) and *funlockfile()*. These functions may safely be  
16142        used in a multi-threaded program if and only if they are called while the invoking thread owns  
16143        the (**FILE\***) object, as is the case after a successful call of the *flockfile()* or *ftrylockfile()* functions.

16144 **RETURN VALUE**

16145        See *getc()*, *getchar()*, *putc()*, and *putchar()*.

16146 **ERRORS**

16147        No errors are defined.

16148 **EXAMPLES**

16149        None.

16150 **APPLICATION USAGE**

16151        Because they may be implemented as macros, *getc\_unlocked()* and *putc\_unlocked()* may treat  
16152        incorrectly a stream argument with side effects. In particular, *getc\_unlocked(\*f++)* and  
16153        *putc\_unlocked(\*f++)* do not necessarily work as expected. Therefore, use of these functions in  
16154        such situations should be preceded by the following statement as appropriate:

```
16155 #undef getc_unlocked
16156 #undef putc_unlocked
```

16157 **RATIONALE**

16158        Some I/O functions are typically implemented as macros for performance reasons (for example,  
16159        *putc()* and *getc()*). For safety, they need to be synchronized, but it is often too expensive to  
16160        synchronize on every character. Nevertheless, it was felt that the safety concerns were more  
16161        important; consequently, the *getc()*, *getchar()*, *putc()*, and *putchar()* functions are required to be  
16162        thread-safe. However, unlocked versions are also provided with names that clearly indicate the  
16163        unsafe nature of their operation but can be used to exploit their higher performance. These  
16164        unlocked versions can be safely used only within explicitly locked program regions, using  
16165        exported locking primitives. In particular, a sequence such as:

```
16166 flockfile(fileptr);
16167 putc_unlocked('1', fileptr);
16168 putc_unlocked('\n', fileptr);
16169 fprintf(fileptr, "Line 2\n");
16170 funlockfile(fileptr);
```

16171        is permissible, and results in the text sequence:

16172 1  
16173 Line 2  
16174 being printed without being interspersed with output from other threads.

16175 It would be wrong to have the standard names such as *getc()*, *putc()*, and so on, map to the  
16176 “faster, but unsafe” rather than the “slower, but safe” versions. In either case, you would still  
16177 want to inspect all uses of *getc()*, *putc()*, and so on, by hand when converting existing code.  
16178 Choosing the safe bindings as the default, at least, results in correct code and maintains the  
16179 “atomicity at the function” invariant. To do otherwise would introduce gratuitous  
16180 synchronization errors into converted code. Other routines that modify the *stdio* (**FILE\***)  
16181 structures or buffers are also safely synchronized.

16182 Note that there is no need for functions of the form *getc\_locked()*, *putc\_locked()*, and so on, since  
16183 this is the functionality of *getc()*, *putc()*, *et al.* It would be inappropriate to use a feature test  
16184 macro to switch a macro definition of *getc()* between *getc\_locked()* and *getc\_unlocked()*, since the  
16185 ISO C standard requires an actual function to exist, a function whose behavior could not be  
16186 changed by the feature test macro. Also, providing both the *xxx\_locked()* and *xxx\_unlocked()*  
16187 forms leads to the confusion of whether the suffix describes the behavior of the function or the  
16188 circumstances under which it should be used.

16189 Three additional routines, *flockfile()*, *ftrylockfile()*, and *funlockfile()* (which may be macros), are  
16190 provided to allow the user to delineate a sequence of I/O statements that are executed  
16191 synchronously.

16192 The *ungetc()* function is infrequently called relative to the other functions/macros so no  
16193 unlocked variation is needed.

16194 **FUTURE DIRECTIONS**  
16195 None.

16196 **SEE ALSO**  
16197 *getc()*, *getchar()*, *putc()*, *putchar()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
16198 **<stdio.h>**

16199 **CHANGE HISTORY**  
16200 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

16201 **Issue 6**  
16202 These functions are marked as part of the Thread-Safe Functions option.

16203 The Open Group corrigenda item U030/2 has been applied adding APPLICATION USAGE  
16204 describing how applications should be written to avoid the case when the functions are  
16205 implemented as macros.

16206 **NAME**

16207            getchar — get a byte from a stdin stream

16208 **SYNOPSIS**

16209            #include <stdio.h>

16210            int getchar(void);

16211 **DESCRIPTION**

16212 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
16213            conflict between the requirements described here and the ISO C standard is unintentional. This  
16214            volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

16215            The *getchar()* function shall be equivalent to *getc(stdin)*.

16216 **RETURN VALUE**

16217            Refer to *fgetc()*.

16218 **ERRORS**

16219            Refer to *fgetc()*.

16220 **EXAMPLES**

16221            None.

16222 **APPLICATION USAGE**

16223            If the integer value returned by *getchar()* is stored into a variable of type **char** and then  
16224            compared against the integer constant EOF, the comparison may never succeed, because sign-  
16225            extension of a variable of type **char** on widening to integer is implementation-defined.

16226 **RATIONALE**

16227            None.

16228 **FUTURE DIRECTIONS**

16229            None.

16230 **SEE ALSO**

16231            *getc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>

16232 **CHANGE HISTORY**

16233            First released in Issue 1. Derived from Issue 1 of the SVID.

16234 **Issue 4**

16235            The words “a character variable” are replaced by “a variable of type **char**”, to emphasize the fact  
16236            that this function deals in byte values.

16237            The following change is incorporated for alignment with the ISO C standard:

- 16238            • The argument list is explicitly defined as **void**.

16239 **NAME**

16240 getcontext, setcontext — get and set current user context

16241 **SYNOPSIS**16242 XSI 

```
#include <ucontext.h>
```

16243 

```
int getcontext(ucontext_t *ucp);
```

16244 

```
int setcontext(const ucontext_t *ucp);
```

16245

16246 **DESCRIPTION**

16247 The *getcontext()* function shall initialize the structure pointed to by *ucp* to the current user  
16248 context of the calling thread. The **ucontext\_t** type that *ucp* points to defines the user context and  
16249 includes the contents of the calling thread's machine registers, the signal mask, and the current  
16250 execution stack.

16251 The *setcontext()* function shall restore the user context pointed to by *ucp*. A successful call to  
16252 *setcontext()* shall not return; program execution resumes at the point specified by the *ucp*  
16253 argument passed to *setcontext()*. The *ucp* argument should be created either by a prior call to  
16254 *getcontext()* or *makecontext()*, or by being passed as an argument to a signal handler. If the *ucp*  
16255 argument was created with *getcontext()*, program execution continues as if the corresponding  
16256 call of *getcontext()* had just returned. If the *ucp* argument was created with *makecontext()*,  
16257 program execution continues with the function passed to *makecontext()*. When that function  
16258 returns, the thread shall continue as if after a call to *setcontext()* with the *ucp* argument that was  
16259 input to *makecontext()*. If the *uc\_link* member of the **ucontext\_t** structure pointed to by the *ucp*  
16260 argument is equal to 0, then this context is the main context, and the thread shall exit when this  
16261 context returns. The effects of passing a *ucp* argument obtained from any other source are  
16262 unspecified.

16263 **RETURN VALUE**

16264 Upon successful completion, *setcontext()* shall not return and *getcontext()* shall return 0;  
16265 otherwise, a value of -1 shall be returned.

16266 **ERRORS**

16267 No errors are defined.

16268 **EXAMPLES**

16269 None.

16270 **APPLICATION USAGE**

16271 When a signal handler is executed, the current user context is saved and a new context is  
16272 created. If the thread leaves the signal handler via *longjmp()*, then it is unspecified whether the  
16273 context at the time of the corresponding *setjmp()* call is restored and thus whether future calls to  
16274 *getcontext()* provide an accurate representation of the current context, since the context restored  
16275 by *longjmp()* does not necessarily contain all the information that *setcontext()* requires. Signal  
16276 handlers should use *siglongjmp()* or *setcontext()* instead.

16277 Portable applications should not modify or access the *uc\_mcontext* member of **ucontext\_t**. A  
16278 portable application cannot assume that context includes any process-wide static data, possibly  
16279 including *errno*. Users manipulating contexts should take care to handle these explicitly when  
16280 required.

16281 Use of contexts to create alternate stacks is not defined by this volume of IEEE Std. 1003.1-200x.



16282 **RATIONALE**

16283           None.

16284 **FUTURE DIRECTIONS**

16285           None.

16286 **SEE ALSO**16287           *bsd\_signal()*, *makecontext()*, *setjmp()*, *sigaction()*, *sigaltstack()*, *sigprocmask()*, *sigsetjmp()*, the Base  
16288           Definitions volume of IEEE Std. 1003.1-200x, <**ucontext.h**>16289 **CHANGE HISTORY**

16290           First released in Issue 4, Version 2.

16291 **Issue 5**

16292           Moved from X/OPEN UNIX extension to BASE.

16293           The following sentence was removed from the DESCRIPTION: “If the *ucp* argument was passed  
16294           to a signal handler, program execution continues with the program instruction following the  
16295           instruction interrupted by the signal.”

16296 **NAME**

16297         getcwd — get the path name of the current working directory

16298 **SYNOPSIS**

16299         #include <unistd.h>

16300         char \*getcwd(char \*buf, size\_t size);

16301 **DESCRIPTION**

16302         The *getcwd()* function shall place an absolute path name of the current working directory in the  
16303         array pointed to by *buf*, and return *buf*. The path name copied to the array shall contain no  
16304         components that are symbolic links. The *size* argument is the size in bytes of the character array  
16305         pointed to by the *buf* argument. If *buf* is a null pointer, the behavior of *getcwd()* is unspecified.

16306 **RETURN VALUE**

16307         Upon successful completion, *getcwd()* shall return the *buf* argument. Otherwise, *getcwd()* shall  
16308         return a null pointer and set *errno* to indicate the error. The contents of the array pointed to by  
16309         *buf* are then undefined.

16310 **ERRORS**

16311         The *getcwd()* function shall fail if:

16312         [EINVAL]         The *size* argument is 0.

16313         [ERANGE]         The size argument is greater than 0, but is smaller than the length of the path  
16314         name +1.

16315         The *getcwd()* function may fail if:

16316         [EACCES]         Read or search permission was denied for a component of the path name.

16317         [ENOMEM]         Insufficient storage space is available.

16318 **EXAMPLES**16319         **Determining the Absolute Path Name of the Current Working Directory**

16320         The following example returns a pointer to an array that holds the absolute path name of the  
16321         current working directory. The pointer is returned in the *ptr* variable, which points to the *buf*  
16322         array where the path name is stored.

16323         #include <stdlib.h>

16324         #include <unistd.h>

16325         ...

16326         long size;

16327         char \*buf;

16328         char \*ptr;

16329         size = pathconf(".", \_PC\_PATH\_MAX);

16330         if ((buf = (char \*)malloc((size\_t)size)) != NULL)

16331             ptr = getcwd(buf, (size\_t)size);

16332         ...

16333 **APPLICATION USAGE**

16334         None.

16335 **RATIONALE**

16336 Since the maximum path name length is arbitrary unless `{PATH_MAX}` is defined, an  
 16337 application generally cannot supply a *buf* with *size* `{{PATH_MAX} + 1}`.

16338 Having `getcwd()` take no arguments and instead use the `malloc()` function to produce space for  
 16339 the returned argument was considered. The advantage is that `getcwd()` knows how big the  
 16340 working directory path name is and can allocate an appropriate amount of space. But the  
 16341 programmer would have to use the `free()` function to free the resulting object, or each use of  
 16342 `getcwd()` would further reduce the available memory. Also, `malloc()` and `free()` are used nowhere  
 16343 else in this volume of IEEE Std. 1003.1-200x. Finally, `getcwd()` is taken from the SVID where it  
 16344 has the two arguments used in this volume of IEEE Std. 1003.1-200x.

16345 The older function `getwd()` was rejected for use in this context because it had only a buffer  
 16346 argument and no *size* argument, and thus had no way to prevent overwriting the buffer, except  
 16347 to depend on the programmer to provide a large enough buffer.

16348 On some implementations, if *buf* is a null pointer, `getcwd()` may obtain *size* bytes of memory  
 16349 using `malloc()`. In this case, the pointer returned by `getcwd()` may be used as the argument in a  
 16350 subsequent call to `free()`. Invoking `getcwd()` with *buf* as a null pointer is not recommended in  
 16351 portable applications.

16352 If a program is operating in a directory where some (grand)parent directory does not permit  
 16353 reading, `getcwd()` may fail, as in most implementations it must read the directory to determine  
 16354 the name of the file. This can occur if search, but not read, permission is granted in an  
 16355 intermediate directory, or if the program is placed in that directory by some more privileged  
 16356 process (for example, login). Including the [EACCES] error condition makes the reporting of the  
 16357 error consistent and warns the application writer that `getcwd()` can fail for reasons beyond the  
 16358 control of the application writer or user. Some implementations can avoid this occurrence (for  
 16359 example, by implementing `getcwd()` using `pwd`, where `pwd` is a set-user-root process), thus the  
 16360 error was made optional.

16361 Because this volume of IEEE Std. 1003.1-200x permits the addition of other errors, this would be  
 16362 a common addition and yet one that applications could not be expected to deal with without  
 16363 this addition.

16364 **FUTURE DIRECTIONS**

16365 None.

16366 **SEE ALSO**

16367 `malloc()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<unistd.h>`

16368 **CHANGE HISTORY**

16369 First released in Issue 1. Derived from Issue 1 of the SVID.

16370 **Issue 4**

16371 The `<unistd.h>` header is added to the SYNOPSIS section.

16372 The [ENOMEM] error is marked as an extension.

16373 The words “as this functionality may be subject to withdrawal” have been deleted from the end  
 16374 of the last sentence in the APPLICATION USAGE section.

16375 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 16376 • The DESCRIPTION is changed to indicate that the effects of passing a null pointer in *buf* are  
 16377 undefined.

16378 **Issue 6**

16379 The following new requirements on POSIX implementations derive from alignment with the  
16380 Single UNIX Specification:

- 16381
- The [ENOMEM] optional error condition is added.

16382 **NAME**

16383 getdate — convert user format date and time

16384 **SYNOPSIS**

```
16385 xSI #include <time.h>
16386 struct tm *getdate(const char *string);
16387
```

16388 **DESCRIPTION**

16389 The *getdate()* function shall convert a string representation of a date or time into a broken-down  
 16390 time.

16391 The external variable or macro *getdate\_err* is used by *getdate()* to return error values.

16392 Templates are used to parse and interpret the input string. The templates are contained in a text  
 16393 file identified by the environment variable *DATEMSK*. The *DATEMSK* variable should be set to  
 16394 indicate the full path name of the file that contains the templates. The first line in the template  
 16395 that matches the input specification is used for interpretation and conversion into the internal  
 16396 time format.

16397 The following field descriptors are supported:

|       |    |                                                                                     |
|-------|----|-------------------------------------------------------------------------------------|
| 16398 | %% | Same as ' % '.                                                                      |
| 16399 | %a | Abbreviated weekday name.                                                           |
| 16400 | %A | Full weekday name.                                                                  |
| 16401 | %b | Abbreviated month name.                                                             |
| 16402 | %B | Full month name.                                                                    |
| 16403 | %c | Locale's appropriate date and time representation.                                  |
| 16404 | %C | Century number (00-99; leading zeros are permitted but not required).               |
| 16405 | %d | Day of month (01-31; the leading 0 is optional).                                    |
| 16406 | %D | Date as %m/%d/%y.                                                                   |
| 16407 | %e | Same as %d.                                                                         |
| 16408 | %h | Abbreviated month name.                                                             |
| 16409 | %H | Hour (00-23).                                                                       |
| 16410 | %I | Hour (01-12).                                                                       |
| 16411 | %m | Month number (01-12).                                                               |
| 16412 | %M | Minute (00-59).                                                                     |
| 16413 | %n | Same as <newline>.                                                                  |
| 16414 | %p | Locale's equivalent of either AM or PM.                                             |
| 16415 | %r | The locale's appropriate representation of time in AM and PM notation. In the POSIX |
| 16416 |    | locale, this is equivalent to %I:%M:%S %p.                                          |
| 16417 | %R | Time as %H:%M.                                                                      |
| 16418 | %S | Seconds (00-61). Leap seconds are allowed but are not predictable through use of    |
| 16419 |    | algorithms.                                                                         |

|       |    |                                                                                                             |
|-------|----|-------------------------------------------------------------------------------------------------------------|
| 16420 | %t | Same as <tab>.                                                                                              |
| 16421 | %T | Time as %H:%M:%S.                                                                                           |
| 16422 | %w | Weekday number (Sunday = 0-6).                                                                              |
| 16423 | %x | Locale's appropriate date representation.                                                                   |
| 16424 | %X | Locale's appropriate time representation.                                                                   |
| 16425 | %y | Year within century. When a century is not otherwise specified, values in the range                         |
| 16426 |    | 69-99 refer to years in the twentieth century (1969 to 1999 inclusive); values in the range                 |
| 16427 |    | 00-68 refer to years in the twenty-first century (2000 to 2068 inclusive).                                  |
| 16428 | %Y | Year as cyy (for example, 1994).                                                                            |
| 16429 | %Z | Timezone name or no characters if no timezone exists. If the timezone supplied by %Z                        |
| 16430 |    | is not the timezone that <i>getdate()</i> expects, an invalid input specification error shall               |
| 16431 |    | result. The <i>getdate()</i> function calculates an expected timezone based on information                  |
| 16432 |    | supplied to the function (such as the hour, day, and month).                                                |
| 16433 |    | The match between the template and input specification performed by <i>getdate()</i> is case-               |
| 16434 |    | insensitive.                                                                                                |
| 16435 |    | The month and weekday names can consist of any combination of upper and lowercase letters.                  |
| 16436 |    | The process can request that the input date or time specification be in a specific language by              |
| 16437 |    | setting the <i>LC_TIME</i> category (see <i>setlocale()</i> ).                                              |
| 16438 |    | Leading 0s are not necessary for the descriptors that allow leading 0s. However, at most two                |
| 16439 |    | digits are allowed for those descriptors, including leading 0s. Extra whitespace in either the              |
| 16440 |    | template file or in <i>string</i> is ignored.                                                               |
| 16441 |    | The field descriptors %c, %x, and %X shall not be supported if they include unsupported field               |
| 16442 |    | descriptors.                                                                                                |
| 16443 |    | The following rules apply for converting the input specification into the internal format:                  |
| 16444 |    | • If %Z is being scanned, then <i>getdate()</i> initializes the broken-down time to be the current time     |
| 16445 |    | in the scanned timezone. Otherwise, it initializes the broken-down time based on the current                |
| 16446 |    | local time as if <i>localtime()</i> had been called.                                                        |
| 16447 |    | • If only the weekday is given, today is assumed if the given day is equal to the current day               |
| 16448 |    | and next week if it is less,                                                                                |
| 16449 |    | • If only the month is given, the current month is assumed if the given month is equal to the               |
| 16450 |    | current month and next year if it is less, and no year is given (the first day of month is                  |
| 16451 |    | assumed if no day is given),                                                                                |
| 16452 |    | • If no hour, minute, and second are given the current hour, minute, and second are assumed,                |
| 16453 |    | • If no date is given, today is assumed if the given hour is greater than the current hour and              |
| 16454 |    | tomorrow is assumed if it is less.                                                                          |
| 16455 |    | If a field descriptor specification in the DATEMSK file does not correspond to one of the field             |
| 16456 |    | descriptors above, the behavior is unspecified.                                                             |
| 16457 |    | The <i>getdate()</i> function need not be reentrant. A function that is not required to be reentrant is not |
| 16458 |    | required to be thread-safe.                                                                                 |

16459 **RETURN VALUE**

16460 Upon successful completion, `getdate()` shall return a pointer to a **struct tm**. Otherwise, it shall  
 16461 return a null pointer and set `getdate_err` to indicate the error.

16462 **ERRORS**

16463 The `getdate()` function shall fail in the following cases, setting `getdate_err` to the value shown in  
 16464 the list below. Any changes to `errno` are unspecified.

- 16465 1. The `DATEMSK` environment variable is null or undefined.
- 16466 2. The template file cannot be opened for reading.
- 16467 3. Failed to get file status information.
- 16468 4. The template file is not a regular file.
- 16469 5. An I/O error is encountered while reading the template file.
- 16470 6. Memory allocation failed (not enough memory available).
- 16471 7. There is no line in the template that matches the input.
- 16472 8. Invalid input specification. For example, February 31; or a time is specified that cannot be  
 16473 represented in a `time_t` (representing the time in seconds since the Epoch).

16474 **EXAMPLES**

- 16475 1. The following example shows the possible contents of a template:

```
16476 %m
16477 %A %B %d, %Y, %H:%M:%S
16478 %A
16479 %B
16480 %m/%d/%y %I %p
16481 %d,%m,%Y %H:%M
16482 at %A the %dst of %B in %Y
16483 run job at %I %p,%B %dnd
16484 %A den %d. %B %Y %H.%M Uhr
```

- 16485 2. The following are examples of valid input specifications for the template in Example 1:

```
16486 getdate("10/1/87 4 PM");
16487 getdate("Friday");
16488 getdate("Friday September 18, 1987, 10:30:30");
16489 getdate("24,9,1986 10:30");
16490 getdate("at monday the 1st of december in 1986");
16491 getdate("run job at 3 PM, december 2nd");
```

16492 If the `LC_TIME` category is set to a German locale that includes *freitag* as a weekday name  
 16493 and *oktober* as a month name, the following would be valid:

```
16494 getdate("freitag den 10. oktober 1986 10.30 Uhr");
```

- 16495 3. The following example shows how local date and time specification can be defined in the  
 16496 template:

16497  
16498  
16499  
16500  
16501  
16502

| Invocation                 | Line in Template |
|----------------------------|------------------|
| getdate("11/27/86")        | %m/%d/%y         |
| getdate("27.11.86")        | %d.%m.%y         |
| getdate("86-11-27")        | %y-%m-%d         |
| getdate("Friday 12:00:00") | %A %H:%M:%S      |

16503  
16504  
16505  
16506

4. The following examples help to illustrate the above rules assuming that the current date is Mon Sep 22 12:19:47 EDT 1986 and the *LC\_TIME* category is set to the default C locale:

16507  
16508  
16509  
16510  
16511  
16512  
16513  
16514  
16515  
16516  
16517  
16518  
16519  
16520

| Input        | Line in Template | Date                         |
|--------------|------------------|------------------------------|
| Mon          | %a               | Mon Sep 22 12:19:47 EDT 1986 |
| Sun          | %a               | Sun Sep 28 12:19:47 EDT 1986 |
| Fri          | %a               | Fri Sep 26 12:19:47 EDT 1986 |
| September    | %B               | Mon Sep 1 12:19:47 EDT 1986  |
| January      | %B               | Thu Jan 1 12:19:47 EST 1987  |
| December     | %B               | Mon Dec 1 12:19:47 EST 1986  |
| Sep Mon      | %b %a            | Mon Sep 1 12:19:47 EDT 1986  |
| Jan Fri      | %b %a            | Fri Jan 2 12:19:47 EST 1987  |
| Dec Mon      | %b %a            | Mon Dec 1 12:19:47 EST 1986  |
| Jan Wed 1989 | %b %a %Y         | Wed Jan 4 12:19:47 EST 1989  |
| Fri 9        | %a %H            | Fri Sep 26 09:00:00 EDT 1986 |
| Feb 10:30    | %b %H:%S         | Sun Feb 1 10:00:30 EST 1987  |
| 10:30        | %H:%M            | Tue Sep 23 10:30:00 EDT 1986 |
| 13:30        | %H:%M            | Mon Sep 22 13:30:00 EDT 1986 |

16521 **APPLICATION USAGE**

16522 Although historical versions of *getdate()* did not require that **<time.h>** declare the external  
16523 variable *getdate\_err*, this volume of IEEE Std. 1003.1-200x does require it. The Open Group  
16524 encourages applications to remove declarations of *getdate\_err* and instead incorporate the  
16525 declaration by including **<time.h>**.

16526 Applications should use %Y (4-digit years) in preference to %y (2-digit years).

16527 **RATIONALE**

16528 None.

16529 **FUTURE DIRECTIONS**

16530 None.

16531 **SEE ALSO**

16532 *ctime()*, *localtime()*, *setlocale()*, *strftime()*, *times()*, the Base Definitions volume of  
16533 IEEE Std. 1003.1-200x, **<time.h>**

16534 **CHANGE HISTORY**

16535 First released in Issue 4, Version 2.

16536 **Issue 5**

16537 Moved from X/OPEN UNIX extension to BASE.

16538 The last paragraph of the DESCRIPTION is added.

16539 The %C specifier is added, and the exact meaning of the %y specifier is clarified in the  
16540 DESCRIPTION.

16541 A note indicating that this function need not be reentrant is added to the DESCRIPTION.



16542 The %R specifier is changed to follow historical practice.

16543 **Issue 6**

16544 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since  
16545 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time* |  
16546 functions.

16547 **NAME**

16548           getegid — get the effective group ID

16549 **SYNOPSIS**

16550           #include <unistd.h>

16551           gid\_t getegid(void);

16552 **DESCRIPTION**

16553           The *getegid()* function shall return the effective group ID of the calling process.

16554 **RETURN VALUE**

16555           The *getegid()* function is always successful and no return value is reserved to indicate an error.

16556 **ERRORS**

16557           No errors are defined.

16558 **EXAMPLES**

16559           None.

16560 **APPLICATION USAGE**

16561           None.

16562 **RATIONALE**

16563           None.

16564 **FUTURE DIRECTIONS**

16565           None.

16566 **SEE ALSO**

16567           *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base  
16568           Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>, <unistd.h>

16569 **CHANGE HISTORY**

16570           First released in Issue 1. Derived from Issue 1 of the SVID.

16571 **Issue 4**

16572           The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
16573           XSI-conformant systems.

16574           The <unistd.h> header is added to the SYNOPSIS section.

16575           The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 16576
  - The argument list is explicitly defined as **void**.

16577 **Issue 6**

16578           In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

16579           The following new requirements on POSIX implementations derive from alignment with the  
16580           Single UNIX Specification:

- 16581
  - The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
16582           required for conforming implementations of previous POSIX specifications, it was not  
16583           required for UNIX applications.

16584 **NAME**

16585            getenv — get value of an environment variable

16586 **SYNOPSIS**

16587            #include &lt;stdlib.h&gt;

16588            char \*getenv(const char \*name);

16589 **DESCRIPTION**

16590 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
 16591 conflict between the requirements described here and the ISO C standard is unintentional. This  
 16592 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

16593        The *getenv()* function shall search the environment of the calling process (see the Base  
 16594 Definitions volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables) for the  
 16595 environment variable *name* if it exists and return a pointer to the value of the environment  
 16596 variable. If the specified environment variable cannot be found, a null pointer shall be returned.  
 16597 The application shall ensure that it does not modify the string pointed to by the *getenv()*  
 16598 function. The string pointed to may be overwritten by a subsequent call to *getenv()*, *setenv()*,  
 16599 **XSI** *unsetenv()*, or *putenv()* but shall not be overwritten by a call to any other function in this volume  
 16600 of IEEE Std. 1003.1-200x.

16601        If the application modifies *environ* or the pointers to which it points, the behavior of *getenv()* is  
 16602 undefined.

16603 **CX**        The *getenv()* function need not be reentrant. A function that is not required to be reentrant is not  
 16604 required to be thread-safe.

16605 **RETURN VALUE**

16606        Upon successful completion, *getenv()* shall return a pointer to a string containing the *value* for  
 16607 the specified *name*. If the specified name cannot be found in the environment of the calling  
 16608 process, a null pointer shall be returned.

16609        The return value from *getenv()* may point to static data which may be overwritten by  
 16610 **XSI** subsequent calls to *getenv()*, *setenv()*, *unsetenv()*, or *putenv()*.

16611 **ERRORS**

16612        No errors are defined.

16613 **EXAMPLES**16614        **Getting the Value of an Environment Variable**16615        The following example gets the value of the *HOME* environment variable.

```
16616 #include <stdlib.h>
16617 ...
16618 const char *name = "HOME";
16619 char *value;
16620 value = getenv(name);
```

16621 **APPLICATION USAGE**

16622        None.

16623 **RATIONALE**

16624        The *clearenv()* function was considered but rejected. The *putenv()* function has now been  
 16625 included for alignment with the Single UNIX Specification.

16626 The *getenv()* function is inherently not reentrant because it returns a value pointing to static  
16627 data.

16628 Conforming applications are required not to modify *environ* directly, but to use only the  
16629 functions described here to manipulate the process environment as an abstract object. Thus, the  
16630 implementation of the environment access functions has complete control over the data  
16631 structure used to represent the environment (subject to the requirement that *environ* be  
16632 maintained as a list of strings with embedded equal signs for applications that wish to scan the  
16633 environment). This constraint allows the implementation to properly manage the memory it  
16634 allocates, either by using allocated storage for all variables (copying them on the first invocation  
16635 of *setenv()* or *unsetenv()*), or keeping track of which strings are currently in allocated space and  
16636 which are not, via a separate table or some other means. This enables the implementation to free  
16637 any allocated space used by strings (and perhaps the pointers to them) stored in *environ* when  
16638 *unsetenv()* is called. A C runtime start-up procedure (that which invokes *main()* and perhaps  
16639 initializes *environ*) can also initialize a flag indicating that none of the environment has yet been  
16640 copied to allocated storage, or that the separate table has not yet been initialized.

16641 In fact, for higher performance of *getenv()*, the implementation could also maintain a separate  
16642 copy of the environment in a data structure that could be searched much more quickly (such as  
16643 an indexed hash table, or a binary tree), and update both it and the linear list at *environ* when  
16644 *setenv()* or *unsetenv()* is invoked.

16645 Performance of *getenv()* can be important for applications which have large numbers of  
16646 environment variables. Typically, applications like this use the environment as a resource  
16647 database of user-configurable parameters. The fact that these variables are in the user's shell  
16648 environment usually means that any other program that uses environment variables (such as *ls*,  
16649 which attempts to use *COLUMNS*, or really almost any utility (*LANG*, *LC\_ALL*, and so on) is  
16650 similarly slowed down by the linear search through the variables.

16651 An implementation that maintains separate data structures, or even one that manages the  
16652 memory it consumes, is not currently required as it was thought it would reduce consensus  
16653 among implementors who do not want to change their historical implementations.

16654 The POSIX Threads Extension states that multi-threaded applications must not modify *environ*  
16655 directly, and that IEEE Std. 1003.1-200x is providing functions which such applications can use  
16656 in the future to manipulate the environment in a thread-safe manner. Thus, moving away from  
16657 application use of *environ* is desirable from that standpoint as well.

#### 16658 FUTURE DIRECTIONS

16659 None.

#### 16660 SEE ALSO

16661 *exec*, *putenv()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<stdlib.h>`, the Base  
16662 Definitions volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables

#### 16663 CHANGE HISTORY

16664 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 16665 Issue 4

16666 The DESCRIPTION is updated to indicate that the return string must not be modified by an  
16667 application, may be overwritten by subsequent calls to *getenv()* or *putenv()*, and is not  
16668 overwritten by calls to other XSI system interfaces.

16669 A reference to *putenv()* has also been added to the APPLICATION USAGE section.

16670 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 16671           • The type of argument *name* is changed from **char\*** to **const char\***.
- 16672 **Issue 5**
- 16673           Normative text previously in the APPLICATION USAGE section is moved to the RETURN
- 16674           VALUE section.
- 16675           A note indicating that this function need not be reentrant is added to the DESCRIPTION.
- 16676 **Issue 6**
- 16677           The following changes were made to align with the IEEE P1003.1a draft standard:
- 16678           • References added to the new *setenv()* and *unsetenv()* functions.
- 16679           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

16680 **NAME**

16681           geteuid — get the effective user ID

16682 **SYNOPSIS**

16683           #include <unistd.h>

16684           uid\_t geteuid(void);

16685 **DESCRIPTION**

16686           The *geteuid()* function shall return the effective user ID of the calling process.

16687 **RETURN VALUE**

16688           The *geteuid()* function is always successful and no return value is reserved to indicate an error.

16689 **ERRORS**

16690           No errors are defined.

16691 **EXAMPLES**

16692           None.

16693 **APPLICATION USAGE**

16694           None.

16695 **RATIONALE**

16696           None.

16697 **FUTURE DIRECTIONS**

16698           None.

16699 **SEE ALSO**

16700           *getegid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base  
16701           Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>, <unistd.h>

16702 **CHANGE HISTORY**

16703           First released in Issue 1. Derived from Issue 1 of the SVID.

16704 **Issue 4**

16705           The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
16706           XSI-conformant systems.

16707           The <unistd.h> header is added to the SYNOPSIS section.

16708           The following change is incorporated for alignment with the ISO POSIX-1 standard:

16709           

- The argument list is explicitly defined as **void**.

16710 **Issue 6**

16711           In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

16712           The following new requirements on POSIX implementations derive from alignment with the  
16713           Single UNIX Specification:

16714           

- The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
16715           required for conforming implementations of previous POSIX specifications, it was not  
16716           required for UNIX applications.

16717 **NAME**

16718           getgid — get the real group ID

16719 **SYNOPSIS**

16720           #include <unistd.h>

16721           gid\_t getgid(void);

16722 **DESCRIPTION**

16723           The *getgid()* function shall return the real group ID of the calling process.

16724 **RETURN VALUE**

16725           The *getgid()* function is always successful and no return value is reserved to indicate an error.

16726 **ERRORS**

16727           No errors are defined.

16728 **EXAMPLES**

16729           None.

16730 **APPLICATION USAGE**

16731           None.

16732 **RATIONALE**

16733           None.

16734 **FUTURE DIRECTIONS**

16735           None.

16736 **SEE ALSO**

16737           *getegid()*, *geteuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base

16738           Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>, <unistd.h>

16739 **CHANGE HISTORY**

16740           First released in Issue 1. Derived from Issue 1 of the SVID.

16741 **Issue 4**

16742           The <sys/types.h> header is now marked as optional (OH); this header need not be included on XSI-conformant systems.

16744           The <unistd.h> header is added to the SYNOPSIS section.

16745           The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 16746
  - The argument list is explicitly defined as **void**.

16747 **Issue 6**

16748           In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

16749           The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 16751
  - The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- 16752
- 16753

16754 **NAME**

16755           getgrent — get the group database entry

16756 **SYNOPSIS**

16757 xSI       #include <grp.h>

16758           struct group \*getgrent(void);

16759

16760 **DESCRIPTION**

16761           Refer to *endgrent()*.



## 16762 NAME

16763 getgrgid, getgrgid\_r — get group database entry for a group ID

## 16764 SYNOPSIS

16765 #include &lt;grp.h&gt;

16766 struct group \*getgrgid(gid\_t gid);

16767 TSF int getgrgid\_r(gid\_t gid, struct group \*grp, char \*buffer,

16768 size\_t bufsize, struct group \*\*result);

16769

## 16770 DESCRIPTION

16771 The *getgrgid()* function shall search the group database for an entry with a matching *gid*.16772 The *getgrgid()* function need not be reentrant. A function that is not required to be reentrant is  
16773 not required to be thread-safe.16774 TSF The *getgrgid\_r()* function updates the **group** structure pointed to by *grp* and stores a pointer to  
16775 that structure at the location pointed to by *result*. The structure contains an entry from the  
16776 group database with a matching *gid*. Storage referenced by the group structure is allocated from  
16777 the memory provided with the *buffer* parameter, which is *bufsize* characters in size. The  
16778 maximum size needed for this buffer can be determined with the `{_SC_GETGR_R_SIZE_MAX}`  
16779 *sysconf()* parameter. A NULL pointer is returned at the location pointed to by *result* on error or if  
16780 the requested entry is not found.

## 16781 RETURN VALUE

16782 Upon successful completion, *getgrgid()* shall return a pointer to a **struct group** with the structure  
16783 defined in `<grp.h>` with a matching entry if one is found. The *getgrgid()* function shall return a  
16784 null pointer if either the requested entry was not found, or an error occurred. On error, *errno*  
16785 shall be set to indicate the error.16786 The return value may point to a static area which is overwritten by a subsequent call to  
16787 *getgrent()*, *getgrgid()*, or *getgrnam()*.16788 TSF If successful, the *getgrgid\_r()* function shall return zero; otherwise, an error number shall be  
16789 returned to indicate the error.

## 16790 ERRORS

16791 The *getgrgid()* and *getgrgid\_r()* functions may fail if:

16792 [EIO] An I/O error has occurred.

16793 [EINTR] A signal was caught during *getgrgid()*.16794 [EMFILE] `{OPEN_MAX}` file descriptors are currently open in the calling process.

16795 [ENFILE] The maximum allowable number of files is currently open in the system.

16796 TSF The *getgrgid\_r()* function may fail if:16797 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to  
16798 be referenced by the resulting **group** structure.

16799 **EXAMPLES**16800 **Finding an Entry in the Group Database**

16801 The following example uses *getgrgid()* to search the group database for a group ID that was  
 16802 previously stored in a **stat** structure, then prints out the group name if it is found. If the group is  
 16803 not found, the program prints the numeric value of the group for the entry.

```
16804 #include <sys/types.h>
16805 #include <grp.h>
16806 #include <stdio.h>
16807 ...
16808 struct stat statbuf;
16809 struct group *grp;
16810 ...
16811 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
16812 printf(" %-8.8s", grp->gr_name);
16813 else
16814 printf(" %-8d", statbuf.st_gid);
16815 ...
```

16816 **APPLICATION USAGE**

16817 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrgid()*.  
 16818 If *errno* is set on return, an error occurred.

16819 The *getgrgid\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
 16820 of possibly using a static data area that may be overwritten by each call.

16821 **RATIONALE**

16822 None.

16823 **FUTURE DIRECTIONS**

16824 None.

16825 **SEE ALSO**

16826 *endgrent()*, *getgrnam()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<grp.h>**,  
 16827 **<limits.h>**, **<sys/types.h>**

16828 **CHANGE HISTORY**

16829 First released in Issue 1. Derived from System V Release 2.0.

16830 **Issue 4**

16831 The DESCRIPTION is clarified.

16832 In the RETURN VALUE section, the reference to the setting of *errno* is marked as an extension.

16833 The errors [EIO], [EINTR], [EMFILE], and [ENFILE] are marked as extensions.

16834 A note is added to the APPLICATION USAGE section advising how applications should check  
 16835 for errors.

16836 The **<sys/types.h>** header is added as optional (OH); this header need not be included on XSI-  
 16837 conformant systems.

16838 **Issue 5**

16839 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 16840 VALUE section.

16841 The *getgrgid\_r()* function is included for alignment with the POSIX Threads Extension.

- 16842 A note indicating that the *getgrgid()* function need not be reentrant is added to the  
16843 DESCRIPTION.
- 16844 **Issue 6**
- 16845 The *getgrgid\_r()* function is marked as part of the Thread-Safe Functions option.
- 16846 The Open Group corrigenda item U028/3 has been applied correcting text in the DESCRIPTION  
16847 describing matching the *gid*.
- 16848 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.
- 16849 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.
- 16850 The following new requirements on POSIX implementations derive from alignment with the  
16851 Single UNIX Specification:
- 16852 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
16853 required for conforming implementations of previous POSIX specifications, it was not  
16854 required for UNIX applications.
  - 16855 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
  - 16856 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.
- 16857 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
16858 its avoidance of possibly using a static data area.

## 16859 NAME

16860 getgrnam, getgrnam\_r — search group database for a name

## 16861 SYNOPSIS

16862 #include <grp.h>

16863 struct group \*getgrnam(const char \*name);

16864 TSF int getgrnam\_r(const char \*name, struct group \*grp, char \*buffer,

16865 size\_t bufsize, struct group \*\*result);

16866

## 16867 DESCRIPTION

16868 The *getgrnam()* function shall search the group database for an entry with a matching *name*.

16869 The *getgrnam()* function need not be reentrant. A function that is not required to be reentrant is  
16870 not required to be thread-safe.

16871 TSF The *getgrnam\_r()* function updates the **group** structure pointed to by *grp* and stores a pointer to  
16872 that structure at the location pointed to by *result*. The structure contains an entry from the  
16873 group database with a matching *gid* or *name*. Storage referenced by the group structure is  
16874 allocated from the memory provided with the *buffer* parameter, which is *bufsize* characters in  
16875 size. The maximum size needed for this buffer can be determined with the  
16876 `{_SC_GETGR_R_SIZE_MAX} sysconf()` parameter. A NULL pointer is returned at the location  
16877 pointed to by *result* on error or if the requested entry is not found.

## 16878 RETURN VALUE

16879 The *getgrnam()* function shall return a pointer to a **struct group** with the structure defined in  
16880 <grp.h> with a matching entry if one is found. The *getgrnam()* function shall return a null  
16881 pointer if either the requested entry was not found, or an error occurred. On error, *errno* shall be  
16882 set to indicate the error.

16883 The return value may point to a static area which is overwritten by a subsequent call to  
16884 *getgrent()*, *getgrgid()*, or *getgrnam()*.

16885 TSF If successful, the *getgrnam\_r()* function shall return zero; otherwise, an error number shall be  
16886 returned to indicate the error.

## 16887 ERRORS

16888 The *getgrnam()* and *getgrnam\_r()* functions may fail if:

16889 [EIO] An I/O error has occurred.

16890 [EINTR] A signal was caught during *getgrnam()*.

16891 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

16892 [ENFILE] The maximum allowable number of files is currently open in the system.

16893 The *getgrnam\_r()* function may fail if:

16894 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to  
16895 be referenced by the resulting **group** structure.

16896 **EXAMPLES**

16897       None.

16898 **APPLICATION USAGE**

16899       Applications wishing to check for error situations should set *errno* to 0 before calling *getgrnam()*.  
 16900       If *errno* is set on return, an error occurred.

16901       The *getgrnam\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
 16902       of possibly using a static data area that may be overwritten by each call.

16903 **RATIONALE**

16904       None.

16905 **FUTURE DIRECTIONS**

16906       None.

16907 **SEE ALSO**

16908       *endgrent()*, *getgrgid()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<grp.h>**, **<limits.h>**,  
 16909       **<sys/types.h>**

16910 **CHANGE HISTORY**

16911       First released in Issue 1. Derived from System V Release 2.0.

16912 **Issue 4**

16913       The DESCRIPTION is clarified.

16914       The **<sys/types.h>** header is added as optional (OH); this header need not be included on XSI-  
 16915       conformant systems.

16916       In the RETURN VALUE section, reference to the setting of *errno* is marked as an extension.

16917       The errors [EIO], [EINTR], [EMFILE], and [ENFILE] are marked as extensions.

16918       A note is added to the APPLICATION USAGE section advising how applications should check  
 16919       for errors.

16920       The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 16921       • The type of argument *name* is changed from **char\*** to **const char\***.

16922 **Issue 5**

16923       Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 16924       VALUE section.

16925       The *getgrnam\_r()* function is included for alignment with the POSIX Threads Extension.

16926       A note indicating that the *getgrnam()* function need not be reentrant is added to the  
 16927       DESCRIPTION.

16928 **Issue 6**16929       The *getgrnam\_r()* function is marked as part of the Thread-Safe Functions option.

16930       In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

16931       In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

16932       The following new requirements on POSIX implementations derive from alignment with the  
 16933       Single UNIX Specification:

- 16934       • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was  
 16935       required for conforming implementations of previous POSIX specifications, it was not  
 16936       required for UNIX applications.

- 16937           • In the RETURN VALUE section, the requirement to set *errno* on error is added.
- 16938           • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.
- 16939           The APPLICATION USAGE section is updated to include a note on the thread-safe function and
- 16940           its avoidance of possibly using a static data area.

16941 **NAME**

16942           getgroups — get supplementary group IDs

16943 **SYNOPSIS**

16944           #include &lt;unistd.h&gt;

16945           int getgroups(int *gidsetsize*, gid\_t *grouplist*[]);16946 **DESCRIPTION**

16947           The *getgroups()* function fills in the array *grouplist* with the current supplementary group IDs of the calling process. It is implementation-defined whether *getgroups()* also returns the effective group ID in the *grouplist* array.

16950           The *gidsetsize* argument specifies the number of elements in the array *grouplist*. The actual number of group IDs stored in the array is returned. The values of array entries with indices greater than or equal to the value returned are undefined.

16953           If *gidsetsize* is 0, *getgroups()* shall return the number of group IDs that it would otherwise return without modifying the array pointed to by *grouplist*.

16955           If the effective group ID of the process is returned with the supplementary group IDs, the value returned shall always be greater than or equal to one and less than or equal to the value of {NGROUPS\_MAX}+1.

16958 **RETURN VALUE**

16959           Upon successful completion, the number of supplementary group IDs shall be returned. A return value of -1 indicates failure and *errno* shall be set to indicate the error.

16961 **ERRORS**16962           The *getgroups()* function shall fail if:

|       |          |                                                                                  |
|-------|----------|----------------------------------------------------------------------------------|
| 16963 | [EINVAL] | The <i>gidsetsize</i> argument is non-zero and less than the number of group IDs |
| 16964 |          | that would have been returned.                                                   |

16965 **EXAMPLES**16966           **Getting the Supplementary Group IDs of the Calling Process**

16967           The following example places the current supplementary group IDs of the calling process into the *group* array.

```
16969 #include <sys/types.h>
16970 #include <unistd.h>
16971 ...
16972 gid_t *group;
16973 int nogroups;
16974 long ngroups_max;

16975 ngroups_max = sysconf(_SC_NGROUPS_MAX);
16976 group = (gid_t *)malloc(ngroups_max *sizeof(gid_t));

16977 ngroups = getgroups(ngroups_max, group);
```

16978 **APPLICATION USAGE**

16979           None.

16980 **RATIONALE**

16981           The related function *setgroups()* is a privileged operation and therefore is not covered by this volume of IEEE Std. 1003.1-200x.

16983 As implied by the definition of supplementary groups, the effective group ID may appear in the  
16984 array returned by *getgroups()* or it may be returned only by *getegid()*. Duplication may exist, but  
16985 the application needs to call *getegid()* to be sure of getting all of the information. Various  
16986 implementation variations and administrative sequences cause the set of groups appearing in  
16987 the result of *getgroups()* to vary in order and as to whether the effective group ID is included,  
16988 even when the set of groups is the same (in the mathematical sense of “set”). (The history of a  
16989 process and its parents could affect the details of result.)

16990 Applications writers should note that {NGROUPS\_MAX} is not necessarily a constant on all  
16991 implementations.

#### 16992 FUTURE DIRECTIONS

16993 None.

#### 16994 SEE ALSO

16995 *getegid()*, *setgid()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>,  
16996 <unistd.h>

#### 16997 CHANGE HISTORY

16998 First released in Issue 3.

16999 Entry included for alignment with the POSIX.1-1988 standard.

#### 17000 Issue 4

17001 The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
17002 XSI-conformant systems.

17003 The <unistd.h> header is added to the SYNOPSIS section.

17004 The following change is incorporated for alignment with the FIPS requirements:

- 17005 • A return value of 0 is no longer permitted, because {NGROUPS\_MAX} cannot be 0.

#### 17006 Issue 5

17007 Normative text previously in the APPLICATION USAGE section is moved to the  
17008 DESCRIPTION.

#### 17009 Issue 6

17010 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

17011 The following new requirements on POSIX implementations derive from alignment with the  
17012 Single UNIX Specification:

- 17013 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
17014 required for conforming implementations of previous POSIX specifications, it was not  
17015 required for UNIX applications.

- 17016 • A return value of 0 is not permitted, because {NGROUPS\_MAX} cannot be 0. This is a FIPS  
17017 requirement.

17018 The following changes were made to align with the IEEE P1003.1a draft standard:

- 17019 • Explanation added that the effective group ID may be included in the supplementary group  
17020 list.



17021 **NAME**

17022 gethostbyaddr (**LEGACY**), gethostbyname (**LEGACY**), getipnodebyaddr, getipnodebyname, —  
 17023 network host database functions

17024 **SYNOPSIS**

```
17025 #include <netdb.h>

17026 struct hostent *gethostbyaddr(const void *addr, socklen_t len,
17027 int type);
17028 struct hostent *gethostbyname(const char *name);
17029 struct hostent *getipnodebyaddr(const void *restrict addr, socklen_t len, |
17030 int type, int *restrict error_num);
17031 struct hostent *getipnodebyname(const char *name, int type, int flags, |
17032 int *error_num);
```

17033 **DESCRIPTION**

17034 These functions enable applications to retrieve information about hosts. This information is  
 17035 considered to be stored in a database that can be accessed sequentially or randomly.  
 17036 Implementation of this database is unspecified.

17037 **Note:** In many cases it is implemented by the Domain Name System, as documented in  
 17038 RFC 1034, RFC 1035, and RFC 1886.

17039 Entries are returned in **hostent** structures.

17040 The *gethostbyaddr()* function shall return an entry containing addresses of address family *type* for  
 17041 the host with address *addr*. *len* contains the length of the address pointed to by *addr*. The  
 17042 *gethostbyaddr()* function need not be reentrant. A function that is not required to be reentrant is  
 17043 not required to be thread-safe.

17044 The *gethostbyname()* function shall return an entry containing addresses of address family  
 17045 AF\_INET for the host with name *name*. The *gethostbyname()* function need not be reentrant. A  
 17046 function that is not required to be reentrant is not required to be thread-safe.

17047 The *getipnodebyaddr()* function shall return the entry containing addresses of address family *type*  
 17048 for the host with address *addr*, opening a connection to the database if necessary. The *len*  
 17049 argument contains the length of the address pointed to by *addr*. If an error occurs, the  
 17050 appropriate error code is returned in *error\_num*. The *getipnodebyaddr()* function is thread-safe.

17051 The *getipnodebyname()* function shall return the entry containing addresses of address family  
 17052 *type* for the host with name *name*, opening a connection to the database if necessary. The *flags*  
 17053 argument affects what information is returned. If an error occurs, the appropriate error code is  
 17054 returned in *error\_num*. The *getipnodebyname()* function is thread-safe.

17055 The *addr* argument of *gethostbyaddr()* or *getipnodebyaddr()* shall be an **in\_addr** structure when  
 17056 IP6 *type* is AF\_INET, and shall be an **in6\_addr** structure when *type* is AF\_INET6. It contains a binary  
 17057 format (that is, not null-terminated) address in network byte order. The *gethostbyaddr()* function  
 17058 is not guaranteed to return addresses of address families other than AF\_INET, even when such  
 17059 addresses exist in the database.

17060 If *gethostbyaddr()* or *getipnodebyaddr()* returns a record, then its *h\_addrtype* field is the same as the  
 17061 *type* argument that was passed to the function, and its *h\_addr\_list* field lists a single address that  
 17062 IP6 is a copy of the *addr* argument that was passed to the function. If *type* is AF\_INET6 and *addr* is  
 17063 an IPv4-mapped IPv6 address or an IPv4-compatible IPv6 address, then the *h\_name* and *h\_aliases*  
 17064 fields are those that would have been returned for address family AF\_INET and address equal to  
 17065 the last four bytes of *addr*.

17066 If *gethostbyaddr()* or *getipnodebyaddr()* are called with *addr* containing the IPv6 unspecified  
 17067 address (all bytes zero), then no query is performed and the function fails with  
 17068 [HOST\_NOT\_FOUND].

17069 The *name* argument of *getipnodebyname()* shall be either a node name or a numeric address  
 17070 string. For IPv4, a numeric address string shall be in the dotted-decimal notation described in  
 17071 IP6 *inet\_addr()*. For IPv6, a numeric address string shall be in one of the standard IPv6 text forms  
 17072 described in *inet\_ntop()*. The *name* argument of *gethostbyname()* shall be a node name; the  
 17073 behavior of *gethostbyname()* when passed a numeric address string is unspecified.

17074 IP6 If *name* is a dotted-decimal IPv4 address and *af* equals AF\_INET, or *name* is an IPv6 hex address  
 17075 and *af* equals AF\_INET6, the members of the returned **hostent** structure are as follows:

17076 *h\_name* Points to a copy of the *name* argument.

17077 *h\_aliases* A NULL pointer.

17078 *h\_addrtype* A copy of the *type* argument.

17079 IP6 *h\_length* Either 4 (for AF\_INET) or 16 (for AF\_INET6).

17080 IP6 *h\_addr\_list*[0] A pointer to the 4-byte or 16-byte binary address.

17081 *h\_addr\_list*[1] A NULL pointer.

17082 IP6 If *name* is a dotted-decimal IPv4 address and *af* equals AF\_INET6 and AI\_V4MAPPED is set in  
 17083 *flags*, an IPv4-mapped IPv6 address is returned, and the members are as follows:

17084 *h\_name* Points to an IPv6 hex address containing the IPv4-mapped IPv6 address.

17085 *h\_aliases* A NULL pointer.

17086 *h\_addrtype* AF\_INET6.

17087 *h\_length* 16.

17088 *h\_addr\_list*[0] A pointer to the 16-byte binary address.

17089 *h\_addr\_list*[1] A NULL pointer.

17090 If *name* is a dotted-decimal IPv4 address and *af* equals AF\_INET6 and AI\_V4MAPPED is not set,  
 17091 then NULL is returned with [HOST\_NOT\_FOUND].

17092 It is an error when *name* is an IPv6 hex address and *af* equals AF\_INET. The function's return  
 17093 value is a NULL pointer with the [HOST\_NOT\_FOUND] error.

17094 If *name* is not a numeric address string and is an alias for a valid host name, then *gethostbyname()*  
 17095 or *getipnodebyname()* return information about the host name to which the alias refers, and *name*  
 17096 is included in the list of aliases returned.

17097 If *name* is a node name, then operation of the *getipnodebyname()* function is modified by the value  
 17098 of the *flags* argument, as follows:

17099 • If *flags* is 0 and *type* is AF\_INET, then a query is made for IPv4 addresses. If it is successful,  
 17100 the IPv4 addresses are returned and the *h\_length* member of the **hostent** structure shall have  
 17101 IP6 a value of 4. Otherwise, the function shall return a NULL pointer.

17102 • If *flags* is 0 and if *type* is AF\_INET6, then a query is made for IPv6 addresses. If it is successful,  
 17103 the IPv6 addresses are returned and the *h\_length* member of the **hostent** structure shall have  
 17104 a value of 16. If unsuccessful, the function shall return a NULL pointer.

17105 • If the AI\_V4MAPPED flag is set and *type* is AF\_INET6, then a query is made for IPv6  
 17106 addresses. If it is successful, the IPv6 addresses are returned, and no query is made for IPv4

17107 addresses. If it is not successful, a query is made for IPv4 addresses and any found are  
 17108 returned as IPv4-mapped IPv6 addresses. *h\_length* shall have a value of 16 in either case of  
 17109 addresses being returned. The AI\_V4MAPPED flag is ignored unless *type* is AF\_INET6.

- 17110 • If the AI\_ALL and AI\_V4MAPPED flags are both set and *type* is AF\_INET6, then a query is  
 17111 made for IPv6 addresses, and any found are returned. Another query is then made for IPv4  
 17112 addresses, and any found are returned as IPv4-mapped IPv6 addresses, and *h\_length* is 16.  
 17113 Only if both queries fail does the function return a NULL pointer. This flag is ignored unless  
 17114 *type* is AF\_INET6.
- 17115 • The AI\_ADDRCONFIG flag specifies that a query for IPv6 addresses should be made only if  
 17116 the node has at least one IPv6 source address configured, and that a query for IPv4 addresses  
 17117 should be made only if the node has at least one IPv4 source address configured.
- 17118 • If the AI\_V4MAPPED and AI\_ADDRCONFIG flags are both set and *type* is AF\_INET6, then:
  - 17119 — If the node has at least one IPv6 source address configured, a query is made for IPv6  
 17120 addresses.
  - 17121 — If it is successful, the IPv6 addresses are returned and no query is made for IPv4  
 17122 addresses.
  - 17123 — If the node has no IPv6 source address configured, or if the query for IPv6 addresses is not  
 17124 successful, then if the node has at least one IPv4 source address configured, a query is  
 17125 made for IPv4 addresses and any found are returned as IPv4-mapped IPv6 addresses.

17126 *h\_length* shall have a value of 16 in either case of addresses being returned.

- 17127 • Macro AI\_DEFAULT is defined as the logical OR of AI\_V4MAPPED and AI\_ADDRCONFIG.

17128 **Note:** It is intended that setting *flags* to AI\_DEFAULT be appropriate for most  
 17129 applications.

17130 **RETURN VALUE**

17131 Upon successful completion, these functions shall return a pointer to a **hostent** structure if the  
 17132 requested entry was found, and a null pointer if the end of the database was reached or the  
 17133 requested entry was not found.

17134 Upon unsuccessful completion, *getipnodebyaddr()* and *getipnodebyname()* shall set their *error\_num*  
 17135 argument to indicate the error, while *gethostbyaddr()* and *gethostbyname()* shall set *h\_errno* to  
 17136 indicate it.

17137 **ERRORS**

17138 These functions shall fail in the following cases. The *getipnodebyaddr()* and *getipnodebyname()*  
 17139 functions shall return the value shown in the list below in *error\_num*; the *gethostbyaddr()* and  
 17140 *gethostbyname()* functions shall set *h\_errno* to that value. Any changes to *errno* are unspecified.

17141 [HOST\_NOT\_FOUND]  
 17142 No such host is known.

17143 [NO\_DATA] The server recognized the request and the name, but no address is available.  
 17144 Another type of request to the name server for the domain might return an  
 17145 answer.

17146 [NO\_RECOVERY]  
 17147 An unexpected server failure occurred which cannot be recovered.

17148 [TRY\_AGAIN] A temporary and possibly transient error occurred, such as a failure of a  
 17149 server to respond.

17150 **EXAMPLES**

17151       None.

17152 **APPLICATION USAGE**

17153       The **hostent** structure returned by *getipnodebyaddr()* and *getipnodebyname()*, and any structures  
17154       pointed to from those structures, are dynamically allocated. Applications should call  
17155       *freehostent()* to free the memory used by these structures.

17156       The *gethostbyaddr()*, and *gethostbyname()* functions may return pointers to static data, which may  
17157       be overwritten by subsequent calls to any of these functions. Applications shall not call  
17158       *freehostent()* for this area.

17159       The *getipnodebyaddr()* function is preferred over the *gethostbyaddr()* function.

17160       The *getipnodebyname()* function is preferred over the *gethostbyname()* function.

17161 **RATIONALE**

17162       None.

17163 **FUTURE DIRECTIONS**

17164       The *gethostbyaddr()* and *gethostbyname()* functions may be withdrawn in a future version.

17165 **SEE ALSO**

17166       *endhostent()*, *freehostent()*, *endservent()*, *inet\_addr()*, the Base Definitions volume of  
17167       IEEE Std. 1003.1-200x, <netdb.h>

17168 **CHANGE HISTORY**

17169       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17170       The **restrict** keyword is added to the *getipnodebyaddr()* prototype for alignment with the  
17171       ISO/IEC 9899:1999 standard.

17172 **NAME**

17173       gethostbyname — network host database functions

17174 **SYNOPSIS**

17175       #include <netdb.h>

17176       struct hostent \*gethostbyname(const char \*name);

17177 **DESCRIPTION**

17178       Refer to *gethostbyaddr()*.

17179 **NAME**

17180           gethostent — network host database functions

17181 **SYNOPSIS**

17182           #include <netdb.h>

17183           struct hostent \*gethostent(void);

17184 **DESCRIPTION**

17185           Refer to *endhostent()*.

17186 **NAME**

17187           gethostid — get an identifier for the current host

17188 **SYNOPSIS**

17189 XSI       #include <unistd.h>

17190           long gethostid(void);

17191

17192 **DESCRIPTION**

17193           The *gethostid()* function retrieves a 32-bit identifier for the current host.

17194 **RETURN VALUE**

17195           Upon successful completion, *gethostid()* shall return an identifier for the current host.

17196 **ERRORS**

17197           No errors are defined.

17198 **EXAMPLES**

17199           None.

17200 **APPLICATION USAGE**

17201           This volume of IEEE Std. 1003.1-200x does not define the domain in which the return value is  
17202           unique.

17203 **RATIONALE**

17204           None.

17205 **FUTURE DIRECTIONS**

17206           None.

17207 **SEE ALSO**

17208           *random()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <unistd.h>

17209 **CHANGE HISTORY**

17210           First released in Issue 4, Version 2.

17211 **Issue 5**

17212           Moved from X/OPEN UNIX extension to BASE.

17213 **NAME**

17214       gethostname — get name of current host

17215 **SYNOPSIS**

17216       #include <unistd.h>

17217       int gethostname(char \*name, socklen\_t namelen);

17218 **DESCRIPTION**

17219       The *gethostname()* function shall return the standard host name for the current machine. The  
17220       *namelen* argument shall specify the size of the array pointed to by the *name* argument. The  
17221       returned name shall be null-terminated, except that if *namelen* is an insufficient length to hold  
17222       the host name, then the returned name shall be truncated and it is unspecified whether the  
17223       returned name is null-terminated.

17224       Host names are limited to 255 bytes.

17225 **RETURN VALUE**

17226       Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned.

17227 **ERRORS**

17228       No errors are defined.

17229 **EXAMPLES**

17230       None.

17231 **APPLICATION USAGE**

17232       None.

17233 **RATIONALE**

17234       None.

17235 **FUTURE DIRECTIONS**

17236       None.

17237 **SEE ALSO**

17238       *gethostid()*, *uname()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <unistd.h>

17239 **CHANGE HISTORY**

17240       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.



17241 **NAME**

17242       getipnodebyaddr — network host database functions

17243 **SYNOPSIS**

17244       #include &lt;netdb.h&gt;

17245       struct hostent \*getipnodebyaddr(const void \*restrict *addr*, socklen\_t *len*, |  
17246                   int *type*, int \*restrict *error\_num*); |17247 **DESCRIPTION**17248       Refer to *gethostbyaddr()*. |

17249 **NAME**

17250       getipnodebyname — network host database functions

17251 **SYNOPSIS**

17252       #include <netdb.h>

17253       struct hostent \*getipnodebyname(const char \*name, int type, int flags,  
17254                                   int \*error\_num);

17255 **DESCRIPTION**

17256       Refer to *gethostbyaddr()*.

17257 **NAME**

17258 getitimer, setitimer — get or set value of interval timer

17259 **SYNOPSIS**

```
17260 xsi #include <sys/time.h>
17261
17261 int getitimer(int which, struct itimerval *value);
17262 int setitimer(int which, const struct itimerval *restrict value,
17263 struct itimerval *restrict ovalue);
17264
```

17265 **DESCRIPTION**

17266 The *getitimer()* function shall store the current value of the timer specified by *which* into the  
 17267 structure pointed to by *value*. The *setitimer()* function shall set the timer specified by *which* to  
 17268 the value specified in the structure pointed to by *value*, and if *ovalue* is not a null pointer, stores  
 17269 the previous value of the timer in the structure pointed to by *ovalue*.

17270 A timer value is defined by the **itimerval** structure, specified in `<sys/time.h>`. If *it\_value* is non-  
 17271 zero, it shall indicate the time to the next timer expiration. If *it\_interval* is non-zero, it shall  
 17272 specify a value to be used in reloading *it\_value* when the timer expires. Setting *it\_value* to 0 shall  
 17273 disable a timer, regardless of the value of *it\_interval*. Setting *it\_interval* to 0 shall disable a timer  
 17274 after its next expiration (assuming *it\_value* is non-zero).

17275 Implementations may place limitations on the granularity of timer values. For each interval  
 17276 timer, if the requested timer value requires a finer granularity than the implementation supports,  
 17277 the actual timer value shall be rounded up to the next supported value.

17278 An XSI-conforming implementation provides each process with at least three interval timers,  
 17279 which are indicated by the *which* argument:

|       |                |                                                                                                                                                                                                                                                                                        |
|-------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 17280 | ITIMER_REAL    | Decrements in real time. A SIGALRM signal is delivered when this timer expires.                                                                                                                                                                                                        |
| 17281 |                |                                                                                                                                                                                                                                                                                        |
| 17282 | ITIMER_VIRTUAL | Decrements in process virtual time. It runs only when the process is executing. A SIGVTALRM signal is delivered when it expires.                                                                                                                                                       |
| 17283 |                |                                                                                                                                                                                                                                                                                        |
| 17284 | ITIMER_PROF    | Decrements both in process virtual time and when the system is running on behalf of the process. It is designed to be used by interpreters in statistically profiling the execution of interpreted programs. Each time the ITIMER_PROF timer expires, the SIGPROF signal is delivered. |
| 17285 |                |                                                                                                                                                                                                                                                                                        |
| 17286 |                |                                                                                                                                                                                                                                                                                        |
| 17287 |                |                                                                                                                                                                                                                                                                                        |

17288 The interaction between *setitimer()* and any of *alarm()*, *sleep()*, or *usleep()* is unspecified.

17289 **RETURN VALUE**

17290 Upon successful completion, *getitimer()* or *setitimer()* shall return 0; otherwise, `-1` shall be  
 17291 returned and *errno* set to indicate the error.

17292 **ERRORS**

17293 The *setitimer()* function shall fail if:

|       |          |                                                                                                                                                                                                        |
|-------|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 17294 | [EINVAL] | The <i>value</i> argument is not in canonical form. (In canonical form, the number of microseconds is a non-negative integer less than 1,000,000 and the number of seconds is a non-negative integer.) |
| 17295 |          |                                                                                                                                                                                                        |
| 17296 |          |                                                                                                                                                                                                        |

17297 The *getitimer()* and *setitimer()* functions may fail if:

|       |          |                                              |
|-------|----------|----------------------------------------------|
| 17298 | [EINVAL] | The <i>which</i> argument is not recognized. |
|-------|----------|----------------------------------------------|

17299 **EXAMPLES**

17300 None.

17301 **APPLICATION USAGE**

17302 None.

17303 **RATIONALE**

17304 None.

17305 **FUTURE DIRECTIONS**

17306 None.

17307 **SEE ALSO**17308 *alarm()*, *sleep()*, *timer\_getoverrun()*, *ualarm()*, *usleep()*, the Base Definitions volume of

17309 IEEE Std. 1003.1-200x, &lt;signal.h&gt;, &lt;sys/time.h&gt;

17310 **CHANGE HISTORY**

17311 First released in Issue 4, Version 2.

17312 **Issue 5**

17313 Moved from X/OPEN UNIX extension to BASE.

17314 **Issue 6**17315 The **restrict** keyword is added to the *setitimer()* prototype for alignment with the

17316 ISO/IEC 9899:1999 standard.

17317 **NAME**17318 `getlogin, getlogin_r` — get login name17319 **SYNOPSIS**17320 `#include <unistd.h>`17321 `char *getlogin(void);`17322 TSF `int getlogin_r(char *name, size_t namesize);`

17323

17324 **DESCRIPTION**

17325 The `getlogin()` function shall return a pointer to a string containing the user name associated by  
 17326 the login activity with the controlling terminal of the current process. If `getlogin()` returns a non-  
 17327 null pointer, then that pointer points to the name that the user logged in under, even if there are  
 17328 several login names with the same user ID.

17329 The `getlogin()` function need not be reentrant. A function that is not required to be reentrant is  
 17330 not required to be thread-safe.

17331 TSF The `getlogin_r()` function puts the name associated by the login activity with the controlling  
 17332 terminal of the current process in the character array pointed to by `name`. The array is `namesize`  
 17333 characters long and should have space for the name and the terminating null character. The  
 17334 maximum size of the login name is `{LOGIN_NAME_MAX}`.

17335 If `getlogin_r()` is successful, `name` points to the name the user used at login, even if there are  
 17336 several login names with the same user ID.

17337 **RETURN VALUE**

17338 Upon successful completion, `getlogin()` shall return a pointer to the login name or a null pointer  
 17339 if the user's login name cannot be found. Otherwise, it shall return a null pointer and set `errno` to  
 17340 indicate the error.

17341 The return value from `getlogin()` may point to static data whose content is overwritten by each  
 17342 call.

17343 TSF If successful, the `getlogin_r()` function shall return zero; otherwise, an error number shall be  
 17344 returned to indicate the error.

17345 **ERRORS**

17346 The `getlogin()` and `getlogin_r()` functions may fail if:

17347 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

17348 [ENFILE] The maximum allowable number of files is currently open in the system.

17349 [ENXIO] The calling process has no controlling terminal.

17350 The `getlogin_r()` function may fail if:

17351 TSF [ERANGE] The value of `namesize` is smaller than the length of the string to be returned  
 17352 including the terminating null character.

## 17353 EXAMPLES

17354 **Getting the User Login Name**

17355 The following example calls the *getlogin()* function to obtain the name of the user associated  
 17356 with the calling process, and passes this information to the *getpwnam()* function to get the  
 17357 associated user database information.

```

17358 #include <unistd.h>
17359 #include <sys/types.h>
17360 #include <pwd.h>
17361 #include <stdio.h>
17362 ...
17363 char *lgn;
17364 struct passwd *pw;
17365 ...
17366 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
17367 fprintf(stderr, "Get of user information failed.\n"); exit(1);
17368 }
```

17369 **APPLICATION USAGE**

17370 Three names associated with the current process can be determined: *getpwuid(geteuid())* shall  
 17371 return the name associated with the effective user ID of the process; *getlogin()* shall return the  
 17372 name associated with the current login activity; and *getpwuid(getuid())* shall return the name  
 17373 associated with the real user ID of the process.

17374 The *getlogin\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
 17375 of possibly using a static data area that may be overwritten by each call.

17376 **RATIONALE**

17377 The *getlogin()* function returns a pointer to the user's login name. The same user ID may be  
 17378 shared by several login names. If it is desired to get the user database entry that is used during  
 17379 login, the result of *getlogin()* should be used to provide the argument to the *getpwnam()*  
 17380 function. (This might be used to determine the user's login shell, particularly where a single user  
 17381 has multiple login shells with distinct login names, but the same user ID.)

17382 The information provided by the *cuserid()* function, which was originally defined in the  
 17383 POSIX.1-1988 standard and subsequently removed, can be obtained by the following:

```
17384 getpwuid(geteuid())
```

17385 while the information provided by historical implementations of *cuserid()* can be obtained by:

```
17386 getpwuid(getuid())
```

17387 The thread-safe version of this function places the user name in a user-supplied buffer and  
 17388 returns a non-zero value if it fails. The non-thread-safe version may return the name in a static  
 17389 data area that may be overwritten by each call.

17390 **FUTURE DIRECTIONS**

17391 None.

17392 **SEE ALSO**

17393 *getpwnam()*, *getpwuid()*, *geteuid()*, *getuid()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
 17394 <limits.h>, <unistd.h>

17395 **CHANGE HISTORY**

17396 First released in Issue 1. Derived from System V Release 2.0.

17397 **Issue 4**

17398 The `<unistd.h>` header is added to the SYNOPSIS section.

17399 In the RETURN VALUE section, reference to the setting of *errno* is marked as an extension.

17400 The behavior of the function when the login name cannot be found is included in the RETURN  
17401 VALUE section instead of the DESCRIPTION.

17402 The errors [EMFILE], [ENFILE], and [ENXIO] are marked as extensions.

17403 The APPLICATION USAGE section is changed to refer to *getpwuid()* rather than *cuserid()*, which  
17404 may be withdrawn in a future version.

17405 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 17406 • The argument list is explicitly defined as **void**.
- 17407 • The DESCRIPTION is updated to state explicitly that the return value is a pointer to a string  
17408 giving the user name, rather than simply a pointer to the user name as stated in previous  
17409 issues.

17410 **Issue 5**

17411 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
17412 VALUE section.

17413 The *getlogin\_r()* function is included for alignment with the POSIX Threads Extension.

17414 A note indicating that the *getlogin()* function need not be reentrant is added to the  
17415 DESCRIPTION.

17416 **Issue 6**

17417 The *getlogin\_r()* function is marked as part of the Thread-Safe Functions option.

17418 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

17419 The following new requirements on POSIX implementations derive from alignment with the  
17420 Single UNIX Specification:

- 17421 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
- 17422 • The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

17423 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
17424 its avoidance of possibly using a static data area.

## 17425 NAME

17426 getmsg, getpmsg — receive next message from a STREAMS file (STREAMS)

## 17427 SYNOPSIS

17428 XSR #include <stropts.h>

```
17429 int getmsg(int fildev, struct strbuf *restrict ctlptr,
17430 struct strbuf *restrict dataptr, int *restrict flagsp);
17431 int getpmsg(int fildev, struct strbuf *restrict ctlptr,
17432 struct strbuf *restrict dataptr, int *restrict bandp,
17433 int *restrict flagsp);
17434
```

## 17435 DESCRIPTION

17436 The *getmsg()* function shall retrieve the contents of a message located at the head of the  
 17437 STREAM head read queue associated with a STREAMS file and place the contents into one or  
 17438 more buffers. The message contains either a data part, a control part, or both. The data and  
 17439 control parts of the message are placed into separate buffers, as described below. The semantics  
 17440 of each part are defined by the originator of the message.

17441 The *getpmsg()* function does the same thing as *getmsg()*, but provides finer control over the  
 17442 priority of the messages received. Except where noted, all requirements on *getmsg()* also pertain  
 17443 to *getpmsg()*.

17444 The *fildev* argument specifies a file descriptor referencing a STREAMS-based file.

17445 The *ctlptr* and *dataptr* arguments each point to a **strbuf** structure, in which the *buf* member points  
 17446 to a buffer in which the data or control information is to be placed, and the *maxlen* member  
 17447 indicates the maximum number of bytes this buffer can hold. On return, the *len* member  
 17448 contains the number of bytes of data or control information actually received. The *len* member is  
 17449 set to 0 if there is a zero-length control or data part and *len* is set to -1 if no data or control  
 17450 information is present in the message.

17451 When *getmsg()* is called, *flagsp* should point to an integer that indicates the type of message the  
 17452 process is able to receive. This is described further below.

17453 The *ctlptr* argument is used to hold the control part of the message, and *dataptr* is used to hold  
 17454 the data part of the message. If *ctlptr* (or *dataptr*) is a null pointer or the *maxlen* member is -1, the  
 17455 control (or data) part of the message is not processed and is left on the STREAM head read  
 17456 queue, and if the *ctlptr* (or *dataptr*) is not a null pointer, *len* is set to -1. If the *maxlen* member is  
 17457 set to 0 and there is a zero-length control (or data) part, that zero-length part is removed from  
 17458 the read queue and *len* is set to 0. If the *maxlen* member is set to 0 and there are more than 0 bytes  
 17459 of control (or data) information, that information is left on the read queue and *len* is set to 0. If  
 17460 the *maxlen* member in *ctlptr* (or *dataptr*) is less than the control (or data) part of the message,  
 17461 *maxlen* bytes are retrieved. In this case, the remainder of the message is left on the STREAM head  
 17462 read queue and a non-zero return value is provided.

17463 By default, *getmsg()* processes the first available message on the STREAM head read queue.  
 17464 However, a process may choose to retrieve only high-priority messages by setting the integer  
 17465 pointed to by *flagsp* to RS\_HIPRI. In this case, *getmsg()* shall only process the next message if it is  
 17466 a high-priority message. When the integer pointed to by *flagsp* is 0, any message shall be  
 17467 retrieved. In this case, on return, the integer pointed to by *flagsp* is set to RS\_HIPRI if a high-  
 17468 priority message was retrieved, or 0 otherwise.

17469 For *getpmsg()*, the flags are different. The *flagsp* argument points to a bitmask with the following  
 17470 mutually-exclusive flags defined: MSG\_HIPRI, MSG\_BAND, and MSG\_ANY. Like *getmsg()*,  
 17471 *getpmsg()* processes the first available message on the STREAM head read queue. A process may



17472 choose to retrieve only high-priority messages by setting the integer pointed to by *flagsp* to  
 17473 MSG\_HIPRI and the integer pointed to by *bandp* to 0. In this case, *getpmsg()* shall only process  
 17474 the next message if it is a high-priority message. In a similar manner, a process may choose to  
 17475 retrieve a message from a particular priority band by setting the integer pointed to by *flagsp* to  
 17476 MSG\_BAND and the integer pointed to by *bandp* to the priority band of interest. In this case,  
 17477 *getpmsg()* shall only process the next message if it is in a priority band equal to, or greater than,  
 17478 the integer pointed to by *bandp*, or if it is a high-priority message. If a process just wants to get  
 17479 the first message off the queue, the integer pointed to by *flagsp* should be set to MSG\_ANY and  
 17480 the integer pointed to by *bandp* should be set to 0. On return, if the message retrieved was a  
 17481 high-priority message, the integer pointed to by *flagsp* is set to MSG\_HIPRI and the integer  
 17482 pointed to by *bandp* shall be set to 0. Otherwise, the integer pointed to by *flagsp* shall be set to  
 17483 MSG\_BAND and the integer pointed to by *bandp* shall be set to the priority band of the message.

17484 If O\_NONBLOCK is not set, *getmsg()* and *getpmsg()* shall block until a message of the type  
 17485 specified by *flagsp* is available at the front of the STREAM head read queue. If O\_NONBLOCK is  
 17486 set and a message of the specified type is not present at the front of the read queue, *getmsg()* and  
 17487 *getpmsg()* shall fail and set *errno* to [EAGAIN].

17488 If a hangup occurs on the STREAM from which messages are retrieved, *getmsg()* and *getpmsg()*  
 17489 continue to operate normally, as described above, until the STREAM head read queue is empty.  
 17490 Thereafter, they return 0 in the *len* members of *ctlptr* and *dataptr*.

#### 17491 RETURN VALUE

17492 Upon successful completion, *getmsg()* and *getpmsg()* shall return a non-negative value. A value  
 17493 of 0 indicates that a full message was read successfully. A return value of MORECTL indicates  
 17494 that more control information is waiting for retrieval. A return value of MOREDATA indicates  
 17495 that more data is waiting for retrieval. A return value of the bitwise-logical OR of MORECTL  
 17496 and MOREDATA indicates that both types of information remain. Subsequent *getmsg()* and  
 17497 *getpmsg()* calls shall retrieve the remainder of the message. However, if a message of higher  
 17498 priority has come in on the STREAM head read queue, the next call to *getmsg()* or *getpmsg()*  
 17499 shall retrieve that higher-priority message before retrieving the remainder of the previous  
 17500 message.

17501 If the high priority control part of the message is consumed, the message shall be placed back on  
 17502 the queue as a normal message of band 0. Subsequent *getmsg()* and *getpmsg()* calls shall retrieve  
 17503 the remainder of the message. If, however, a priority message arrives or already exists on the  
 17504 STREAM head, the subsequent call to *getmsg()* or *getpmsg()* retrieves the higher-priority  
 17505 message before retrieving the remainder of the message that was put back.

17506 Upon failure, *getmsg()* and *getpmsg()* shall return -1 and set *errno* to indicate the error.

#### 17507 ERRORS

17508 The *getmsg()* and *getpmsg()* functions shall fail if:

|       |           |                                                                                                                                                                                                |  |
|-------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| 17509 | [EAGAIN]  | The O_NONBLOCK flag is set and no messages are available.                                                                                                                                      |  |
| 17510 | [EBADF]   | The <i>fildev</i> argument is not a valid file descriptor open for reading.                                                                                                                    |  |
| 17511 | [EBADMSG] | The queued message to be read is not valid for <i>getmsg()</i> or <i>getpmsg()</i> or a<br>17512 pending file descriptor is at the STREAM head.                                                |  |
| 17513 | [EINTR]   | A signal was caught during <i>getmsg()</i> or <i>getpmsg()</i> .                                                                                                                               |  |
| 17514 | [EINVAL]  | An illegal value was specified by <i>flagsp</i> , or the STREAM or multiplexer<br>17515 referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a<br>17516 multiplexer. |  |

17517 [ENOSTR] A STREAM is not associated with *fildev*.

17518 In addition, *getmsg()* and *getpmsg()* shall fail if the STREAM head had processed an  
17519 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of  
17520 *getmsg()* or *getpmsg()* but reflects the prior error.

## 17521 EXAMPLES

### 17522 Getting Any Message

17523 In the following example, the value of *fd* is assumed to refer to an open STREAMS file. The call  
17524 to *getmsg()* retrieves any available message on the associated STREAM-head read queue,  
17525 returning control and data information to the buffers pointed to by *ctrlbuf* and *databuf*,  
17526 respectively.

```
17527 #include <stropts.h>
17528 ...
17529 int fd;
17530 char ctrlbuf[128];
17531 char databuf[512];
17532 struct strbuf ctrl;
17533 struct strbuf data;
17534 int flags = 0;
17535 int ret;

17536 ctrl.buf = ctrlbuf;
17537 ctrl.maxlen = sizeof(ctrlbuf);

17538 data.buf = databuf;
17539 data.maxlen = sizeof(databuf);

17540 ret = getmsg (fd, &ctrl, &data, &flags);
```

### 17541 Getting the First Message off the Queue

17542 In the following example, the call to *getpmsg()* retrieves the first available message on the  
17543 associated STREAM-head read queue.

```
17544 #include <stropts.h>
17545 ...

17546 int fd;
17547 char ctrlbuf[128];
17548 char databuf[512];
17549 struct strbuf ctrl;
17550 struct strbuf data;
17551 int band = 0;
17552 int flags = MSG_ANY;
17553 int ret;

17554 ctrl.buf = ctrlbuf;
17555 ctrl.maxlen = sizeof(ctrlbuf);

17556 data.buf = databuf;
17557 data.maxlen = sizeof(databuf);

17558 ret = getpmsg (fd, &ctrl, &data, &band, &flags);
```

17559 **APPLICATION USAGE**

17560 None.

17561 **RATIONALE**

17562 None.

17563 **FUTURE DIRECTIONS**

17564 None.

17565 **SEE ALSO**

17566 *poll()*, *putmsg()*, *read()*, *write()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
17567 `<stropts.h>`, Section 2.6 (on page 539)

17568 **CHANGE HISTORY**

17569 First released in Issue 4, Version 2.

17570 **Issue 5**

17571 Moved from X/OPEN UNIX extension to BASE.

17572 A paragraph regarding “high-priority control parts of messages” is added to the RETURN  
17573 VALUE section.

17574 **Issue 6**

17575 This function is marked as part of the XSI STREAMS Option Group.

17576 The **restrict** keyword is added to the *getmsg()* and *getpmsg()* prototypes for alignment with the  
17577 ISO/IEC 9899:1999 standard.

17578 **NAME**

17579 getnameinfo — get name information

17580 **Notes to Reviewers**17581 *This section with side shading will not appear in the final copy. - Ed.*17582 The IPv6 developers believe that `getnameinfo()` should not truncate the result if the user buffer is  
17583 too small, but return an error status.17584 **SYNOPSIS**

17585 #include &lt;sys/socket.h&gt;

17586 #include &lt;netdb.h&gt;

```
17587 int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
17588 char *restrict node, socklen_t nodelen, char *restrict service,
17589 socklen_t servicelen, unsigned flags);
```

17590 **DESCRIPTION**17591 The `getnameinfo()` function translates a socket address to a node name and service location, all of  
17592 which are defined as in `getaddrinfo()`.17593 The `sa` argument points to a socket address structure to be translated.

17594 If the `node` argument is non-NULL and the `nodelen` argument is non-zero, then the `node` argument  
17595 points to a buffer able to contain up to `nodelen` characters that receives the node name as a null-  
17596 terminated string. If the `node` argument is NULL or the `nodelen` argument is zero, the node name  
17597 shall not be returned. If the node's name cannot be located, the numeric form of the node's  
17598 address is returned instead of its name.

17599 If the `service` argument is non-NULL and the `servicelen` argument is non-zero, then the `service`  
17600 argument points to a buffer able to contain up to `servicelen` characters that receives the service  
17601 name as a null-terminated string. If the `service` argument is NULL or the `servicelen` argument is  
17602 zero, the service name shall not be returned. If the service's name cannot be located, the numeric  
17603 form of the service address (for example, its port number) is returned instead of its name.

17604 The `node` and `service` arguments cannot both be NULL.17605 The `flags` argument is a flag that changes the default actions of the function. By default the fully-  
17606 qualified domain name (FQDN) for the host is returned, but:

- 17607 • If the flag bit `NI_NOFQDN` is set, only the node name portion of the FQDN shall be returned  
17608 for local hosts.
- 17609 • If the flag bit `NI_NUMERICHOST` is set, the numeric form of the host's address shall be  
17610 returned instead of its name, under all circumstances.
- 17611 • If the flag bit `NI_NAMEREQD` is set, an error shall be returned if the host's name cannot be  
17612 located.
- 17613 • If the flag bit `NI_NUMERICSERV` is set, the numeric form of the service address shall be  
17614 returned (for example, its port number) instead of its name, under all circumstances.
- 17615 • If the flag bit `NI_DGRAM` is set, this indicates that the service is a datagram service  
17616 (`SOCK_DGRAM`). The default behavior shall assume that the service is a stream service  
17617 (`SOCK_STREAM`).

17618 **Notes:**

- 17619 1. The two `NI_NUMERICxxx` flags are required to support the `-n` flag that many  
17620 commands provide.

17621                   2. The NI\_DGRAM flag is required for the few AF\_INET and AF\_INET6 port  
17622                   numbers (for example, 512-514) that represent different services for UDP and  
17623                   TCP.

17624                   The *getnameinfo()* function shall be thread-safe.

#### 17625 RETURN VALUE

17626                   A zero return value for *getnameinfo()* indicates successful completion; a non-zero return value  
17627                   indicates failure. The possible values for the failures are listed in the ERRORS section.

17628                   Upon successful completion, *getnameinfo()* shall return the *node* and *service* names, if requested,  
17629                   in the buffers provided. The returned names are always null-terminated strings, and may be  
17630                   truncated if the actual values are longer than can be stored in the buffers provided.

#### 17631 ERRORS

17632                   The *getnameinfo()* function shall fail and return the corresponding value if:

17633                   [EAI\_AGAIN]    The name could not be resolved at this time. Future attempts may succeed.

17634                   [EAI\_BADFLAGS]

17635                   The *flags* had an invalid value.

17636                   [EAI\_FAIL]     A non-recoverable error occurred.

17637                   [EAI\_FAMILY]   The address family was not recognized or the address length was invalid for  
17638                   the specified family.

17639                   [EAI\_MEMORY]   There was a memory allocation failure.

17640                   [EAI\_NONAME]   The name does not resolve for the supplied parameters.

17641                   NI\_NAMEREQD is set and the host's name cannot be located, or both  
17642                   *nodename* and *servname* were null.

17643                   [EAI\_SYSTEM]   A system error occurred. The error code can be found in *errno*.

#### 17644 EXAMPLES

17645                   None.

#### 17646 APPLICATION USAGE

17647                   If the returned values are to be used as part of any further name resolution (for example, passed  
17648                   to *getaddrinfo()*), applications shall either provide buffers large enough to store any result  
17649                   possible on the system or shall check for truncation and handle that case appropriately.

#### 17650 RATIONALE

17651                   None.

#### 17652 FUTURE DIRECTIONS

17653                   None.

#### 17654 SEE ALSO

17655                   *getaddrinfo()*, *getservbyname()*, *getservbyport()*, *inet\_ntop()*, *socket()*, the Base Definitions volume  
17656                   of IEEE Std. 1003.1-200x, <netdb.h>, <sys/socket.h>

#### 17657 CHANGE HISTORY

17658                   First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17659                   The **restrict** keyword is added to the *getnameinfo()* prototype for alignment with the  
17660                   ISO/IEC 9899:1999 standard.

17661 **NAME**

17662           getnetbyaddr — network database functions

17663 **SYNOPSIS**

17664           #include <netdb.h>

17665           struct netent \*getnetbyaddr(uint32\_t net, int type);

17666 **DESCRIPTION**

17667           Refer to *endnetent()*.

17668 **NAME**

17669       getnetbyname — network database functions

17670 **SYNOPSIS**

17671       #include <netdb.h>

17672       struct netent \*getnetbyname(const char \*name);

17673 **DESCRIPTION**

17674       Refer to *endnetent()*.

17675 **NAME**

17676       getnetent — network database functions

17677 **SYNOPSIS**

17678       #include <netdb.h>

17679       struct netent \*getnetent(void);

17680 **DESCRIPTION**

17681       Refer to *endnetent()*.



17682 **NAME**

17683            getopt, optarg, opterr, optind, optopt — command option parsing

17684 **SYNOPSIS**

17685            #include &lt;unistd.h&gt;

17686            int getopt(int argc, char \* const argv[], const char \*optstring);

17687            extern char \*optarg;

17688            extern int optind, opterr, optopt;

17689 **DESCRIPTION**

17690            The *getopt()* function is a command-line parser that can be used by applications that follow  
 17691            Utility Syntax Guidelines 3, 4, 5, 6, 7, 9, and 10 in the Base Definitions volume of  
 17692            IEEE Std. 1003.1-200x, Section 12.2, Utility Syntax Guidelines. The remaining guidelines are not  
 17693            addressed by *getopt()* and are the responsibility of the application.

17694            The parameters *argc* and *argv* are the argument count and argument array as passed to *main()*  
 17695            (see *exec*). The argument *optstring* is a string of recognized option characters; if a character is  
 17696            followed by a colon, the option takes an argument. All option characters allowed by Utility  
 17697            Syntax Guideline 3 are allowed in *optstring*. The implementation may accept other characters as  
 17698            an extension.

17699            The variable *optind* is the index of the next element of the *argv[]* vector to be processed. It is  
 17700            initialized to 1 by the system, and *getopt()* updates it when it finishes with each element of  
 17701            *argv[]*. When an element of *argv[]* contains multiple option characters, it is unspecified how  
 17702            *getopt()* determines which options have already been processed.

17703            The *getopt()* function shall return the next option character (if one is found) from *argv* that  
 17704            matches a character in *optstring*, if there is one that matches. If the option takes an argument,  
 17705            *getopt()* shall set the variable *optarg* to point to the option-argument as follows:

- 17706            1. If the option was the last character in the string pointed to by an element of *argv*, then  
 17707            *optarg* contains the next element of *argv*, and *optind* is incremented by 2. If the resulting  
 17708            value of *optind* is greater than *argc*, this indicates a missing option-argument, and *getopt()*  
 17709            shall return an error indication.
- 17710            2. Otherwise, *optarg* points to the string following the option character in that element of  
 17711            *argv*, and *optind* is incremented by 1.

17712            If, when *getopt()* is called:

17713            *argv[optind]*    is a null pointer  
 17714            \**argv[optind]*    is not the character -  
 17715            *argv[optind]*    points to the string "--"

17716            *getopt()* shall return -1 without changing *optind*. If:

17717            *argv[optind]*    points to the string "--"

17718            *getopt()* shall return -1 after incrementing *optind*.

17719            If *getopt()* encounters an option character that is not contained in *optstring*, it shall return the  
 17720            question-mark ('?') character. If it detects a missing option-argument, it shall return the colon  
 17721            character (':') if the first character of *optstring* was a colon, or a question-mark character ('?')  
 17722            otherwise. In either case, *getopt()* shall set the variable *optopt* to the option character that caused  
 17723            the error. If the application has not set the variable *opterr* to 0 and the first character of *optstring*  
 17724            is not a colon, *getopt()* shall also print a diagnostic message to *stderr* in the format specified for  
 17725            the *getopts* utility.

17726 The *getopt()* function need not be reentrant. A function that is not required to be reentrant is not  
17727 required to be thread-safe.

#### 17728 RETURN VALUE

17729 The *getopt()* function shall return the next option character specified on the command line.

17730 A colon (':') shall be returned if *getopt()* detects a missing argument and the first character of  
17731 *optstring* was a colon (':').

17732 A question mark ('?') shall be returned if *getopt()* encounters an option character not in  
17733 *optstring* or detects a missing argument and the first character of *optstring* was not a colon (':').

17734 Otherwise, *getopt()* shall return  $-1$  when all command line options are parsed.

#### 17735 ERRORS

17736 No errors are defined.

#### 17737 EXAMPLES

##### 17738 Parsing Command Line Options

17739 The following code fragment shows how you might process the arguments for a utility that can  
17740 take the mutually-exclusive options *a* and *b* and the options *f* and *o*, both of which require  
17741 arguments:

```
17742 #include <unistd.h>
17743 int
17744 main(int argc, char *argv[])
17745 {
17746 int c;
17747 int bflg, aflag, errflg;
17748 char *ifile;
17749 char *ofile;
17750 extern char *optarg;
17751 extern int optind, optopt;
17752 . . .
17753 while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
17754 switch(c) {
17755 case 'a':
17756 if (bflg)
17757 errflg++;
17758 else
17759 aflag++;
17760 break;
17761 case 'b':
17762 if (aflag)
17763 errflg++;
17764 else {
17765 bflg++;
17766 bproc();
17767 }
17768 break;
17769 case 'f':
17770 ifile = optarg;
17771 break;
17772 case 'o':
```

```

17773 ofile = optarg;
17774 break;
17775 case ':': /* -f or -o without operand */
17776 fprintf(stderr,
17777 "Option -%c requires an operand\n", optopt);
17778 errflg++;
17779 break;
17780 case '?':
17781 fprintf(stderr,
17782 "Unrecognized option: -%c\n", optopt);
17783 errflg++;
17784 }
17785 }
17786 if (errflg) {
17787 fprintf(stderr, "usage: . . . ");
17788 exit(2);
17789 }
17790 for (; optind < argc; optind++) {
17791 if (access(argv[optind], R_OK)) {
17792 . . .
17793 }

```

17794 This code accepts any of the following as equivalent:

```

17795 cmd -ao arg path path
17796 cmd -a -o arg path path
17797 cmd -o arg -a path path
17798 cmd -a -o arg — path path
17799 cmd -a -oarg path path
17800 cmd -aoarg path path

```

### 17801 **Checking Options and Arguments**

17802 The following example parses a set of command line options and prints messages to standard  
 17803 output for each option and argument that it encounters.

```

17804 #include <unistd.h>
17805 #include <stdio.h>
17806 ...
17807 int c;
17808 char *filename;
17809 extern char *optarg;
17810 extern int optind, optopt, opterr;
17811 ...
17812 while ((c = getopt(argc, argv, ":abf:")) != -1) {
17813 switch(c) {
17814 case 'a':
17815 printf("a is set\n");
17816 break;
17817 case 'b':
17818 printf("b is set\n");
17819 break;
17820 case 'f':
17821 filename = optarg;

```

```

17822 printf("filename is %s\n", filename);
17823 break;
17824 case ':':
17825 printf("--%c without filename\n", optopt);
17826 break;
17827 case '?':
17828 printf("unknown arg %c\n", optopt);
17829 break;
17830 }
17831 }

```

### 17832 **Selecting Options from the Command Line**

17833 The following example selects the type of database routines the user wants to use based on the  
 17834 *Options* argument.

```

17835 #include <unistd.h>
17836 #include <string.h>
17837 ...
17838 char *Options = "hdbtl";
17839 ...
17840 int dbtype, i;
17841 char c;
17842 char *st;
17843 ...
17844 dbtype = 0;
17845 while ((c = getopt(argc, argv, Options)) != -1) {
17846 if ((st = strchr(Options, c)) != NULL) {
17847 dbtype = st - Options;
17848 break;
17849 }
17850 }

```

### 17851 **APPLICATION USAGE**

17852 The *getopt()* function is only required to support option characters included in Utility Syntax  
 17853 Guideline 3. Many historical implementations of *getopt()* support other characters as options.  
 17854 This is an allowed extension, but applications that use extensions are not maximally portable.  
 17855 Note that support for multi-byte option characters is only possible when such characters can be  
 17856 represented as type **int**.

### 17857 **RATIONALE**

17858 The *optopt* variable represents historical practice and allows the application to obtain the identity  
 17859 of the invalid option.

17860 The description has been written to make it clear that *getopt()*, like the *getopts* utility, deals with  
 17861 option-arguments whether separated from the option by <blank> characters or not. Note that  
 17862 the requirements on *getopt()* and *getopts* are more stringent than the Utility Syntax Guidelines.

17863 The *getopt()* function shall return  $-1$ , rather than EOF, so that <**stdio.h**> is not required.

17864 The special significance of a colon as the first character of *optstring* makes *getopt()* consistent  
 17865 with the *getopts* utility. It allows an application to make a distinction between a missing  
 17866 argument and an incorrect option letter without having to examine the option letter. It is true  
 17867 that a missing argument can only be detected in one case, but that is a case that has to be  
 17868 considered.

17869 **FUTURE DIRECTIONS**

17870 None.

17871 **SEE ALSO**

17872 *exec*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>, the Shell and Utilities  
17873 volume of IEEE Std. 1003.1-200x

17874 **CHANGE HISTORY**

17875 First released in Issue 1. Derived from Issue 1 of the SVID.

17876 **Issue 4**

17877 The following changes are incorporated for alignment with the ISO POSIX-2 standard:

- 17878 • The <**unistd.h**> header is added to the SYNOPSIS section and <**stdio.h**> is deleted.
- 17879 • The type of argument *argv* is changed from **char\*\*** to **char\*const[ ]**.
- 17880 • The integer *optopt* is added to the list of external data items.
- 17881 • The DESCRIPTION is largely rewritten, without functional change, for alignment with the  
17882 ISO POSIX-2 standard, although the following differences should be noted:
  - 17883 — If the function detects a missing option-argument, it returns a colon (':') and sets *optopt*  
17884 to the option character.
  - 17885 — The termination conditions under which *getopt()* returns -1 are extended. Also note that  
17886 the termination condition is explicitly -1, rather than the value of EOF.
- 17887 • The EXAMPLES section is changed to illustrate the new functionality.

17888 **Issue 5**17889 A note indicating that the *getopt()* function need not be reentrant is added to the DESCRIPTION.17890 **Issue 6**

17891 IEEE PASC Interpretation 1003.2 #150 is applied.

17892 **NAME**

17893           getpeername — get the name of the peer socket

17894 **SYNOPSIS**

17895           #include <sys/socket.h>

17896           int getpeername(int *socket*, struct sockaddr \*restrict *address*,  
17897                           socklen\_t \**address\_len*);

17898 **DESCRIPTION**

17899           The *getpeername()* function shall retrieve the peer address of the specified socket, store this  
17900           address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this  
17901           address in the object pointed to by the *address\_len* argument.

17902           If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
17903           the stored address shall be truncated.

17904           If the protocol permits connections by unbound clients, and the peer is not bound, then the value  
17905           stored in the object pointed to by *address* is unspecified.

17906 **RETURN VALUE**

17907           Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
17908           indicate the error.

17909 **ERRORS**

17910           The *getpeername()* function shall fail if:

17911           [EBADF]           The *socket* argument is not a valid file descriptor.

17912           [EINVAL]          The socket has been shut down.

17913           [ENOTCONN]        The socket is not connected or otherwise has not had the peer prespecified.

17914           [ENOTSOCK]        The *socket* argument does not refer to a socket.

17915           [EOPNOTSUPP]      The operation is not supported for the socket protocol.

17916           The *getpeername()* function may fail if:

17917           [ENOBUFS]         Insufficient resources were available in the system to complete the call.

17918 **EXAMPLES**

17919           None.

17920 **APPLICATION USAGE**

17921           None.

17922 **RATIONALE**

17923           None.

17924 **FUTURE DIRECTIONS**

17925           None.

17926 **SEE ALSO**

17927           *accept()*, *bind()*, *getsockname()*, *socket()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
17928           <sys/socket.h>

17929 **CHANGE HISTORY**

17930           First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17931           The **restrict** keyword is added to the *getpeername()* prototype for alignment with the  
17932           ISO/IEC 9899:1999 standard.

17933 **NAME**

17934 getpgid — get the process group ID for a process

17935 **SYNOPSIS**

17936 XSI #include &lt;unistd.h&gt;

17937 pid\_t getpgid(pid\_t pid);

17938

17939 **DESCRIPTION**17940 The *getpgid()* function shall return the process group ID of the process whose process ID is equal  
17941 to *pid*. If *pid* is equal to 0, *getpgid()* shall return the process group ID of the calling process.17942 **RETURN VALUE**17943 Upon successful completion, *getpgid()* shall return a process group ID. Otherwise, it shall return  
17944 (**pid\_t**)-1 and set *errno* to indicate the error.17945 **ERRORS**17946 The *getpgid()* function shall fail if:17947 [EPERM] The process whose process ID is equal to *pid* is not in the same session as the  
17948 calling process, and the implementation does not allow access to the process  
17949 group ID of that process from the calling process.17950 [ESRCH] There is no process with a process ID equal to *pid*.17951 The *getpgid()* function may fail if:17952 [EINVAL] The value of the *pid* argument is invalid.17953 **EXAMPLES**

17954 None.

17955 **APPLICATION USAGE**

17956 None.

17957 **RATIONALE**

17958 None.

17959 **FUTURE DIRECTIONS**

17960 None.

17961 **SEE ALSO**17962 *exec*, *fork()*, *getpgrp()*, *getpid()*, *getsid()*, *setpgid()*, *setsid()*, the Base Definitions volume of  
17963 IEEE Std. 1003.1-200x, <unistd.h>17964 **CHANGE HISTORY**

17965 First released in Issue 4, Version 2.

17966 **Issue 5**

17967 Moved from X/OPEN UNIX extension to BASE.

17968 **NAME**

17969           getpgrp — get the process group ID of the calling process

17970 **SYNOPSIS**

17971           #include <unistd.h>  
17972           pid\_t getpgrp(void);

17973 **DESCRIPTION**

17974           The *getpgrp()* function shall return the process group ID of the calling process.

17975 **RETURN VALUE**

17976           The *getpgrp()* function is always successful and no return value is reserved to indicate an error.

17977 **ERRORS**

17978           No errors are defined.

17979 **EXAMPLES**

17980           None.

17981 **APPLICATION USAGE**

17982           None.

17983 **RATIONALE**

17984           4.3 BSD provides a *getpgrp()* function that returns the process group ID for a specified process.  
17985           Although this function is used to support job control, all known job control shells always specify  
17986           the calling process with this function. Thus, the simpler System V *getpgrp()* suffices, and the  
17987           added complexity of the 4.3 BSD *getpgrp()* has been omitted from this volume of  
17988           IEEE Std. 1003.1-200x.

17989 **FUTURE DIRECTIONS**

17990           None.

17991 **SEE ALSO**

17992           *exec*, *fork()*, *getpgid()*, *getpid()*, *getppid()*, *kill()*, *setpgid()*, *setsid()*, the Base Definitions volume of  
17993           IEEE Std. 1003.1-200x, <sys/types.h>, <unistd.h>

17994 **CHANGE HISTORY**

17995           First released in Issue 1. Derived from Issue 1 of the SVID.

17996 **Issue 4**

17997           The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
17998           XSI-conformant systems.

17999           The <unistd.h> header is added to the SYNOPSIS section.

18000           The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 18001           • The argument list is explicitly defined as **void**.

18002 **Issue 6**

18003           In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

18004           The following new requirements on POSIX implementations derive from alignment with the  
18005           Single UNIX Specification:

- 18006           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
18007           required for conforming implementations of previous POSIX specifications, it was not  
18008           required for UNIX applications.



18009 **NAME**

18010            *getpid* — get the process ID

18011 **SYNOPSIS**

18012            #include <unistd.h>

18013            pid\_t *getpid*(void);

18014 **DESCRIPTION**

18015            The *getpid*() function shall return the process ID of the calling process.

18016 **RETURN VALUE**

18017            The *getpid*() function is always successful and no return value is reserved to indicate an error.

18018 **ERRORS**

18019            No errors are defined.

18020 **EXAMPLES**

18021            None.

18022 **APPLICATION USAGE**

18023            None.

18024 **RATIONALE**

18025            None.

18026 **FUTURE DIRECTIONS**

18027            None.

18028 **SEE ALSO**

18029            *exec*, *fork*(), *getpgrp*(), *getppid*(), *kill*(), *setpgid*(), *setsid*(), the Base Definitions volume of  
18030            IEEE Std. 1003.1-200x, <sys/types.h>, <unistd.h>

18031 **CHANGE HISTORY**

18032            First released in Issue 1. Derived from Issue 1 of the SVID.

18033 **Issue 4**

18034            The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
18035            XSI-conformant systems.

18036            The <unistd.h> header is added to the SYNOPSIS section.

18037            The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 18038            • The argument list is explicitly defined as **void**.

18039 **Issue 6**

18040            In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

18041            The following new requirements on POSIX implementations derive from alignment with the  
18042            Single UNIX Specification:

- 18043            • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
18044            required for conforming implementations of previous POSIX specifications, it was not  
18045            required for UNIX applications.

18046 **NAME**

18047       getpmsg — receive next message from a STREAMS file

18048 **SYNOPSIS**

18049 XSI     #include <stropts.h>

```
18050 int getpmsg(int fildev, struct strbuf *restrict ctlptr,
18051 struct strbuf *restrict dataptr, int *restrict bandp,
18052 int *restrict flagsp);
```

18053

18054 **DESCRIPTION**

18055       Refer to *getmsg()*.

18056 **NAME**

18057           getppid — get the parent process ID

18058 **SYNOPSIS**

18059           #include <unistd.h>

18060           pid\_t getppid(void);

18061 **DESCRIPTION**

18062           The *getppid()* function shall return the parent process ID of the calling process.

18063 **RETURN VALUE**

18064           The *getppid()* function is always successful and no return value is reserved to indicate an error.

18065 **ERRORS**

18066           No errors are defined.

18067 **EXAMPLES**

18068           None.

18069 **APPLICATION USAGE**

18070           None.

18071 **RATIONALE**

18072           None.

18073 **FUTURE DIRECTIONS**

18074           None.

18075 **SEE ALSO**

18076           *exec*, *fork()*, *getpgid()*, *getpgrp()*, *getpid()*, *kill()*, *setpgid()*, *setsid()*, the Base Definitions volume of  
18077 IEEE Std. 1003.1-200x, <sys/types.h>, <unistd.h>

18078 **CHANGE HISTORY**

18079           First released in Issue 1. Derived from Issue 1 of the SVID.

18080 **Issue 4**

18081           The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
18082 XSI-conformant systems.

18083           The <unistd.h> header is added to the SYNOPSIS section.

18084           The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 18085           • The argument list is explicitly defined as **void**.

18086 **Issue 6**

18087           In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

18088           The following new requirements on POSIX implementations derive from alignment with the  
18089 Single UNIX Specification:

- 18090           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
18091 required for conforming implementations of previous POSIX specifications, it was not  
18092 required for UNIX applications.

## 18093 NAME

18094 getpriority, setpriority — get or set the nice value

## 18095 SYNOPSIS

```
18096 xsi #include <sys/resource.h>
18097
18097 int getpriority(int which, id_t who);
18098 int setpriority(int which, id_t who, int value);
18099
```

## 18100 DESCRIPTION

18101 The *getpriority()* function obtains the nice value of a process, process group, or user. The  
 18102 *setpriority()* function sets the nice value of a process, process group, or user to *value*+ {NZERO}.

18103 Target processes are specified by the values of the *which* and *who* arguments. The *which*  
 18104 argument may be one of the following values: PRIO\_PROCESS, PRIO\_PGRP, or PRIO\_USER,  
 18105 indicating that the *who* argument is to be interpreted as a process ID, a process group ID, or an  
 18106 effective user ID, respectively. A 0 value for the *who* argument specifies the current process,  
 18107 process group, or user.

18108 The nice value set with *setpriority()* shall be applied to the process. If the process is multi-  
 18109 threaded, the nice value shall affect all system scope threads in the process.

18110 If more than one process is specified, *getpriority()* shall return value {NZERO} less than the  
 18111 lowest nice value pertaining to any of the specified processes, and *setpriority()* sets the nice  
 18112 values of all of the specified processes to *value*+ {NZERO}.

18113 The default nice value is {NZERO}; lower nice values cause more favorable scheduling. While  
 18114 the range of valid nice values is [0,{NZERO}\*2 -1], implementations may enforce more  
 18115 restrictive limits. If *value*+ {NZERO} is less than the system's lowest supported nice value,  
 18116 *setpriority()* sets the nice value to the lowest supported value; if *value*+ {NZERO} is greater than  
 18117 the system's highest supported nice value, *setpriority()* sets the nice value to the highest  
 18118 supported value.

18119 Only a process with appropriate privileges can lower its nice value.

18120 PS|TPS Any processes or threads using SCHED\_FIFO or SCHED\_RR are unaffected by a call to  
 18121 *setpriority()*. This is not considered an error.

18122 The effect of changing the nice value may vary depending on the process-scheduling algorithm  
 18123 in effect.

18124 Because *getpriority()* can return the value -1 on successful completion, it is necessary to set *errno*  
 18125 to 0 prior to a call to *getpriority()*. If *getpriority()* returns the value -1, then *errno* can be checked  
 18126 to see if an error occurred or if the value is a legitimate nice value.

## 18127 RETURN VALUE

18128 Upon successful completion, *getpriority()* shall return an integer in the range from -{NZERO} to  
 18129 {NZERO}-1. Otherwise, -1 shall be returned and *errno* set to indicate the error.

18130 Upon successful completion, *setpriority()* shall return 0; otherwise, -1 shall be returned and *errno*  
 18131 set to indicate the error.

## 18132 ERRORS

18133 The *getpriority()* and *setpriority()* functions shall fail if:

|       |         |                                                                                   |
|-------|---------|-----------------------------------------------------------------------------------|
| 18134 | [ESRCH] | No process could be located using the <i>which</i> and <i>who</i> argument values |
| 18135 |         | specified.                                                                        |

18136 [EINVAL] The value of the *which* argument was not recognized, or the value of the *who* |  
 18137 argument is not a valid process ID, process group ID, or user ID.

18138 In addition, *setpriority()* may fail if:

18139 [EPERM] A process was located, but neither the real nor effective user ID of the |  
 18140 executing process match the effective user ID of the process whose nice value |  
 18141 is being changed.

18142 [EACCES] A request was made to change the nice value to a lower numeric value and |  
 18143 the current process does not have appropriate privileges.

#### 18144 EXAMPLES

##### 18145 Using *getpriority()*

18146 The following example returns the current scheduling priority for the process ID returned by the |  
 18147 call to *getpid()*.

```
18148 #include <sys/resource.h>
18149 ...
18150 int which = PRIO_PROCESS;
18151 id_t pid;
18152 int ret;

18153 pid = getpid();
18154 ret = getpriority(which, pid);
```

##### 18155 Using *setpriority()*

18156 The following example sets the priority for the current process ID to  $-20$ .

```
18157 #include <sys/resource.h>
18158 ...
18159 int which = PRIO_PROCESS;
18160 id_t pid;
18161 int priority = -20;
18162 int ret;

18163 pid = getpid();
18164 ret = setpriority(which, pid, priority);
```

#### 18165 APPLICATION USAGE

18166 The *getpriority()* and *setpriority()* functions work with an offset nice value (nice value |  
 18167  $-\{\text{NZERO}\}$ ). The nice value is in the range  $[0, 2 * \{\text{NZERO}\} - 1]$ , while the return value for |  
 18168 *getpriority()* and the third parameter for *setpriority()* are in the range  $[-\{\text{NZERO}\}, \{\text{NZERO}\} - 1]$ .

#### 18169 RATIONALE

18170 None.

#### 18171 FUTURE DIRECTIONS

18172 None.

#### 18173 SEE ALSO

18174 *nice()*, *sched\_get\_priority\_max()*, *sched\_setscheduler()*, the Base Definitions volume of |  
 18175 IEEE Std. 1003.1-200x, *<sys/resource.h>*

18176 **CHANGE HISTORY**

18177 First released in Issue 4, Version 2.

18178 **Issue 5**

18179 Moved from X/OPEN UNIX extension to BASE.

18180 The DESCRIPTION is reworded in terms of the nice value rather than *priority* to avoid confusion  
18181 with functionality in the POSIX Realtime Extension.

18182 **NAME**

18183       getprotobyname — network protocol database functions

18184 **SYNOPSIS**

18185       #include <netdb.h>

18186       struct protoent \*getprotobyname(const char \*name);

18187 **DESCRIPTION**

18188       Refer to *endprotoent()*.

18189 **NAME**

18190           getprotobynumber — network protocol database functions

18191 **SYNOPSIS**

18192           #include <netdb.h>

18193           struct protoent \*getprotobynumber(int *proto*);

18194 **DESCRIPTION**

18195           Refer to *endprotoent()*.



18196 **NAME**

18197       getprotoent — network protocol database functions

18198 **SYNOPSIS**

18199       #include <netdb.h>

18200       struct protoent \*getprotoent(void);

18201 **DESCRIPTION**

18202       Refer to *endprotoent()*.

18203 **NAME**

18204           getpwent — get user database entry

18205 **SYNOPSIS**

18206 xSI       #include <pwd.h>

18207           struct passwd \*getpwent(void);

18208

18209 **DESCRIPTION**

18210           Refer to *endpwent()*.

## 18211 NAME

18212 getpwnam, getpwnam\_r — search user database for a name

## 18213 SYNOPSIS

18214 #include <pwd.h>

18215 struct passwd \*getpwnam(const char \*name);

18216 TSF int getpwnam\_r(const char \*name, struct passwd \*pwd, char \*buffer,

18217 size\_t bufsize, struct passwd \*\*result);

18218

## 18219 DESCRIPTION

18220 The *getpwnam()* function shall search the user database for an entry with a matching *name*.

18221 The *getpwnam()* function need not be reentrant. A function that is not required to be reentrant is  
18222 not required to be thread-safe.

18223 Applications wishing to check for error situations should set *errno* to 0 before calling  
18224 *getpwnam()*. If *getpwnam()* returns a null pointer and *errno* is non-zero, an error occurred.

18225 TSF The *getpwnam\_r()* function updates the **passwd** structure pointed to by *pwd* and stores a pointer  
18226 to that structure at the location pointed to by *result*. The structure shall contain an entry from  
18227 the user database with a matching *name*. Storage referenced by the structure is allocated from  
18228 the memory provided with the *buffer* parameter, which is *bufsize* characters in size.

## 18229 Notes to Reviewers

18230 *This section with side shading will not appear in the final copy. - Ed.*

18231 D3, XSH, ERN 301 says that the size above is in bytes, not characters, and proposes changing  
18232 "characters" to "bytes".

18233 The maximum size needed for this buffer can be determined with the  
18234 `{_SC_GETPW_R_SIZE_MAX} sysconf()` parameter. A NULL pointer shall be returned at the  
18235 location pointed to by *result* on error or if the requested entry is not found.

## 18236 RETURN VALUE

18237 The *getpwnam()* function shall return a pointer to a **struct passwd** with the structure as defined  
18238 in <pwd.h> with a matching entry if found. A null pointer shall be returned if the requested  
18239 entry is not found, or an error occurs. On error, *errno* shall be set to indicate the error.

18240 The return value may point to a static area which is overwritten by a subsequent call to  
18241 *getpwent()*, *getpwnam()*, or *getpwuid()*.

18242 TSF If successful, the *getpwnam\_r()* function shall return zero; otherwise, an error number shall be  
18243 returned to indicate the error.

## 18244 ERRORS

18245 The *getpwnam()* and *getpwnam\_r()* functions may fail if:

18246 [EIO] An I/O error has occurred.

18247 [EINTR] A signal was caught during *getpwnam()*.

18248 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

18249 [ENFILE] The maximum allowable number of files is currently open in the system.

18250 TSF The *getpwnam\_r()* function may fail if:

18251 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to  
18252 be referenced by the resulting **passwd** structure.

18253 **EXAMPLES**18254 **Getting an Entry for the Login Name**

18255 The following example uses the *getlogin()* function to return the name of the user who logged in;  
18256 this information is passed to the *getpwnam()* function to get the user database entry for that user.

```
18257 #include <sys/types.h>
18258 #include <pwd.h>
18259 #include <unistd.h>
18260 #include <stdio.h>
18261 #include <stdlib.h>
18262 ...
18263 char *lgn;
18264 struct passwd *pw;
18265 ...
18266 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
18267 fprintf(stderr, "Get of user information failed.\n"); exit(1);
18268 }
18269 ...
```

18270 **APPLICATION USAGE**

18271 Three names associated with the current process can be determined: *getpwuid(geteuid())* returns  
18272 the name associated with the effective user ID of the process; *getlogin()* returns the name  
18273 associated with the current login activity; and *getpwuid(getuid())* returns the name associated  
18274 with the real user ID of the process.

18275 The *getpwnam\_r()* function is thread-safe and shall return values in a user-supplied buffer  
18276 instead of possibly using a static data area that may be overwritten by each call.

18277 **RATIONALE**

18278 None.

18279 **FUTURE DIRECTIONS**

18280 None.

18281 **SEE ALSO**

18282 *getpwuid()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<limits.h>**, **<pwd.h>**,  
18283 **<sys/types.h>**

18284 **CHANGE HISTORY**

18285 First released in Issue 1. Derived from System V Release 2.0.

18286 **Issue 4**

18287 The DESCRIPTION is clarified.

18288 The **<sys/types.h>** header is now marked as optional (OH); this header need not be included on  
18289 XSI-conformant systems.

18290 The last sentence in the RETURN VALUE section, indicating that *errno* is set on error, is marked  
18291 as an extension.

18292 The errors [EIO], [EINTR], [EMFILE], and [ENFILE] are marked as extensions.

18293 The APPLICATION USAGE section is expanded to warn about possible reuses of the area used  
18294 to pass the return value, and to indicate how applications should check for errors.

18295 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 18296
- The type of argument *name* is changed from **char\*** to **const char\***.
- 18297 **Issue 5**
- 18298 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
18299 VALUE section.
- 18300 The *getpwnam\_r()* function is included for alignment with the POSIX Threads Extension.
- 18301 A note indicating that the *getpwnam()* function need not be reentrant is added to the  
18302 DESCRIPTION.
- 18303 **Issue 6**
- 18304 The *getpwnam\_r()* function is marked as part of the Thread-Safe Functions option.
- 18305 The Open Group corrigenda item U028/3 has been applied correcting text in the DESCRIPTION  
18306 describing matching the *name*.
- 18307 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.
- 18308 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.
- 18309 The following new requirements on POSIX implementations derive from alignment with the  
18310 Single UNIX Specification:
- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
  - In the RETURN VALUE section, the requirement to set *errno* on error is added.
  - The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.
- 18316 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
18317 its avoidance of possibly using a static data area.

## 18318 NAME

18319 getpwuid, getpwuid\_r — search user database for a user ID

## 18320 SYNOPSIS

18321 #include <pwd.h>

18322 struct passwd \*getpwuid(uid\_t uid);

18323 TSF int getpwuid\_r(uid\_t uid, struct passwd \*pwd, char \*buffer,

18324 size\_t bufsize, struct passwd \*\*result);

18325

## 18326 DESCRIPTION

18327 The *getpwuid()* function shall search the user database for an entry with a matching *uid*.

18328 The *getpwuid()* function need not be reentrant. A function that is not required to be reentrant is  
18329 not required to be thread-safe.

18330 Applications wishing to check for error situations should set *errno* to 0 before calling *getpwuid()*.  
18331 If *getpwuid()* returns a null pointer and *errno* is set to non-zero, an error occurred.

18332 TSF The *getpwuid\_r()* function updates the **passwd** structure pointed to by *pwd* and stores a pointer  
18333 to that structure at the location pointed to by *result*. The structure shall contain an entry from  
18334 the user database with a matching *uid*. Storage referenced by the structure is allocated from the  
18335 memory provided with the *buffer* parameter, which is *bufsize* characters in size. The maximum  
18336 size needed for this buffer can be determined with the `{_SC_GETPW_R_SIZE_MAX}` *sysconf()*  
18337 parameter. A NULL pointer shall be returned at the location pointed to by *result* on error or if the  
18338 requested entry is not found.

## 18339 RETURN VALUE

18340 The *getpwuid()* function shall return a pointer to a **struct passwd** with the structure as defined in  
18341 <**pwd.h**> with a matching entry if found. A null pointer shall be returned if the requested entry  
18342 is not found, or an error occurs. On error, *errno* shall be set to indicate the error.

18343 The return value may point to a static area which is overwritten by a subsequent call to  
18344 *getpwent()*, *getpwnam()*, or *getpwuid()*.

18345 TSF If successful, the *getpwuid\_r()* function shall return zero; otherwise, an error number shall be  
18346 returned to indicate the error.

## 18347 ERRORS

18348 The *getpwuid()* and *getpwuid\_r()* functions may fail if:

18349 [EIO] An I/O error has occurred.

18350 [EINTR] A signal was caught during *getpwuid()*.

18351 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

18352 [ENFILE] The maximum allowable number of files is currently open in the system.

18353 TSF The *getpwuid\_r()* function may fail if:

18354 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to  
18355 be referenced by the resulting **passwd** structure.

18356 **EXAMPLES**18357 **Getting an Entry for the Root User**

18358 The following example gets the user database entry for the user with user ID 0 (root).

```
18359 #include <sys/types.h>
18360 #include <pwd.h>
18361 ...
18362 uid_t id = 0;
18363 struct passwd *pwd;
18364 pwd = getpwuid(id);
```

18365 **Finding the Name for the Effective User ID**

18366 The following example defines *pws* as a pointer to a structure of type **passwd**, which is used to  
 18367 store the structure pointer returned by the call to the *getpwuid()* function. The *geteuid()* function  
 18368 shall return the effective user ID of the calling process; this is used as the search criteria for the  
 18369 *getpwuid()* function. The call to *getpwuid()* shall return a pointer to the structure containing that  
 18370 user ID value.

```
18371 #include <unistd.h>
18372 #include <sys/types.h>
18373 #include <pwd.h>
18374 ...
18375 struct passwd *pws;
18376 pws = getpwuid(geteuid());
```

18377 **Finding an Entry in the User Database**

18378 The following example uses *getpwuid()* to search the user database for a user ID that was  
 18379 previously stored in a **stat** structure, then prints out the user name if it is found. If the user is not  
 18380 found, the program prints the numeric value of the user ID for the entry.

```
18381 #include <sys/types.h>
18382 #include <pwd.h>
18383 #include <stdio.h>
18384 ...
18385 struct stat statbuf;
18386 struct passwd *pwd;
18387 ...
18388 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
18389 printf(" %-8.8s", pwd->pw_name);
18390 else
18391 printf(" %-8d", statbuf.st_uid);
```

18392 **APPLICATION USAGE**

18393 Three names associated with the current process can be determined: *getpwuid(geteuid())* returns  
 18394 the name associated with the effective user ID of the process; *getlogin()* returns the name  
 18395 associated with the current login activity; and *getpwuid(getuid())* returns the name associated  
 18396 with the real user ID of the process.

18397 The *getpwuid\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
 18398 of possibly using a static data area that may be overwritten by each call.

18399 **RATIONALE**

18400 None.

18401 **FUTURE DIRECTIONS**

18402 None.

18403 **SEE ALSO**

18404 *getpwnam()*, *geteuid()*, *getuid()*, *getlogin()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
 18405 `<limits.h>`, `<pwd.h>`, `<sys/types.h>`

18406 **CHANGE HISTORY**

18407 First released in Issue 1. Derived from System V Release 2.0.

18408 **Issue 4**

18409 The DESCRIPTION is clarified.

18410 The `<sys/types.h>` header is now marked as optional (OH); this header need not be included on  
 18411 XSI-conformant systems.

18412 The last sentence in the RETURN VALUE section, indicating that *errno* is set on error, is marked  
 18413 as an extension.

18414 The errors [EIO], [EINTR], [EMFILE], and [ENFILE] are marked as extensions.

18415 A note is added to the APPLICATION USAGE section indicating how an application should  
 18416 check for errors.

18417 **Issue 5**

18418 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 18419 VALUE section.

18420 The *getpwuid\_r()* function is included for alignment with the POSIX Threads Extension.

18421 A note indicating that the *getpwuid()* function need not be reentrant is added to the  
 18422 DESCRIPTION.

18423 **Issue 6**18424 The *getpwuid\_r()* function is marked as part of the Thread-Safe Functions option.

18425 The Open Group corrigenda item U028/3 has been applied correcting text in the DESCRIPTION  
 18426 describing matching the *uid*.

18427 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

18428 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

18429 The following new requirements on POSIX implementations derive from alignment with the  
 18430 Single UNIX Specification:

18431 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 18432 required for conforming implementations of previous POSIX specifications, it was not  
 18433 required for UNIX applications.

18434 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

18435 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

18436 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
 18437 its avoidance of possibly using a static data area.



18438 **NAME**

18439 getrlimit, setrlimit — control maximum resource consumption

18440 **SYNOPSIS**18441 xSI `#include <sys/resource.h>`18442 `int getrlimit(int resource, struct rlimit *rlp);`18443 `int setrlimit(int resource, const struct rlimit *rlp);`

18444

18445 **DESCRIPTION**18446 Limits on the consumption of a variety of resources by the calling process may be obtained with  
18447 *getrlimit()* and set with *setrlimit()*.18448 Each call to either *getrlimit()* or *setrlimit()* identifies a specific resource to be operated upon as  
18449 well as a resource limit. A resource limit is represented by an **rlimit** structure. The *rlim\_cur*  
18450 member specifies the current or soft limit and the *rlim\_max* member specifies the maximum or  
18451 hard limit. Soft limits may be changed by a process to any value that is less than or equal to the  
18452 hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or  
18453 equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both  
18454 hard and soft limits can be changed in a single call to *setrlimit()* subject to the constraints  
18455 described above.18456 The value RLIM\_INFINITY, defined in `<sys/resource.h>`, shall be considered to be larger than  
18457 any other limit value. If a call to *getrlimit()* returns RLIM\_INFINITY for a resource, it means the  
18458 implementation shall not enforce limits on that resource. Specifying RLIM\_INFINITY as any  
18459 resource limit value on a successful call to *setrlimit()* inhibits enforcement of that resource limit.

18460 The following resources are defined:

18461 **RLIMIT\_CORE** This is the maximum size of a core file in bytes that may be created by a  
18462 process. A limit of 0 shall prevent the creation of a core file. If this limit is  
18463 exceeded, the writing of a core file shall terminate at this size.18464 **RLIMIT\_CPU** This is the maximum amount of CPU time in seconds used by a process.  
18465 If this limit is exceeded, SIGXCPU shall be generated for the process. If  
18466 the process is catching or ignoring SIGXCPU, or all threads belonging to  
18467 that process are blocking SIGXCPU, the behavior is unspecified.18468 **RLIMIT\_DATA** This is the maximum size of a process' data segment in bytes. If this limit  
18469 is exceeded, the *malloc()* function shall fail with *errno* set to [ENOMEM].18470 **RLIMIT\_FSIZE** This is the maximum size of a file in bytes that may be created by a  
18471 process. If a write or truncate operation would cause this limit to be  
18472 exceeded, SIGXFSZ shall be generated for the thread. If the thread is  
18473 blocking, or the process is catching or ignoring SIGXFSZ, continued  
18474 attempts to increase the size of a file from end-of-file to beyond the limit  
18475 shall fail with *errno* set to [EFBIG].18476 **RLIMIT\_NOFILE** This is a number one greater than the maximum value that the system  
18477 may assign to a newly-created descriptor. If this limit is exceeded,  
18478 functions that allocate new file descriptors may fail with *errno* set to  
18479 [EMFILE]. This limit constrains the number of file descriptors that a  
18480 process may allocate.18481 **RLIMIT\_STACK** This is the maximum size of a process' stack in bytes. The  
18482 implementation does not automatically grow the stack beyond this limit.  
18483 If this limit is exceeded, SIGSEGV shall be generated for the thread. If the

18484 thread is blocking SIGSEGV, or the process is ignoring or catching  
 18485 SIGSEGV and has not made arrangements to use an alternate stack, the  
 18486 disposition of SIGSEGV shall be set to SIG\_DFL before it is generated.

18487 RLIMIT\_AS This is the maximum size of a process' total available memory, in bytes. If  
 18488 this limit is exceeded, the *malloc()* and *mmap()* functions shall fail with  
 18489 *errno* set to [ENOMEM]. In addition, the automatic stack growth fails  
 18490 with the effects outlined above.

18491 When using the *getrlimit()* function, if a resource limit can be represented correctly in an object  
 18492 of type **rlim\_t**, then its representation is returned; otherwise, if the value of the resource limit is  
 18493 equal to that of the corresponding saved hard limit, the value returned shall be  
 18494 RLIM\_SAVED\_MAX; otherwise, the value returned shall be RLIM\_SAVED\_CUR.

18495 When using the *setrlimit()* function, if the requested new limit is RLIM\_INFINITY, the new limit  
 18496 shall be “no limit”; otherwise, if the requested new limit is RLIM\_SAVED\_MAX, the new limit  
 18497 shall be the corresponding saved hard limit; otherwise, if the requested new limit is  
 18498 RLIM\_SAVED\_CUR, the new limit shall be the corresponding saved soft limit; otherwise, the  
 18499 new limit shall be the requested value. In addition, if the corresponding saved limit can be  
 18500 represented correctly in an object of type **rlim\_t** then it shall be overwritten with the new limit.

18501 The result of setting a limit to RLIM\_SAVED\_MAX or RLIM\_SAVED\_CUR is unspecified unless  
 18502 a previous call to *getrlimit()* returned that value as the soft or hard limit for the corresponding  
 18503 resource limit.

18504 The determination of whether a limit can be correctly represented in an object of type **rlim\_t** is  
 18505 implementation-defined. For example, some implementations permit a limit whose value is  
 18506 greater than RLIM\_INFINITY and others do not.

18507 The *exec* family of functions also cause resource limits to be saved.

#### 18508 RETURN VALUE

18509 Upon successful completion, *getrlimit()* and *setrlimit()* shall return 0. Otherwise, these functions  
 18510 shall return -1 and set *errno* to indicate the error.

#### 18511 ERRORS

18512 The *getrlimit()* and *setrlimit()* functions shall fail if:

18513 [EINVAL] An invalid *resource* was specified; or in a *setrlimit()* call, the new *rlim\_cur*  
 18514 exceeds the new *rlim\_max*.

18515 [EPERM] The limit specified to *setrlimit()* would have raised the maximum limit value,  
 18516 and the calling process does not have appropriate privileges.

18517 The *setrlimit()* function may fail if:

18518 [EINVAL] The limit specified cannot be lowered because current usage is already higher  
 18519 than the limit.

#### 18520 EXAMPLES

18521 None.

#### 18522 APPLICATION USAGE

18523 If a process attempts to set the hard limit or soft limit for RLIMIT\_NOFILE to less than the value  
 18524 of {\_POSIX\_OPEN\_MAX} from <**limits.h**>, unexpected behavior may occur.

18525 **RATIONALE**

18526           None.

18527 **FUTURE DIRECTIONS**

18528           None.

18529 **SEE ALSO**18530           *exec*, *fork()*, *malloc()*, *open()*, *sigaltstack()*, *sysconf()*, *ulimit()*, the Base Definitions volume of

18531           IEEE Std. 1003.1-200x, &lt;stropts.h&gt;, &lt;sys/resource.h&gt;

18532 **CHANGE HISTORY**

18533           First released in Issue 4, Version 2.

18534 **Issue 5**

18535           Moved from X/OPEN UNIX extension to BASE and an APPLICATION USAGE section is added.

18536           Large File Summit extensions are added.

18537 **NAME**

18538           getrusage — get information about resource utilization

18539 **SYNOPSIS**

18540 XSI       #include &lt;sys/resource.h&gt;

18541       int getrusage(int *who*, struct rusage \**r\_usage*);

18542

18543 **DESCRIPTION**

18544       The *getrusage()* function provides measures of the resources used by the current process or its  
18545       terminated and waited-for child processes. If the value of the *who* argument is RUSAGE\_SELF,  
18546       information shall be returned about resources used by the current process. If the value of the *who*  
18547       argument is RUSAGE\_CHILDREN, information shall be returned about resources used by the  
18548       terminated and waited-for children of the current process. If the child is never waited for (for  
18549       example, if the parent has SA\_NOCLDWAIT set or sets SIGCHLD to SIG\_IGN), the resource  
18550       information for the child process is discarded and not included in the resource information  
18551       provided by *getrusage()*.

18552       The *r\_usage* argument is a pointer to an object of type **struct rusage** in which the returned  
18553       information is stored.

18554 **RETURN VALUE**

18555       Upon successful completion, *getrusage()* shall return 0; otherwise, -1 shall be returned and *errno*  
18556       set to indicate the error.

18557 **ERRORS**18558       The *getrusage()* function shall fail if:18559       [EINVAL]       The value of the *who* argument is not valid.18560 **EXAMPLES**18561       **Using getrusage()**

18562       The following example returns information about the resources used by the current process.

18563       #include &lt;sys/resource.h&gt;

18564       ...

18565       int who = RUSAGE\_SELF;

18566       struct rusage usage;

18567       int ret;

18568       ret = getrusage(who, &amp;usage);

18569 **APPLICATION USAGE**

18570       None.

18571 **RATIONALE**

18572       None.

18573 **FUTURE DIRECTIONS**

18574       None.

18575 **SEE ALSO**18576       *exit()*, *sigaction()*, *time()*, *times()*, *wait()*, the Base Definitions volume of IEEE Std. 1003.1-200x,

18577       &lt;sys/resource.h&gt;

18578 **CHANGE HISTORY**

18579 First released in Issue 4, Version 2.

18580 **Issue 5**

18581 Moved from X/OPEN UNIX extension to BASE.

18582 **NAME**

18583       gets — get a string from a stdin stream

18584 **SYNOPSIS**

18585       #include &lt;stdio.h&gt;

18586       char \*gets(char \*s);

18587 **DESCRIPTION**

18588 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
18589       conflict between the requirements described here and the ISO C standard is unintentional. This  
18590       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

18591       The *gets()* function shall read bytes from the standard input stream, *stdin*, into the array pointed  
18592       to by *s*, until a newline is read or an end-of-file condition is encountered. Any newline is  
18593       discarded and a null byte is placed immediately after the last byte read into the array.

18594 cx       The *gets()* function may mark the *st\_atime* field of the file associated with *stream* for update. The  
18595       *st\_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,  
18596       *fread()*, *getc()*, *getchar()*, *gets()*, *fscanf()*, or *scanf()* using *stream* that returns data not supplied by  
18597       a prior call to *ungetc()*.

18598 **RETURN VALUE**

18599       Upon successful completion, *gets()* shall return *s*. If the stream is at end-of-file, the end-of-file  
18600       indicator for the stream shall be set and *gets()* shall return a null pointer. If a read error occurs,  
18601 cx       the error indicator for the stream shall be set, *gets()* shall return a null pointer and set *errno* to  
18602       indicate the error.

18603 **ERRORS**18604       Refer to *fgetc()*.18605 **EXAMPLES**

18606       None.

18607 **APPLICATION USAGE**

18608       Reading a line that overflows the array pointed to by *s* results in undefined behavior. The use of  
18609       *fgets()* is recommended.

18610       Since the user cannot specify the length of the buffer passed to *gets()*, use of this function is  
18611       discouraged. The length of the string read is unlimited. It is possible to overflow this buffer in  
18612       such a way as to cause applications to fail, or possible system security violations.

18613       It is recommended that the *fgets()* function should be used to read input lines.

18614 **RATIONALE**

18615       None.

18616 **FUTURE DIRECTIONS**

18617       None.

18618 **SEE ALSO**18619       *feof()*, *ferror()*, *fgets()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**stdio.h**>18620 **CHANGE HISTORY**

18621       First released in Issue 1. Derived from Issue 1 of the SVID.

18622 **Issue 4**

18623       In the DESCRIPTION:

- 18624       • The text is changed to make it clear that the function reads bytes rather than (possibly multi-  
18625       byte) characters.

- 18626 • The list of functions that may cause the *st\_atime* field to be updated is revised.
- 18627 **Issue 6**
- 18628 Extensions beyond the ISO C standard are now marked.

18629 **NAME**

18630        getservbyname — network services database functions

18631 **SYNOPSIS**

18632        #include <netdb.h>

18633        struct servent \*getservbyname(const char \*name, const char \*proto);

18634 **DESCRIPTION**

18635        Refer to *endservent()*.



18636 **NAME**

18637        getservbyport — network services database functions

18638 **SYNOPSIS**

18639        #include &lt;netdb.h&gt;

18640        struct servent \*getservbyport(int *port*, const char \**proto*);18641 **DESCRIPTION**18642        Refer to *endservent()*.

18643 **NAME**

18644        getservent — network services database functions

18645 **SYNOPSIS**

18646        #include <netdb.h>

18647        struct servent \*getservent(void);

18648 **DESCRIPTION**

18649        Refer to *endservent()*.

18650 **NAME**

18651 getsid — get the process group ID of a session leader

18652 **SYNOPSIS**

18653 XSI #include &lt;unistd.h&gt;

18654 pid\_t getsid(pid\_t pid);

18655

18656 **DESCRIPTION**18657 The *getsid()* function obtains the process group ID of the process that is the session leader of the  
18658 process specified by *pid*. If *pid* is (**pid\_t**)0, it specifies the calling process.18659 **RETURN VALUE**18660 Upon successful completion, *getsid()* shall return the process group ID of the session leader of  
18661 the specified process. Otherwise, it shall return (**pid\_t**)-1 and set *errno* to indicate the error.18662 **ERRORS**18663 The *getsid()* function shall fail if:18664 [EPERM] The process specified by *pid* is not in the same session as the calling process,  
18665 and the implementation does not allow access to the process group ID of the  
18666 session leader of that process from the calling process.18667 [ESRCH] There is no process with a process ID equal to *pid*.18668 **EXAMPLES**

18669 None.

18670 **APPLICATION USAGE**

18671 None.

18672 **RATIONALE**

18673 None.

18674 **FUTURE DIRECTIONS**

18675 None.

18676 **SEE ALSO**18677 *exec*, *fork()*, *getpid()*, *getpgid()*, *setpgid()*, *setsid()*, the Base Definitions volume of  
18678 IEEE Std. 1003.1-200x, <unistd.h>18679 **CHANGE HISTORY**

18680 First released in Issue 4, Version 2.

18681 **Issue 5**

18682 Moved from X/OPEN UNIX extension to BASE.

18683 **NAME**

18684 getsockname — get the socket name

18685 **SYNOPSIS**

18686 #include &lt;sys/socket.h&gt;

18687 int getsockname(int *socket*, struct sockaddr \*restrict *address*,  
18688 socklen\_t \**address\_len*);18689 **DESCRIPTION**18690 The *getsockname()* function shall retrieve the locally-bound name of the specified socket, store  
18691 this address in the **sockaddr** structure pointed to by the *address* argument, and store the length of  
18692 this address in the object pointed to by the *address\_len* argument.18693 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
18694 the stored address shall be truncated.18695 If the socket has not been bound to a local name, the value stored in the object pointed to by  
18696 *address* is unspecified.18697 **RETURN VALUE**18698 Upon successful completion, 0 shall be returned, the *address* argument shall point to the address  
18699 of the socket, and the *address\_len* argument shall point to the length of the address. Otherwise, -1  
18700 shall be returned and *errno* set to indicate the error.18701 **ERRORS**18702 The *getsockname()* function shall fail if:18703 [EBADF] The *socket* argument is not a valid file descriptor.18704 [ENOTSOCK] The *socket* argument does not refer to a socket.

18705 [EOPNOTSUPP] The operation is not supported for this socket's protocol.

18706 The *getsockname()* function may fail if:

18707 [EINVAL] The socket has been shut down.

18708 [ENOBUFS] Insufficient resources were available in the system to complete the function.

18709 **EXAMPLES**

18710 None.

18711 **APPLICATION USAGE**

18712 None.

18713 **RATIONALE**

18714 None.

18715 **FUTURE DIRECTIONS**

18716 None.

18717 **SEE ALSO**18718 *accept()*, *bind()*, *getpeername()*, *socket()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
18719 <sys/socket.h>18720 **CHANGE HISTORY**

18721 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

18722 The **restrict** keyword is added to the *getsockname()* prototype for alignment with the  
18723 ISO/IEC 9899:1999 standard.

18724 **NAME**

18725 getsockopt — get the socket options

18726 **SYNOPSIS**

18727 #include &lt;sys/socket.h&gt;

```
18728 int getsockopt(int socket, int level, int option_name,
18729 void *restrict option_value, socklen_t *restrict option_len);
```

18730 **DESCRIPTION**18731 The *getsockopt()* function manipulates options associated with a socket.

18732 The *getsockopt()* function shall retrieve the value for the option specified by the *option\_name*  
 18733 argument for the socket specified by the *socket* argument. If the size of the option value is greater  
 18734 than *option\_len*, the value stored in the object pointed to by the *option\_value* argument shall be  
 18735 silently truncated. Otherwise, the object pointed to by the *option\_len* argument shall be modified  
 18736 to indicate the actual length of the value.

18737 The *level* argument specifies the protocol level at which the option resides. To retrieve options at  
 18738 the socket level, specify the *level* argument as SOL\_SOCKET. To retrieve options at other levels,  
 18739 supply the appropriate level identifier for the protocol controlling the option. For example, to  
 18740 indicate that an option is interpreted by the TCP (Transmission Control Protocol), set *level* to  
 18741 IPPROTO\_TCP as defined in the <netinet/in.h> header.

18742 The socket in use may require the process to have appropriate privileges to use the *getsockopt()*  
 18743 function.

18744 The *option\_name* argument specifies a single option to be retrieved. It can be one of the following  
 18745 values defined in <sys/socket.h>:

18746 SO\_DEBUG Reports whether debugging information is being recorded. This option  
 18747 stores an **int** value. This is a Boolean option.

18748 SO\_ACCEPTCONN Reports whether socket listening is enabled. This option stores an **int**  
 18749 value. This is a Boolean option.

18750 SO\_BROADCAST Reports whether transmission of broadcast messages is supported, if this  
 18751 is supported by the protocol. This option stores an **int** value. This is a  
 18752 Boolean option.

18753 SO\_REUSEADDR Reports whether the rules used in validating addresses supplied to *bind()*  
 18754 should allow reuse of local addresses, if this is supported by the protocol.  
 18755 This option stores an **int** value. This is a Boolean option.

18756 SO\_KEEPALIVE Reports whether connections are kept active with periodic transmission  
 18757 of messages, if this is supported by the protocol.

18758 If the connected socket fails to respond to these messages, the connection  
 18759 is broken and threads writing to that socket are notified with a SIGPIPE  
 18760 signal. This option stores an **int** value.

18761 This is a Boolean option.

18762 SO\_LINGER Reports whether the socket lingers on *close()* if data is present. If  
 18763 SO\_LINGER is set, the system blocks the process during *close()* until it  
 18764 can transmit the data or until the end of the interval indicated by the  
 18765 *l\_linger* member, whichever comes first. If SO\_LINGER is not specified,  
 18766 and *close()* is issued, the system handles the call in a way that allows the  
 18767 process to continue as quickly as possible. This option stores a **linger**  
 18768 structure.

|       |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 18769 | SO_OOBINLINE | Reports whether the socket leaves received out-of-band data (data marked urgent) inline. This option stores an <b>int</b> value. This is a Boolean option.                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18770 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18771 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18772 | SO_SNDBUF    | Reports send buffer size information. This option stores an <b>int</b> value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 18773 | SO_RCVBUF    | Reports receive buffer size information. This option stores an <b>int</b> value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 18774 | SO_ERROR     | Reports information about error status and clears it. This option stores an <b>int</b> value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 18775 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18776 | SO_TYPE      | Reports the socket type. This option stores an <b>int</b> value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 18777 | SO_DONTROUTE | Reports whether outgoing messages bypass the standard routing facilities. The destination shall be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option stores an <b>int</b> value. This is a Boolean option.                                                                                                                                                                                                                             |
| 18778 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18779 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18780 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18781 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18782 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18783 | SO_RCVLOWAT  | Reports the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different than that returned; for example, out-of-band data.) This option stores an <b>int</b> value. Note that not all implementations allow this option to be retrieved. |
| 18784 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18785 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18786 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18787 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18788 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18789 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18790 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18791 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18792 | SO_RCVTIMEO  | Reports the timeout value for input operations. This option stores a <b>timeval</b> structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data was received. The default for this option is zero, which indicates that a receive operation shall not time out. Note that not all implementations allow this option to be retrieved.         |
| 18793 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18794 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18795 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18796 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18797 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18798 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18799 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18800 | SO_SNDLOWAT  | Reports the minimum number of bytes to process for socket output operations. Non-blocking output operations shall process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option stores an <b>int</b> value. Note that not all implementations allow this option to be retrieved.                                                                                                                                                                                                                                             |
| 18801 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18802 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18803 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18804 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18805 | SO_SNDTIMEO  | Reports the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent. If a send operation has blocked for this time, it shall return with a partial count or with <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data were sent. The default for this option is zero, which indicates that a send operation shall not time out. The option stores a <b>timeval</b> structure. Note that not all implementations allow this option to be retrieved.                                                                                  |
| 18806 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18807 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18808 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18809 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18810 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18811 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18812 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18813 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 18814 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|       |              | For Boolean options, a zero value indicates that the option is disabled and a non-zero value indicates that the option is enabled.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|       |              | Options at other protocol levels vary in format and name.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

18815 The socket in use may require the process to have appropriate privileges to use the *getsockopt()*  
18816 function.

#### 18817 RETURN VALUE

18818 Upon successful completion, *getsockopt()* shall return 0; otherwise, -1 shall be returned and *errno*  
18819 set to indicate the error.

#### 18820 ERRORS

18821 The *getsockopt()* function shall fail if:

18822 [EBADF] The *socket* argument is not a valid file descriptor.

18823 [EINVAL] The specified option is invalid at the specified socket level.

18824 [ENOPROTOOPT]

18825 The option is not supported by the protocol.

18826 [ENOTSOCK] The *socket* argument does not refer to a socket.

18827 The *getsockopt()* function may fail if:

18828 [EACCES] The calling process does not have the appropriate privileges.

18829 [EINVAL] The socket has been shut down.

18830 [ENOBUFS] Insufficient resources are available in the system to complete the function.

#### 18831 EXAMPLES

18832 None.

#### 18833 APPLICATION USAGE

18834 None.

#### 18835 RATIONALE

18836 None.

#### 18837 FUTURE DIRECTIONS

18838 None.

#### 18839 SEE ALSO

18840 *bind()*, *close()*, *endprotoent()*, *setsockopt()*, *socket()*, the Base Definitions volume of  
18841 IEEE Std. 1003.1-200x, <sys/socket.h>, <netinet/in.h>

#### 18842 CHANGE HISTORY

18843 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

18844 The **restrict** keyword is added to the *getsockopt()* prototype for alignment with the  
18845 ISO/IEC 9899:1999 standard.

18846 **NAME**

18847       getsubopt — parse suboption arguments from a string

18848 **SYNOPSIS**

18849 XSI       #include &lt;stdlib.h&gt;

18850       int getsubopt(char \*\*optionp, char \* const \*tokens, char \*\*valuep);

18851

18852 **DESCRIPTION**

18853       The *getsubopt()* function parses suboption arguments in a flag argument that was initially parsed  
 18854       by *getopt()*. The application shall ensure that these suboption arguments are separated by  
 18855       commas and may consist of either a single token, or a token-value pair separated by an equal  
 18856       sign. Because commas delimit suboption arguments in the option string, they are not allowed to  
 18857       be part of the suboption arguments or the value of a suboption argument. Similarly, because the  
 18858       equal sign separates a token from its value, undefined behavior will result if the application  
 18859       passes a token that contains an equal sign.

18860       The *getsubopt()* function takes the address of a pointer to the option argument string, a vector of  
 18861       possible tokens, and the address of a value string pointer. If the option argument string at  
 18862       \**optionp* contains only one suboption argument, *getsubopt()* updates \**optionp* to point to the null  
 18863       at the end of the string. Otherwise, it isolates the suboption argument by replacing the comma  
 18864       separator with a null, and updates \**optionp* to point to the start of the next suboption argument.  
 18865       If the suboption argument has an associated value, *getsubopt()* updates \**valuep* to point to the  
 18866       value's first character. Otherwise, it sets \**valuep* to a null pointer.

18867       The token vector is organized as a series of pointers to strings. The end of the token vector is  
 18868       identified by a null pointer.

18869       When *getsubopt()* returns, if \**valuep* is not a null pointer, then the suboption argument processed  
 18870       included a value. The calling program may use this information to determine whether the  
 18871       presence or lack of a value for this suboption is an error.

18872       Additionally, when *getsubopt()* fails to match the suboption argument with the tokens in the  
 18873       *tokens* array, the calling program should decide if this is an error, or if the unrecognized option  
 18874       should be passed on to another program.

18875 **RETURN VALUE**

18876       The *getsubopt()* function shall return the index of the matched token string, or -1 if no token  
 18877       strings were matched.

18878 **ERRORS**

18879       No errors are defined.

18880 **EXAMPLES**

```
18881 #include <stdio.h>
18882 #include <stdlib.h>
18883 int do_all;
18884 const char *type;
18885 int read_size;
18886 int write_size;
18887 int read_only;
18888 enum
18889 {
18890 RO_OPTION = 0,
18891 RW_OPTION,
```



```

18892 READ_SIZE_OPTION,
18893 WRITE_SIZE_OPTION
18894 };
18895 const char *mount_opts[] =
18896 {
18897 [RO_OPTION] = "ro",
18898 [RW_OPTION] = "rw",
18899 [READ_SIZE_OPTION] = "rsize",
18900 [WRITE_SIZE_OPTION] = "wsize",
18901 NULL
18902 };
18903 int
18904 main(int argc, char *argv[])
18905 {
18906 char *subopts, *value;
18907 int opt;
18908
18909 while ((opt = getopt(argc, argv, "at:o:")) != -1)
18910 switch(opt)
18911 {
18912 case 'a':
18913 do_all = 1;
18914 break;
18915 case 't':
18916 type = optarg;
18917 break;
18918 case 'o':
18919 subopts = optarg;
18920 while (*subopts != ' ')
18921 switch(getsubopt(&subopts, mount_opts, &value))
18922 {
18923 case RO_OPTION:
18924 read_only = 1;
18925 break;
18926 case RW_OPTION:
18927 read_only = 0;
18928 break;
18929 case READ_SIZE_OPTION:
18930 if (value == NULL)
18931 abort();
18932 read_size = atoi(value);
18933 break;
18934 case WRITE_SIZE_OPTION:
18935 if (value == NULL)
18936 abort();
18937 write_size = atoi(value);
18938 break;
18939 default:
18940 /* Unknown suboption. */
18941 printf("Unknown suboption '%s'\n", value);
18942 break;
18943 }
18944 }

```

```

18943 break;
18944 default:
18945 abort();
18946 }
18947 /* Do the real work. */
18948 return 0;
18949 }

```

### 18950 Parsing Suboptions

18951 The following example uses the *getsubopt()* function to parse a value argument in the *optarg*  
 18952 external variable returned by a call to *getopt()*.

```

18953 #include <stdlib.h>
18954 ...
18955 char *tokens[] = {"HOME", "PATH", "LOGNAME", (char *) NULL };
18956 char *value;
18957 int opt, index;
18958 while ((opt = getopt(argc, argv, "e:")) != -1) {
18959 switch(opt) {
18960 case 'e' :
18961 while ((index = getsubopt(&optarg, tokens, &value)) != -1) {
18962 switch(index) {
18963 ...
18964 }
18965 break;
18966 ...
18967 }
18968 }
18969 ...

```

### 18970 APPLICATION USAGE

18971 None.

### 18972 RATIONALE

18973 None.

### 18974 FUTURE DIRECTIONS

18975 None.

### 18976 SEE ALSO

18977 *getopt()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>

### 18978 CHANGE HISTORY

18979 First released in Issue 4, Version 2.

### 18980 Issue 5

18981 Moved from X/OPEN UNIX extension to BASE.

18982 **NAME**

18983           gettimeofday — get the date and time

18984 **SYNOPSIS**

18985 XSI       #include &lt;sys/time.h&gt;

18986           int gettimeofday(struct timeval \*restrict *tp*, void \*tzp);

18987

18988 **DESCRIPTION**

18989       The *gettimeofday()* function obtains the current time, expressed as seconds and microseconds since the Epoch, and stores it in the **timeval** structure pointed to by *tp*. The resolution of the system clock is unspecified.

18992       If *tzp* is not a null pointer, the behavior is unspecified.

18993 **RETURN VALUE**

18994       The *gettimeofday()* function shall return 0 and no value shall be reserved to indicate an error.

18995 **ERRORS**

18996       No errors are defined.

18997 **EXAMPLES**

18998       None.

18999 **APPLICATION USAGE**

19000       None.

19001 **RATIONALE**

19002       None.

19003 **FUTURE DIRECTIONS**

19004       None.

19005 **SEE ALSO**

19006       *ctime()*, *ftime()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/time.h>

19007 **CHANGE HISTORY**

19008       First released in Issue 4, Version 2.

19009 **Issue 5**

19010       Moved from X/OPEN UNIX extension to BASE.

19011 **Issue 6**

19012       The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time* functions.

19015 **NAME**

19016       getuid — get a real user ID

19017 **SYNOPSIS**

19018       #include <unistd.h>

19019       uid\_t getuid(void);

19020 **DESCRIPTION**

19021       The *getuid()* function shall return the real user ID of the calling process.

19022 **RETURN VALUE**

19023       The *getuid()* function is always successful and no return value is reserved to indicate the error.

19024 **ERRORS**

19025       No errors are defined.

19026 **EXAMPLES**19027       **Setting the Effective User ID to the Real User ID**

19028       The following example sets the effective user ID and the real user ID of the current process to the  
19029       real user ID of the caller.

```
19030 #include <unistd.h>
19031 #include <sys/types.h>
19032 ...
19033 setreuid(getuid(), getuid());
19034 ...
```

19035 **APPLICATION USAGE**

19036       None.

19037 **RATIONALE**

19038       None.

19039 **FUTURE DIRECTIONS**

19040       None.

19041 **SEE ALSO**

19042       *getegid()*, *geteuid()*, *getgid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base  
19043       Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>, <unistd.h>

19044 **CHANGE HISTORY**

19045       First released in Issue 1. Derived from Issue 1 of the SVID.

19046 **Issue 4**

19047       The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
19048       XSI-conformant systems.

19049       The <unistd.h> header is added to the SYNOPSIS section.

19050       The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 19051       • The argument list is explicitly defined as **void**.

19052 **Issue 6**

19053       In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

19054       The following new requirements on POSIX implementations derive from alignment with the  
19055       Single UNIX Specification:

19056  
19057  
19058

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

19059 **NAME**

19060           getutxent, getutxid, getutxline — get user accounting database entries

19061 **SYNOPSIS**

19062 XSI       #include <utmpx.h>

19063           struct utmpx \*getutxent(void);

19064           struct utmpx \*getutxid(const struct utmpx \*id);

19065           struct utmpx \*getutxline(const struct utmpx \*line);

19066

19067 **DESCRIPTION**

19068           Refer to *endutxent()*.

19069 **NAME**19070 `getwc` — get a wide character from a stream19071 **SYNOPSIS**19072 `#include <stdio.h>`19073 `#include <wchar.h>`19074 `wint_t getwc(FILE *stream);`19075 **DESCRIPTION**

19076 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
19077 conflict between the requirements described here and the ISO C standard is unintentional. This  
19078 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

19079 The `getwc()` function is equivalent to `fgetwc()`, except that if it is implemented as a macro it may  
19080 evaluate *stream* more than once, so the argument should never be an expression with side effects.

19081 **RETURN VALUE**19082 Refer to `fgetwc()`.19083 **ERRORS**19084 Refer to `fgetwc()`.19085 **EXAMPLES**

19086 None.

19087 **APPLICATION USAGE**

19088 Because it may be implemented as a macro, `getwc()` may treat incorrectly a *stream* argument with  
19089 side effects. In particular, `getwc(*f++)` does not necessarily work as expected. Therefore, use of  
19090 this function is not recommended; `fgetwc()` should be used instead.

19091 **RATIONALE**

19092 None.

19093 **FUTURE DIRECTIONS**

19094 None.

19095 **SEE ALSO**19096 `fgetwc()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<stdio.h>`, `<wchar.h>`19097 **CHANGE HISTORY**

19098 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working  
19099 draft.

19100 **Issue 5**19101 The Optional Header (OH) marking is removed from `<stdio.h>`.

19102 **NAME**

19103       getwchar — get a wide character from a stdin stream

19104 **SYNOPSIS**

19105       #include <wchar.h>

19106       wint\_t getwchar(void);

19107 **DESCRIPTION**

19108 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
19109       conflict between the requirements described here and the ISO C standard is unintentional. This  
19110       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

19111       The *getwchar()* function is equivalent to *getwc(stdin)*.

19112 **RETURN VALUE**

19113       Refer to *fgetwc()*.

19114 **ERRORS**

19115       Refer to *fgetwc()*.

19116 **EXAMPLES**

19117       None.

19118 **APPLICATION USAGE**

19119       If the **wint\_t** value returned by *getwchar()* is stored into a variable of type **wchar\_t** and then  
19120       compared against the **wint\_t** macro WEOF, the result may be incorrect. Only the **wint\_t** type is  
19121       guaranteed to be able to represent any wide character and WEOF.

19122 **RATIONALE**

19123       None.

19124 **FUTURE DIRECTIONS**

19125       None.

19126 **SEE ALSO**

19127       *fgetwc()*, *getwc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**wchar.h**>

19128 **CHANGE HISTORY**

19129       First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working  
19130       draft.



19131 **NAME**

19132 getwd — get the current working directory path name (**LEGACY**)

19133 **SYNOPSIS**

```
19134 xSI #include <unistd.h>
```

```
19135 char *getwd(char *path_name);
```

19136

19137 **DESCRIPTION**

19138 The *getwd()* function determines an absolute path name of the current working directory of the  
19139 calling process, and copies that path name into the array pointed to by the *path\_name* argument.

19140 If the length of the path name of the current working directory is greater than ( $\{\text{PATH\_MAX}\}+1$ )  
19141 including the null byte, *getwd()* shall fail and return a null pointer.

19142 **RETURN VALUE**

19143 Upon successful completion, a pointer to the string containing the absolute path name of the  
19144 current working directory shall be returned. Otherwise, *getwd()* shall return a null pointer and  
19145 the contents of the array pointed to by *path\_name* are undefined.

19146 **ERRORS**

19147 No errors are defined.

19148 **EXAMPLES**

19149 None.

19150 **APPLICATION USAGE**

19151 For applications portability, the *getcwd()* function should be used to determine the current  
19152 working directory instead of *getwd()*.

19153 **RATIONALE**

19154 Since the user cannot specify the length of the buffer passed to *getwd()*, use of this function is  
19155 discouraged. The length of a path name described in  $\{\text{PATH\_MAX}\}$  is file system-dependent and  
19156 may vary from one mount point to another, or might even be unlimited. It is possible to  
19157 overflow this buffer in such a way as to cause applications to fail, or possible system security  
19158 violations.

19159 It is recommended that the *getcwd()* function should be used to determine the current working  
19160 directory.

19161 **FUTURE DIRECTIONS**

19162 This function may be withdrawn in a future version.

19163 **SEE ALSO**

19164 *getcwd()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<unistd.h>**

19165 **CHANGE HISTORY**

19166 First released in Issue 4, Version 2.

19167 **Issue 5**

19168 Moved from X/OPEN UNIX extension to BASE.

19169 **Issue 6**

19170 This function is marked LEGACY.

19171 **NAME**

19172 glob, globfree — generate path names matching a pattern

19173 **SYNOPSIS**

```
19174 #include <glob.h>
19175 int glob(const char *restrict pattern, int flags,
19176 int(*restrict errfunc)(const char *restrict epath, int eerrno),
19177 glob_t *restrict pglob);
19178 void globfree(glob_t *pglob);
```

19179 **DESCRIPTION**

19180 The *glob()* function is a path name generator that implements the rules defined in the Shell and  
 19181 Utilities volume of IEEE Std. 1003.1-200x, Section 2.14, Pattern Matching Notation, with optional  
 19182 support for rule 3 in the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.14.3,  
 19183 Patterns Used for File Name Expansion.

19184 The structure type **glob\_t** is defined in **<glob.h>** and includes at least the following members:

19185  
 19186  
 19187  
 19188  
 19189

| Member Type | Member Name | Description                                            |
|-------------|-------------|--------------------------------------------------------|
| size_t      | gl_pathc    | Count of paths matched by <i>pattern</i> .             |
| char **     | gl_pathv    | Pointer to a list of matched path names.               |
| size_t      | gl_offs     | Slots to reserve at the beginning of <i>gl_pathv</i> . |

19190 The argument *pattern* is a pointer to a path name pattern to be expanded. The *glob()* function  
 19191 matches all accessible path names against this pattern and develops a list of all path names that  
 19192 match. In order to have access to a path name, *glob()* requires search permission on every  
 19193 component of a path except the last, and read permission on each directory of any file name  
 19194 component of *pattern* that contains any of the following special characters: '\*', '?', and '['.

19195 The *glob()* function stores the number of matched path names into *pglob->gl\_pathc* and a pointer  
 19196 to a list of pointers to path names into *pglob->gl\_pathv*. The path names are in sort order as  
 19197 defined by the current setting of the *LC\_COLLATE* category; see the Base Definitions volume of  
 19198 IEEE Std. 1003.1-200x, Section 7.3.2, *LC\_COLLATE*. The first pointer after the last path name is a  
 19199 null pointer. If the pattern does not match any path names, the returned number of matched  
 19200 paths is set to 0, and the contents of *pglob->gl\_pathv* are implementation-defined.

19201 It is the caller's responsibility to create the structure pointed to by *pglob*. The *glob()* function  
 19202 allocates other space as needed, including the memory pointed to by *gl\_pathv*. The *globfree()*  
 19203 function frees any space associated with *pglob* from a previous call to *glob()*.

19204 The *flags* argument is used to control the behavior of *glob()*. The value of *flags* is a bitwise-  
 19205 inclusive OR of zero or more of the following constants, which are defined in **<glob.h>**:

- 19206 **GLOB\_APPEND** Append path names generated to the ones from a previous call to *glob()*.
- 19207 **GLOB\_DOOFFS** Make use of *pglob->gl\_offs*. If this flag is set, *pglob->gl\_offs* is used to  
 19208 specify how many null pointers to add to the beginning of *pglob->gl\_pathv*. In other words,  
 19209 *pglob->gl\_pathv* shall point to *pglob->gl\_offs* null pointers, followed by *pglob->gl\_pathc*  
 19210 path name pointers, followed by a null pointer.  
 19211
- 19212 **GLOB\_ERR** Causes *glob()* to return when it encounters a directory that it cannot open  
 19213 or read. Ordinarily, *glob()* continues to find matches.
- 19214 **GLOB\_MARK** Each path name that is a directory that matches *pattern* has a slash  
 19215 appended.

|       |               |                                                                                                                                                                                                                                                                                                          |
|-------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 19216 | GLOB_NOCHECK  | Supports rule 3 in the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.14.3, Patterns Used for File Name Expansion. If <i>pattern</i> does not match any path name, then <i>glob()</i> shall return a list consisting of only <i>pattern</i> , and the number of matched path names is 1. |
| 19217 |               |                                                                                                                                                                                                                                                                                                          |
| 19218 |               |                                                                                                                                                                                                                                                                                                          |
| 19219 |               |                                                                                                                                                                                                                                                                                                          |
| 19220 | GLOB_NOESCAPE | Disable backslash escaping.                                                                                                                                                                                                                                                                              |
| 19221 | GLOB_NOSORT   | Ordinarily, <i>glob()</i> sorts the matching path names according to the current setting of the <i>LC_COLLATE</i> category, see the Base Definitions volume of IEEE Std. 1003.1-200x, Section 7.3.2, <i>LC_COLLATE</i> . When this flag is used, the order of path names returned is unspecified.        |
| 19222 |               |                                                                                                                                                                                                                                                                                                          |
| 19223 |               |                                                                                                                                                                                                                                                                                                          |
| 19224 |               |                                                                                                                                                                                                                                                                                                          |

The GLOB\_APPEND flag can be used to append a new set of path names to those found in a previous call to *glob()*. The following rules apply to applications when two or more calls to *glob()* are made with the same value of *pglob* and without intervening calls to *globfree()*:

- 19228 1. The first such call shall not set GLOB\_APPEND. All subsequent calls shall set it.
- 19229 2. All the calls shall set GLOB\_DOOFFS, or all shall not set it.
- 19230 3. After the second call, *pglob->gl\_pathv* points to a list containing the following:
  - 19231 a. Zero or more null pointers, as specified by GLOB\_DOOFFS and *pglob->gl\_offs*.
  - 19232 b. Pointers to the path names that were in the *pglob->gl\_pathv* list before the call, in the same order as before.
  - 19233 c. Pointers to the new path names generated by the second call, in the specified order.
- 19234 4. The count returned in *pglob->gl\_pathc* shall be the total number of path names from the two calls.
- 19235 5. The application can change any of the fields after a call to *glob()*. If it does, the application shall reset them to the original value before a subsequent call, using the same *pglob* value, to *globfree()* or *glob()* with the GLOB\_APPEND flag.

19240 If, during the search, a directory is encountered that cannot be opened or read and *errfunc* is not  
 19241 a null pointer, *glob()* calls (*\*errfunc()*) with two arguments:

- 19242 1. The *epath* argument is a pointer to the path that failed.
- 19243 2. The *errno* argument is the value of *errno* from the failure, as set by *opendir()*, *readdir()*, or  
 19244 *stat()*. (Other values may be used to report other errors not explicitly documented for  
 19245 those functions.)

19246 The following constants are defined as error return values for *glob()*:

|       |              |                                                                                           |
|-------|--------------|-------------------------------------------------------------------------------------------|
| 19247 | GLOB_ABORTED | The scan was stopped because GLOB_ERR was set or ( <i>*errfunc()</i> ) returned non-zero. |
| 19248 |              |                                                                                           |
| 19249 | GLOB_NOMATCH | The pattern does not match any existing path name, and GLOB_NOCHECK was not set in flags. |
| 19250 |              |                                                                                           |
| 19251 | GLOB_NOSPACE | An attempt to allocate memory failed.                                                     |

19252 If (*\*errfunc()*) is called and returns non-zero, or if the GLOB\_ERR flag is set in *flags*, *glob()* shall  
 19253 stop the scan and return GLOB\_ABORTED after setting *gl\_pathc* and *gl\_pathv* in *pglob* to reflect  
 19254 the paths already scanned. If GLOB\_ERR is not set and either *errfunc* is a null pointer or  
 19255 (*\*errfunc()*) returns 0, the error shall be ignored.

## 19256 RETURN VALUE

19257 Upon successful completion, *glob()* shall return 0. The argument *pglob->gl\_pathc* shall return the  
 19258 number of matched path names and the argument *pglob->gl\_pathv* shall contain a pointer to a  
 19259 null-terminated list of matched and sorted path names. However, if *pglob->gl\_pathc* is 0, the  
 19260 content of *pglob->gl\_pathv* is undefined.

19261 The *globfree()* function shall return no value.

19262 If *glob()* terminates due to an error, it shall return one of the non-zero constants defined in  
 19263 `<glob.h>`. The arguments *pglob->gl\_pathc* and *pglob->gl\_pathv* are still set as defined above.

## 19264 ERRORS

19265 No errors are defined.

## 19266 EXAMPLES

19267 One use of the GLOB\_DOOFFS flag is by applications that build an argument list for use with  
 19268 *execv()*, *execve()*, or *execvp()*. Suppose, for example, that an application wants to do the  
 19269 equivalent of:

```
19270 ls -l *.c
```

19271 but for some reason:

```
19272 system("ls -l *.c")
```

19273 is not acceptable. The application could obtain approximately the same result using the  
 19274 sequence:

```
19275 globbuf.gl_offs = 2;
19276 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
19277 globbuf.gl_pathv[0] = "ls";
19278 globbuf.gl_pathv[1] = "-l";
19279 execvp("ls", &globbuf.gl_pathv[0]);
```

19280 Using the same example:

```
19281 ls -l *.c *.h
```

19282 could be approximately simulated using GLOB\_APPEND as follows:

```
19283 globbuf.gl_offs = 2;
19284 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
19285 glob("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
19286 ...
```

## 19287 APPLICATION USAGE

19288 This function is not provided for the purpose of enabling utilities to perform path name  
 19289 expansion on their arguments, as this operation is performed by the shell, and utilities are  
 19290 explicitly not expected to redo this. Instead, it is provided for applications that need to do path  
 19291 name expansion on strings obtained from other sources, such as a pattern typed by a user or  
 19292 read from a file.

19293 If a utility needs to see if a path name matches a given pattern, it can use *fnmatch()*.

19294 Note that *gl\_pathc* and *gl\_pathv* have meaning even if *glob()* fails. This allows *glob()* to report  
 19295 partial results in the event of an error. However, if *gl\_pathc* is 0, *gl\_pathv* is unspecified even if  
 19296 *glob()* did not return an error.

19297 The GLOB\_NOCHECK option could be used when an application wants to expand a path name  
 19298 if wildcards are specified, but wants to treat the pattern as just a string otherwise. The *sh* utility  
 19299 might use this for option-arguments, for example.

19300 The new path names generated by a subsequent call with GLOB\_APPEND are not sorted  
 19301 together with the previous path names. This mirrors the way that the shell handles path name  
 19302 expansion when multiple expansions are done on a command line.

19303 Applications that need tilde and parameter expansion should use *wordexp()*.

#### 19304 RATIONALE

19305 It was claimed that the GLOB\_DOOFFS flag is unnecessary because it could be simulated using:

```
19306 new = (char **)malloc((n + pglob->gl_pathc + 1)
19307 * sizeof(char *));
19308 (void) memcpy(new+n, pglob->gl_pathv,
19309 pglob->gl_pathc * sizeof(char *));
19310 (void) memset(new, 0, n * sizeof(char *));
19311 free(pglob->gl_pathv);
19312 pglob->gl_pathv = new;
```

19313 However, this assumes that the memory pointed to by *gl\_pathv* is a block that was separately  
 19314 created using *malloc()*. This is not necessarily the case. An application should make no  
 19315 assumptions about how the memory referenced by fields in *pglob* was allocated. It might have  
 19316 been obtained from *malloc()* in a large chunk and then carved up within *glob()*, or it might have  
 19317 been created using a different memory allocator. It is not the intent of the standard developers to  
 19318 specify or imply how the memory used by *glob()* is managed.

19319 The GLOB\_APPEND flag would be used when an application wants to expand several different  
 19320 patterns into a single list.

#### 19321 FUTURE DIRECTIONS

19322 None.

#### 19323 SEE ALSO

19324 *exec*, *fnmatch()*, *opendir()*, *readdir()*, *stat()*, *wordexp()*, the Base Definitions volume of  
 19325 IEEE Std. 1003.1-200x, <*glob.h*>, the Shell and Utilities volume of IEEE Std. 1003.1-200x

#### 19326 CHANGE HISTORY

19327 First released in Issue 4. Derived from the ISO POSIX-2 standard.

#### 19328 Issue 5

19329 Moved from POSIX2 C-language Binding to BASE.

#### 19330 Issue 6

19331 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

19332 The **restrict** keyword is added to the *glob()* prototype for alignment with the ISO/IEC 9899:1999  
 19333 standard.

## 19334 NAME

19335 gmtime, gmtime\_r — convert a time value to a broken-down UTC time

## 19336 SYNOPSIS

19337 #include <time.h>

19338 struct tm \*gmtime(const time\_t \*timer);

19339 TSF struct tm \*gmtime\_r(const time\_t \*restrict timer, struct tm \*restrict result);

19340

## 19341 DESCRIPTION

19342 CX For *gmtime()*: The functionality described on this reference page is aligned with the ISO C  
19343 standard. Any conflict between the requirements described here and the ISO C standard is  
19344 unintentional. This volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

19345 The *gmtime()* function shall convert the time in seconds since the Epoch pointed to by *timer* into  
19346 a broken-down time, expressed as Coordinated Universal Time (UTC).

19347 CX The *gmtime()* function need not be reentrant. A function that is not required to be reentrant is not  
19348 required to be thread-safe.

19349 TSF The *gmtime\_r()* function shall convert the time in seconds since the Epoch pointed to by *timer*  
19350 into a broken-down time expressed as Coordinated Universal Time (UTC). The broken-down  
19351 time is stored in the structure referred to by *result*. The *gmtime\_r()* function shall also return the  
19352 address of the same structure.

## 19353 RETURN VALUE

19354 The *gmtime()* function shall return a pointer to a **struct tm**.

19355 TSF Upon successful completion, *gmtime\_r()* shall return the address of the structure pointed to by  
19356 the argument *result*. If an error is detected, or UTC is not available, *gmtime\_r()* shall return a null  
19357 pointer.

## 19358 ERRORS

19359 No errors are defined.

## 19360 EXAMPLES

19361 None.

## 19362 APPLICATION USAGE

19363 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions return values in one of two static  
19364 objects: a broken-down time structure and an array of **char**. Execution of any of the functions  
19365 may overwrite the information returned in either of these objects by any of the other functions.

19366 The *gmtime\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
19367 of possibly using a static data area that may be overwritten by each call.

## 19368 RATIONALE

19369 None.

## 19370 FUTURE DIRECTIONS

19371 None.

## 19372 SEE ALSO

19373 *asctime()*, *clock()*, *ctime()*, *difftime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,  
19374 the Base Definitions volume of IEEE Std. 1003.1-200x, <**time.h**>

19375 **CHANGE HISTORY**

19376 First released in Issue 1. Derived from Issue 1 of the SVID.

19377 **Issue 4**

19378 In the APPLICATION USAGE section, the list of functions with which this function may interact  
19379 is revised and the wording clarified.

19380 The following change is incorporated for alignment with the ISO C standard:

- 19381 • The type of argument *timer* is changed from **time\_t\*** to **const time\_t\***.

19382 **Issue 5**

19383 A note indicating that the *gmtime()* function need not be reentrant is added to the  
19384 DESCRIPTION.

19385 The *gmtime\_r()* function is included for alignment with the POSIX Threads Extension.

19386 **Issue 6**

19387 The *gmtime\_r()* function is marked as part of the Thread-Safe Functions option.

19388 Extensions beyond the ISO C standard are now marked.

19389 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
19390 its avoidance of possibly using a static data area.

19391 The **restrict** keyword is added to the *gmtime\_r()* prototype for alignment with the  
19392 ISO/IEC 9899:1999 standard.

19393 **NAME**

19394 grantpt — grant access to the slave pseudo-terminal device

19395 **SYNOPSIS**19396 XSI `#include <stdlib.h>`19397 `int grantpt(int fildes);`

19398

19399 **DESCRIPTION**

19400 The *grantpt()* function shall change the mode and ownership of the slave pseudo-terminal  
 19401 device associated with its master pseudo-terminal counterpart. The *fildes* argument is a file  
 19402 descriptor that refers to a master pseudo-terminal device. The user ID of the slave shall be set to  
 19403 the real UID of the calling process and the group ID shall be set to an unspecified group ID. The  
 19404 permission mode of the slave pseudo-terminal shall be set to readable and writable by the  
 19405 owner, and writable by the group.

19406 The behavior of the *grantpt()* function is unspecified if the application has installed a signal  
 19407 handler to catch SIGCHLD signals.

19408 **RETURN VALUE**

19409 Upon successful completion, *grantpt()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 19410 indicate the error.

19411 **ERRORS**19412 The *grantpt()* function may fail if:

|       |          |                                                                                    |  |
|-------|----------|------------------------------------------------------------------------------------|--|
| 19413 | [EBADF]  | The <i>fildes</i> argument is not a valid open file descriptor.                    |  |
| 19414 | [EINVAL] | The <i>fildes</i> argument is not associated with a master pseudo-terminal device. |  |
| 19415 | [EACCES] | The corresponding slave pseudo-terminal device could not be accessed.              |  |

19416 **EXAMPLES**

19417 None.

19418 **APPLICATION USAGE**

19419 None.

19420 **RATIONALE**

19421 None.

19422 **FUTURE DIRECTIONS**

19423 None.

19424 **SEE ALSO**19425 *open()*, *ptsname()*, *unlockpt()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<stdlib.h>` |19426 **CHANGE HISTORY**

19427 First released in Issue 4, Version 2.

19428 **Issue 5**

19429 Moved from X/OPEN UNIX extension to BASE.

19430 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section in  
 19431 previous issues.



19432 **NAME**

19433 h\_errno — error return value for network database operations (**LEGACY**)

19434 **SYNOPSIS**

19435 #include <netdb.h>

19436 **DESCRIPTION**

19437 Note that this method of returning errors is used only in connection with legacy functions.

19438 The <**netdb.h**> header provides a declaration of *h\_errno* as a modifiable *l*-value of type **int**.

19439 It is unspecified whether *h\_errno* is a macro or an identifier declared with external linkage. If a  
19440 macro definition is suppressed in order to access an actual object, or a program defines an  
19441 identifier with the name *h\_errno*, the behavior is undefined.

19442 **RETURN VALUE**

19443 None.

19444 **ERRORS**

19445 No errors are defined.

19446 **EXAMPLES**

19447 None.

19448 **APPLICATION USAGE**

19449 Applications should obtain the definition of *h\_errno* by the inclusion of the <**netdb.h**> header.

19450 The practice of defining *h\_errno* in a program as an **extern int h\_errno** is obsolescent.

19451 **RATIONALE**

19452 None.

19453 **FUTURE DIRECTIONS**

19454 *h\_errno* may be withdrawn in a future version.

19455 **SEE ALSO**

19456 *endhostent()*, *errno*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**netdb.h**>

19457 **CHANGE HISTORY**

19458 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19459 **NAME**

19460 hcreate, hdestroy, hsearch — manage hash search table

19461 **SYNOPSIS**

```
19462 xSI #include <search.h>
19463 int hcreate(size_t nel);
19464 void hdestroy(void);
19465 ENTRY *hsearch(ENTRY item, ACTION action);
19466
```

19467 **DESCRIPTION**19468 The *hcreate()*, *hdestroy()*, and *hsearch()* functions manage hash search tables.

19469 The *hcreate()* function allocates sufficient space for the table, and the application shall ensure it is called before *hsearch()* is used. The *nel* argument is an estimate of the maximum number of entries that the table shall contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

19473 The *hdestroy()* function disposes of the search table, and may be followed by another call to *hcreate()*. After the call to *hdestroy()*, the data can no longer be considered accessible.

19475 The *hsearch()* function is a hash-table search routine. It shall return a pointer into a hash table indicating the location at which an entry can be found. The *item* argument is a structure of type **ENTRY** (defined in the *<search.h>* header) containing two pointers: *item.key* points to the comparison key (a **char\***), and *item.data* (a **void\***) points to any other data to be associated with that key. The comparison function used by *hsearch()* is *strcmp()*. The *action* argument is a member of an enumeration type **ACTION** indicating the disposition of the entry if it cannot be found in the table. **ENTER** indicates that the item should be inserted in the table at an appropriate point. **FIND** indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a null pointer.

19484 These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

19486 **RETURN VALUE**

19487 The *hcreate()* function shall return 0 if it cannot allocate sufficient space for the table; otherwise, it shall return non-zero.

19489 The *hdestroy()* function shall return no value.

19490 The *hsearch()* function shall return a null pointer if either the action is **FIND** and the item could not be found or the action is **ENTER** and the table is full.

19492 **ERRORS**

19493 The *hcreate()* and *hsearch()* functions may fail if:

19494 [ENOMEM] Insufficient storage space is available.

19495 **EXAMPLES**

19496 The following example reads in strings followed by two numbers and stores them in a hash table, discarding duplicates. It then reads in strings and finds the matching entry in the hash table and prints it out.

```
19499 #include <stdio.h>
19500 #include <search.h>
19501 #include <string.h>
19502 struct info { /* This is the info stored in the table */
19503 int age, room; /* other than the key. */
```

```

19504 };
19505 #define NUM_EMPL 5000 /* # of elements in search table. */
19506 int main(void)
19507 {
19508 char string_space[NUM_EMPL*20]; /* Space to store strings. */
19509 struct info info_space[NUM_EMPL]; /* Space to store employee info. */
19510 char *str_ptr = string_space; /* Next space in string_space. */
19511 struct info *info_ptr = info_space;
19512 /* Next space in info_space. */
19513 ENTRY item;
19514 ENTRY *found_item; /* Name to look for in table. */
19515 char name_to_find[30];
19516
19516 int i = 0;
19517
19517 /* Create table; no error checking is performed. */
19518 (void) hcreate(NUM_EMPL);
19519 while (scanf("%s%d%d", str_ptr, &info_ptr->age,
19520 &info_ptr->room) != EOF && i++ < NUM_EMPL) {
19521
19521 /* Put information in structure, and structure in item. */
19522 item.key = str_ptr;
19523 item.data = info_ptr;
19524 str_ptr += strlen(str_ptr) + 1;
19525 info_ptr++;
19526
19526 /* Put item into table. */
19527 (void) hsearch(item, ENTER);
19528 }
19529
19529 /* Access table. */
19530 item.key = name_to_find;
19531 while (scanf("%s", item.key) != EOF) {
19532 if ((found_item = hsearch(item, FIND)) != NULL) {
19533
19533 /* If item is in the table. */
19534 (void)printf("found %s, age = %d, room = %d\n",
19535 found_item->key,
19536 ((struct info *)found_item->data)->age,
19537 ((struct info *)found_item->data)->room);
19538 } else
19539 (void)printf("no such employee %s\n", name_to_find);
19540 }
19541 return 0;
19542 }

```

#### 19543 APPLICATION USAGE

19544 The *hcreate()* and *hsearch()* functions may use *malloc()* to allocate space.

#### 19545 RATIONALE

19546 None.

19547 **FUTURE DIRECTIONS**

19548           None.

19549 **SEE ALSO**

19550           *bsearch()*, *lsearch()*, *malloc()*, *strcmp()*, *tsearch()*, the Base Definitions volume of  
19551           IEEE Std. 1003.1-200x, <**search.h**>

19552 **CHANGE HISTORY**

19553           First released in Issue 1. Derived from Issue 1 of the SVID.

19554 **Issue 4**

19555           In the SYNOPSIS section, the type of argument *nel* in the declaration of *hcreate()* is changed from  
19556           **unsigned** to **size\_t**, and the argument list is explicitly defined as **void** in the declaration of  
19557           *hdestroy()*.

19558           In the DESCRIPTION, the type of the comparison key is explicitly defined as **char\***, the type of  
19559           *item.data* is explicitly defined as **void\***, and a statement is added indicating that *hsearch()* uses  
19560           *strcmp()* as the comparison function.

19561           In the EXAMPLES section, the sample code is updated to use ISO C standard syntax.

19562           An ERRORS section is added and [ENOMEM] is defined as an error that may be returned by  
19563           *hsearch()* and *hcreate()*.

19564 **Issue 6**

19565           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

19566 **NAME**

19567 htonl, htons, ntohl, ntohs — convert values between host and network byte order

19568 **SYNOPSIS**

```
19569 #include <arpa/inet.h>

19570 uint32_t htonl(uint32_t hostlong);
19571 uint16_t htons(uint16_t hostshort);
19572 uint32_t ntohl(uint32_t netlong);
19573 uint16_t ntohs(uint16_t netshort);
```

19574 **DESCRIPTION**

19575 These functions shall convert 16-bit and 32-bit quantities between network byte order and host  
19576 byte order.

19577 On some implementations, these functions are defined as macros.

19578 The **uint32\_t** and **uint16\_t** types shall be defined as described in **<inttypes.h>**.

19579 **RETURN VALUE**

19580 The *htonl()* and *htons()* functions shall return the argument value converted from host to  
19581 network byte order.

19582 The *ntohl()* and *ntohs()* functions shall return the argument value converted from network to  
19583 host byte order.

19584 **ERRORS**

19585 No errors are defined.

19586 **EXAMPLES**

19587 None.

19588 **APPLICATION USAGE**

19589 These functions are most often used in conjunction with IPv4 addresses and ports as returned by  
19590 *gethostent()* and *getservent()*.

19591 **RATIONALE**

19592 None.

19593 **FUTURE DIRECTIONS**

19594 None.

19595 **SEE ALSO**

19596 *endhostent()*, *endservent()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<inttypes.h>**,  
19597 **<arpa/inet.h>**

19598 **CHANGE HISTORY**

19599 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19600 **NAME**

19601           htons — convert values between host and network byte order

19602 **SYNOPSIS**

19603           #include <arpa/inet.h>

19604           uint16\_t htons(uint16\_t *hostshort*);

19605 **DESCRIPTION**

19606           Refer to *htonl()*.

19607 **NAME**

19608 hypot, hypotf, hypotl — Euclidean distance function

19609 **SYNOPSIS**

```
19610 xSI #include <math.h>
19611 double hypot(double x, double y);
19612 float hypotf(float x, float y);
19613 long double hypotl(long double x, long double y);
19614
```

19615 **DESCRIPTION**19616 These functions shall compute the value of the square root of  $x^2+y^2$ .

19617 An application wishing to check for error situations should set *errno* to 0 before calling *hypot()*.  
 19618 If *errno* is non-zero on return, or the return value is HUGE\_VAL or NaN, an error has occurred.

19619 **RETURN VALUE**19620 Upon successful completion, these functions shall return the length of the hypotenuse of a  
19621 right-angled triangle with sides of length *x* and *y*.19622 If the result would cause overflow, HUGE\_VAL shall be returned and *errno* may be set to  
19623 [ERANGE].19624 If *x* or *y* is NaN, NaN shall be returned. and *errno* may be set to [EDOM].19625 If the correct result would cause underflow, 0 shall be returned and *errno* may be set to  
19626 [ERANGE].19627 **ERRORS**

19628 These functions may fail if:

19629 [EDOM] The value of *x* or *y* is NaN.

19630 [ERANGE] The result overflows or underflows.

19631 No other errors shall occur.

19632 **EXAMPLES**

19633 None.

19634 **APPLICATION USAGE**

19635 The *hypot()* function takes precautions against overflow during intermediate steps of the  
 19636 computation. If the calculated result would still overflow a double, then *hypot()* shall return  
 19637 HUGE\_VAL.

19638 **RATIONALE**

19639 None.

19640 **FUTURE DIRECTIONS**

19641 None.

19642 **SEE ALSO**19643 *isnan()*, *sqrt()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>19644 **CHANGE HISTORY**

19645 First released in Issue 1. Derived from Issue 1 of the SVID.

19646 **Issue 4**

19647       References to *matherr()* are removed.

19648       The RETURN VALUE and ERRORS sections are substantially rewritten to rationalize error  
19649       handling in the mathematics functions.

19650 **Issue 5**

19651       The DESCRIPTION is updated to indicate how an application should check for an error. This  
19652       text was previously published in the APPLICATION USAGE section.

19653 **Issue 6**

19654       The *hypotf()* and *hypotl()* functions are added for alignment with the ISO/IEC 9899:1999  
19655       standard.



19656 **NAME**

19657 iconv — codeset conversion function

19658 **SYNOPSIS**19659 XSI 

```
#include <iconv.h>
```

```
19660 size_t iconv(iconv_t cd, char **restrict inbuf,
19661 size_t *restrict inbytesleft, char **restrict outbuf,
19662 size_t *restrict outbytesleft);
```

19663

19664 **DESCRIPTION**

19665 The *iconv()* function shall convert the sequence of characters from one codeset, in the array  
 19666 specified by *inbuf*, into a sequence of corresponding characters in another codeset, in the array  
 19667 specified by *outbuf*. The codesets are those specified in the *iconv\_open()* call that returned the  
 19668 conversion descriptor, *cd*. The *inbuf* argument points to a variable that points to the first  
 19669 character in the input buffer and *inbytesleft* indicates the number of bytes to the end of the buffer  
 19670 to be converted. The *outbuf* argument points to a variable that points to the first available byte in  
 19671 the output buffer and *outbytesleft* indicates the number of the available bytes to the end of the  
 19672 buffer.

19673 For state-dependent encodings, the conversion descriptor *cd* is placed into its initial shift state by  
 19674 a call for which *inbuf* is a null pointer, or for which *inbuf* points to a null pointer. When *iconv()*  
 19675 is called in this way, and if *outbuf* is not a null pointer or a pointer to a null pointer, and *outbytesleft*  
 19676 points to a positive value, *iconv()* shall place, into the output buffer, the byte sequence to change  
 19677 the output buffer to its initial shift state. If the output buffer is not large enough to hold the  
 19678 entire reset sequence, *iconv()* shall fail and set *errno* to [E2BIG]. Subsequent calls with *inbuf* as  
 19679 other than a null pointer or a pointer to a null pointer cause the conversion to take place from  
 19680 the current state of the conversion descriptor.

19681 If a sequence of input bytes does not form a valid character in the specified codeset, conversion  
 19682 stops after the previous successfully converted character. If the input buffer ends with an  
 19683 incomplete character or shift sequence, conversion stops after the previous successfully  
 19684 converted bytes. If the output buffer is not large enough to hold the entire converted input,  
 19685 conversion stops just prior to the input bytes that would cause the output buffer to overflow.  
 19686 The variable pointed to by *inbuf* is updated to point to the byte following the last byte  
 19687 successfully used in the conversion. The value pointed to by *inbytesleft* is decremented to reflect  
 19688 the number of bytes still not converted in the input buffer. The variable pointed to by *outbuf* is  
 19689 updated to point to the byte following the last byte of converted output data. The value pointed  
 19690 to by *outbytesleft* is decremented to reflect the number of bytes still available in the output buffer.  
 19691 For state-dependent encodings, the conversion descriptor is updated to reflect the shift state in  
 19692 effect at the end of the last successfully converted byte sequence.

19693 If *iconv()* encounters a character in the input buffer that is valid, but for which an identical  
 19694 character does not exist in the target codeset, *iconv()* performs an implementation-defined  
 19695 conversion on this character.

19696 **RETURN VALUE**

19697 The *iconv()* function shall update the variables pointed to by the arguments to reflect the extent  
 19698 of the conversion and return the number of non-identical conversions performed. If the entire  
 19699 string in the input buffer is converted, the value pointed to by *inbytesleft* shall be 0. If the input  
 19700 conversion is stopped due to any conditions mentioned above, the value pointed to by *inbytesleft*  
 19701 shall be non-zero and *errno* shall be set to indicate the condition. If an error occurs *iconv()* shall  
 19702 return (*size\_t*)-1 and set *errno* to indicate the error.

19703 **ERRORS**

19704 The *iconv()* function shall fail if:

19705 [EILSEQ] Input conversion stopped due to an input byte that does not belong to the  
19706 input codeset.

19707 [E2BIG] Input conversion stopped due to lack of space in the output buffer.

19708 [EINVAL] Input conversion stopped due to an incomplete character or shift sequence at  
19709 the end of the input buffer.

19710 The *iconv()* function may fail if:

19711 [EBADF] The *cd* argument is not a valid open conversion descriptor.

19712 **EXAMPLES**

19713 None.

19714 **APPLICATION USAGE**

19715 The *inbuf* argument indirectly points to the memory area which contains the conversion input  
19716 data. The *outbuf* argument indirectly points to the memory area which is to contain the result of  
19717 the conversion. The objects indirectly pointed to by *inbuf* and *outbuf* are not restricted to  
19718 containing data that is directly representable in the ISO C standard language **char** data type. The  
19719 type of *inbuf* and *outbuf*, **char\*\***, does not imply that the objects pointed to are interpreted as  
19720 null-terminated C strings or arrays of characters. Any interpretation of a byte sequence that  
19721 represents a character in a given character set encoding scheme is done internally within the  
19722 codeset converters. For example, the area pointed to indirectly by *inbuf* and/or *outbuf* can  
19723 contain all zero octets that are not interpreted as string terminators but as coded character data  
19724 according to the respective codeset encoding scheme. The type of the data (**char**, **short**, **long**, and  
19725 so on) read or stored in the objects is not specified, but may be inferred for both the input and  
19726 output data by the converters determined by the *fromcode* and *toctype* arguments of *iconv\_open()*.

19727 Regardless of the data type inferred by the converter, the size of the remaining space in both  
19728 input and output objects (the *inbytesleft* and *outbytesleft* arguments) is always measured in bytes.

19729 For implementations that support the conversion of state-dependent encodings, the conversion  
19730 descriptor must be able to accurately reflect the shift-state in effect at the end of the last  
19731 successful conversion. It is not required that the conversion descriptor itself be updated, which  
19732 would require it to be a pointer type. Thus, implementations are free to implement the  
19733 descriptor as a handle (other than a pointer type) by which the conversion information can be  
19734 accessed and updated.

19735 **RATIONALE**

19736 None.

19737 **FUTURE DIRECTIONS**

19738 None.

19739 **SEE ALSO**

19740 *iconv\_open()*, *iconv\_close()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**iconv.h**>

19741 **CHANGE HISTORY**

19742 First released in Issue 4. Derived from the HP-UX Manual.

19743 **Issue 6**

19744 The SYNOPSIS has been corrected to align with the <**iconv.h**> reference page.

19745 The **restrict** keyword is added to the *iconv()* prototype for alignment with the  
19746 ISO/IEC 9899:1999 standard.

19747 **NAME**

19748 iconv\_close — codeset conversion deallocation function

19749 **SYNOPSIS**

19750 XSI #include &lt;iconv.h&gt;

19751 int iconv\_close(iconv\_t *cd*);

19752

19753 **DESCRIPTION**19754 The *iconv\_close()* function deallocates the conversion descriptor *cd* and all other associated  
19755 resources allocated by *iconv\_open()*.19756 If a file descriptor is used to implement the type **iconv\_t**, that file descriptor is closed.19757 **RETURN VALUE**19758 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
19759 indicate the error.19760 **ERRORS**19761 The *iconv\_close()* function may fail if:

19762 [EBADF] The conversion descriptor is invalid.

19763 **EXAMPLES**

19764 None.

19765 **APPLICATION USAGE**

19766 None.

19767 **RATIONALE**

19768 None.

19769 **FUTURE DIRECTIONS**

19770 None.

19771 **SEE ALSO**19772 *iconv()*, *iconv\_open()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**iconv.h**>19773 **CHANGE HISTORY**

19774 First released in Issue 4. Derived from the HP-UX Manual.

19775 **NAME**

19776 iconv\_open — codeset conversion allocation function

19777 **SYNOPSIS**19778 XSI `#include <iconv.h>`19779 `iconv_t iconv_open(const char *tocode, const char *fromcode);`

19780

19781 **DESCRIPTION**

19782 The *iconv\_open()* function shall return a conversion descriptor that describes a conversion from  
 19783 the codeset specified by the string pointed to by the *fromcode* argument to the codeset specified  
 19784 by the string pointed to by the *tocode* argument. For state-dependent encodings, the conversion  
 19785 descriptor is in a codeset-dependent initial shift state, ready for immediate use with *iconv()*.

19786 Settings of *fromcode* and *tocode* and their permitted combinations are implementation-defined.19787 A conversion descriptor remains valid until it is closed by *iconv\_close()* or an implicit close.

19788 If a file descriptor is used to implement conversion descriptors, the FD\_CLOEXEC flag shall be  
 19789 set; see <fcntl.h>.

19790 **RETURN VALUE**

19791 Upon successful completion, *iconv\_open()* shall return a conversion descriptor for use on  
 19792 subsequent calls to *iconv()*. Otherwise, *iconv\_open()* shall return (**iconv\_t**)-1 and set *errno* to  
 19793 indicate the error.

19794 **ERRORS**19795 The *iconv\_open()* function may fail if:

19796 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

19797 [ENFILE] Too many files are currently open in the system.

19798 [ENOMEM] Insufficient storage space is available.

19799 [EINVAL] The conversion specified by *fromcode* and *tocode* is not supported by the  
 19800 implementation.

19801 **EXAMPLES**

19802 None.

19803 **APPLICATION USAGE**

19804 Some implementations of *iconv\_open()* use *malloc()* to allocate space for internal buffer areas.  
 19805 The *iconv\_open()* function may fail if there is insufficient storage space to accommodate these  
 19806 buffers.

19807 Portable applications must assume that conversion descriptors are not valid after a call to one of  
 19808 the *exec* functions.

19809 **RATIONALE**

19810 None.

19811 **FUTURE DIRECTIONS**

19812 None.

19813 **SEE ALSO**19814 *iconv()*, *iconv\_close()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <fcntl.h>, <iconv.h>

19815 **CHANGE HISTORY**

19816 First released in Issue 4. Derived from the HP-UX Manual.

19817 **NAME**

19818 if\_freenameindex — free memory allocated by *ifnameindex()*

19819 **SYNOPSIS**

19820 #include <net/if.h>

19821 void if\_freenameindex(struct if\_nameindex \*ptr);

19822 **DESCRIPTION**

19823 The *if\_freenameindex()* function shall free the memory allocated by *if\_nameindex*. The *ptr* |  
19824 argument shall be a pointer that was returned by *if\_nameindex*. After *if\_freenameindex()* has been |  
19825 called, the application should not use the array of which *ptr* is the address.

19826 **RETURN VALUE**

19827 None.

19828 **ERRORS**

19829 No errors are defined.

19830 **EXAMPLES**

19831 None.

19832 **APPLICATION USAGE**

19833 None.

19834 **RATIONALE**

19835 None.

19836 **FUTURE DIRECTIONS**

19837 None.

19838 **SEE ALSO**

19839 *getsockopt()*, *if\_indextoname()*, *if\_nameindex()*, *if\_nametoindex()*, *setsockopt()*, the Base Definitions |  
19840 volume of IEEE Std. 1003.1-200x, <net/if.h>

19841 **CHANGE HISTORY**

19842 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19843 **NAME**

19844 if\_indextoname — map a network interface index to its corresponding name

19845 **SYNOPSIS**

19846 #include <net/if.h>

19847 char \*if\_indextoname(unsigned *ifindex*, char \**ifname*);

19848 **DESCRIPTION**

19849 The *if\_indextoname()* function shall map an interface index to its corresponding name.

19850 When this function is called, *ifname* shall point to a buffer of at least {IFNAMSIZ} bytes. The  
19851 function shall place in this buffer the name of the interface with index *ifindex*.

19852 **RETURN VALUE**

19853 If *ifindex* is an interface index, then the function shall return the value supplied in *ifname*, which  
19854 points to a buffer now containing the interface name. Otherwise, the function shall return a  
19855 NULL pointer and set *errno* to indicate the error.

19856 **ERRORS**

19857 The *if\_indextoname()* function shall fail if:

19858 [ENXIO] The interface does not exist.

19859 **EXAMPLES**

19860 None.

19861 **APPLICATION USAGE**

19862 None.

19863 **RATIONALE**

19864 None.

19865 **FUTURE DIRECTIONS**

19866 None.

19867 **SEE ALSO**

19868 *getsockopt()*, *if\_freenameindex()*, *if\_nameindex()*, *if\_nametoindex()*, *setsockopt()*, the Base  
19869 Definitions volume of IEEE Std. 1003.1-200x, <net/if.h>

19870 **CHANGE HISTORY**

19871 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19872 **NAME**

19873 if\_nameindex — return all network interface names and indexes

19874 **SYNOPSIS**

19875 #include <net/if.h>

19876 struct if\_nameindex \*if\_nameindex(void);

19877 **DESCRIPTION**

19878 The *if\_nameindex()* function shall return an array of *if\_nameindex* structures, one structure per  
19879 interface. The end of the array is indicated by a structure with an *if\_index* field of zero and an  
19880 *if\_name* field of NULL.

19881 Applications should call *if\_freenameindex()* to release the memory that may be dynamically  
19882 allocated by this function, after they have finished using it.

19883 **RETURN VALUE**

19884 Array of structures identifying local interfaces. A NULL pointer is returned upon an error, with  
19885 *errno* set to indicate the error.

19886 **ERRORS**

19887 The *if\_nameindex()* function may fail if:

19888 [ENOBUFS] Insufficient resources are available to complete the function.

19889 **EXAMPLES**

19890 None.

19891 **APPLICATION USAGE**

19892 None.

19893 **RATIONALE**

19894 None.

19895 **FUTURE DIRECTIONS**

19896 None.

19897 **SEE ALSO**

19898 *getsockopt()*, *if\_freenameindex()*, *if\_indextoname()*, *if\_nametoindex()*, *setsockopt()*, the Base  
19899 Definitions volume of IEEE Std. 1003.1-200x, <net/if.h>

19900 **CHANGE HISTORY**

19901 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.



19902 **NAME**

19903 if\_nametoindex — map a network interface name to its corresponding index |

19904 **SYNOPSIS**

19905 #include <net/if.h>

19906 unsigned if\_nametoindex(const char \*ifname); |

19907 **DESCRIPTION**

19908 The *if\_nametoindex()* function shall return the interface index corresponding to name *ifname*. |

19909 **RETURN VALUE**

19910 The corresponding index if *ifname* is the name of an interface; otherwise, zero. |

19911 **ERRORS**

19912 No errors are defined. |

19913 **EXAMPLES**

19914 None.

19915 **APPLICATION USAGE**

19916 None.

19917 **RATIONALE**

19918 None.

19919 **FUTURE DIRECTIONS**

19920 None.

19921 **SEE ALSO**

19922 *getsockopt()*, *if\_freenameindex()*, *if\_indextoname()*, *if\_nameindex()*, *setsockopt()*, the Base |

19923 Definitions volume of IEEE Std. 1003.1-200x, <net/if.h> |

19924 **CHANGE HISTORY**

19925 First released in Issue 6. Derived from the XNS, Issue 5.2 specification. |

19926 **NAME**

19927           ilogb, ilogbf, ilogbl — return an unbiased exponent

19928 **SYNOPSIS**

19929           #include <math.h>

19930           int ilogb(double x);

19931           int ilogbf(float x);

19932           int ilogbl(long double x);

19933 **DESCRIPTION**

19934           These functions shall return the exponent part of  $x$ . Formally, the return value is the integral part of  $\log_r |x|$  as a signed integral value, for non-zero  $x$ , where  $r$  is the radix of the machine's floating point arithmetic, which is the value of FLT\_RADIX defined in <float.h>.

19937 **RETURN VALUE**

19938           Upon successful completion, these functions shall return the exponent part of  $x$  as a signed integer value.

19940           If  $x$  is zero, these functions shall return the value FP\_ILOGB0; if  $x$  is infinite, they shall return the value {INT\_MAX}; if  $x$  is a NaN, they shall return the value FP\_ILOGBNAN; otherwise, they shall be equivalent to calling the corresponding *logb()* function and casting the returned value to type **int**.

19944 **ERRORS**

19945           These functions may fail if:

19946           [ERANGE]       The value of  $x$  is 0.

19947 **EXAMPLES**

19948           None.

19949 **APPLICATION USAGE**

19950           None.

19951 **RATIONALE**

19952           None.

19953 **FUTURE DIRECTIONS**

19954           None.

19955 **SEE ALSO**

19956           *logb()*, *scalb()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <float.h>, <math.h>

19957 **CHANGE HISTORY**

19958           First released in Issue 4, Version 2.

19959 **Issue 5**

19960           Moved from X/OPEN UNIX extension to BASE.

19961 **Issue 6**

19962           The *ilogbf()* and *ilogbl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

19964 **NAME**

19965           imaxabs — return absolute value

19966 **SYNOPSIS**

19967           #include &lt;inttypes.h&gt;

19968           intmax\_t imaxabs(intmax\_t j);

19969 **DESCRIPTION**

19970 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
19971       conflict between the requirements described here and the ISO C standard is unintentional. This  
19972       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

19973       The *imaxabs()* function shall compute the absolute value of an integer *j*. If the result cannot be  
19974       represented, the behavior is undefined.

19975 **RETURN VALUE**19976       The *imaxabs()* function shall return the absolute value.19977 **ERRORS**

19978       No errors are defined.

19979 **EXAMPLES**

19980       None.

19981 **APPLICATION USAGE**

19982       The absolute value of the most negative number cannot be represented in two's complement.

19983 **RATIONALE**

19984       None.

19985 **FUTURE DIRECTIONS**

19986       None.

19987 **SEE ALSO**19988       *imaxdiv()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <inttypes.h>19989 **CHANGE HISTORY**

19990       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

19991 **NAME**

19992           imaxdiv — return quotient and remainder

19993 **SYNOPSIS**

19994           #include <inttypes.h>

19995           imaxdiv\_t imaxdiv(intmax\_t numer, intmax\_t denom);

19996 **DESCRIPTION**

19997 **cx**       The functionality described on this reference page is aligned with the ISO C standard. Any  
19998           conflict between the requirements described here and the ISO C standard is unintentional. This  
19999           volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

20000           The *imaxdiv()* function shall compute *numer / denom* and *numer % denom* in a single operation.

20001 **RETURN VALUE**

20002           The *imaxdiv()* function shall return a structure of type **imaxdiv\_t**, comprising both the quotient  
20003           and the remainder. The structure shall contain (in either order) the members *quot* (the quotient)  
20004           and *rem* (the remainder), each of which has type **intmax\_t**.

20005           If either part of the result cannot be represented, the behavior is undefined.

20006 **ERRORS**

20007           No errors are defined.

20008 **EXAMPLES**

20009           None.

20010 **APPLICATION USAGE**

20011           None.

20012 **RATIONALE**

20013           None.

20014 **FUTURE DIRECTIONS**

20015           None.

20016 **SEE ALSO**

20017           *imaxabs()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**inttypes.h**>

20018 **CHANGE HISTORY**

20019           First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20020 **NAME**

20021 index — character string operations (**LEGACY**)

20022 **SYNOPSIS**

```
20023 xsi #include <strings.h>
```

```
20024 char *index(const char *s, int c);
```

20025

20026 **DESCRIPTION**

20027 The *index()* function is identical to *strchr()*.

20028 **RETURN VALUE**

20029 See *strchr()*.

20030 **ERRORS**

20031 See *strchr()*.

20032 **EXAMPLES**

20033 None.

20034 **APPLICATION USAGE**

20035 *strchr()* is preferred over this function.

20036 For maximum portability, it is recommended to replace the function call to *index()* as follows:

```
20037 #define index(a,b) strchr((a),(b))
```

20038 **RATIONALE**

20039 None.

20040 **FUTURE DIRECTIONS**

20041 This function may be withdrawn in a future version.

20042 **SEE ALSO**

20043 *strchr()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**strings.h**>

20044 **CHANGE HISTORY**

20045 First released in Issue 4, Version 2.

20046 **Issue 5**

20047 Moved from X/OPEN UNIX extension to BASE.

20048 **Issue 6**

20049 This function is marked LEGACY.

## 20050 NAME

20051 inet\_addr, inet\_lnaof (LEGACY), inet\_makeaddr (LEGACY), inet\_netof (LEGACY),  
 20052 inet\_network (LEGACY), inet\_ntoa — IPv4 address manipulation

## 20053 SYNOPSIS

```
20054 #include <arpa/inet.h>

20055 in_addr_t inet_addr(const char *cp);
20056 in_addr_t inet_lnaof(struct in_addr in);
20057 struct in_addr inet_makeaddr(in_addr_t net, in_addr_t lna);
20058 in_addr_t inet_netof(struct in_addr in);
20059 in_addr_t inet_network(const char *cp);
20060 char *inet_ntoa(struct in_addr in);
```

## 20061 DESCRIPTION

20062 The *inet\_addr()* function shall convert the string pointed to by *cp*, in the standard IPv4 dotted  
 20063 decimal notation, to an integer value suitable for use as an Internet address.

20064 The *inet\_lnaof()* function shall take an Internet host address specified by *in* and extract the local  
 20065 network address part, in host byte order.

20066 The *inet\_makeaddr()* function shall take the Internet network number specified by *net* and the  
 20067 local network address specified by *lna*, both in host byte order, and construct an Internet address  
 20068 from them.

20069 The *inet\_netof()* function shall take an Internet host address specified by *in* and extract the  
 20070 network number part, in host byte order.

20071 The *inet\_network()* function shall convert the string pointed to by *cp*, in the standard IPv4 dotted  
 20072 decimal notation, to an integer value suitable for use as an Internet network number.

20073 The *inet\_ntoa()* function shall convert the Internet host address specified by *in* to a string in the  
 20074 Internet standard dot notation.

20075 All Internet addresses shall be returned in network order (bytes ordered from left to right).

20076 Values specified using IPv4 dotted decimal notation take one of the following forms:

20077 a.b.c.d     When four parts are specified, each is interpreted as a byte of data and assigned,  
 20078 from left to right, to the four bytes of an Internet address.

20079 a.b.c       When a three-part address is specified, the last part is interpreted as a 16-bit  
 20080 quantity and placed in the rightmost two bytes of the network address. This makes  
 20081 the three-part address format convenient for specifying Class B network addresses  
 20082 as **128.net.host**.

20083 a.b         When a two-part address is supplied, the last part is interpreted as a 24-bit  
 20084 quantity and placed in the rightmost three bytes of the network address. This  
 20085 makes the two-part address format convenient for specifying Class A network  
 20086 addresses as **net.host**.

20087 a           When only one part is given, the value is stored directly in the network address  
 20088 without any byte rearrangement.

20089 All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or  
 20090 hexadecimal, as specified in the ISO C standard (that is, a leading "0x" or "0X" implies  
 20091 hexadecimal; otherwise, a leading '0' implies octal; otherwise, the number is interpreted as  
 20092 decimal).

**20093 RETURN VALUE**

20094 Upon successful completion, *inet\_addr()* shall return the Internet address. Otherwise, it shall  
20095 return **(in\_addr\_t)**(-1).

20096 The *inet\_lnaof()* function shall return the local network address part.

20097 The *inet\_makeaddr()* function shall return the constructed Internet address.

20098 The *inet\_netof()* function shall return the network number.

20099 Upon successful completion, *inet\_network()* shall return the converted Internet network number.  
20100 Otherwise, it shall return **(in\_addr\_t)**(-1).

20101 The *inet\_ntoa()* function shall return a pointer to the network address in Internet standard dot  
20102 notation.

**20103 ERRORS**

20104 No errors are defined.

**20105 EXAMPLES**

20106 None.

**20107 APPLICATION USAGE**

20108 The *inet\_lnaof()*, *inet\_makeaddr()*, *inet\_netof()*, and *inet\_network()* functions are marked LEGACY  
20109 and should not be used by new applications.

20110 The return value of *inet\_ntoa()* may point to static data that may be overwritten by subsequent  
20111 calls to *inet\_ntoa()*.

**20112 RATIONALE**

20113 None.

**20114 FUTURE DIRECTIONS**

20115 The *inet\_lnaof()*, *inet\_makeaddr()*, *inet\_netof()*, and *inet\_network()* functions may be withdrawn in  
20116 a future version.

**20117 SEE ALSO**

20118 *endhostent()*, *endnetent()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <arpa/inet.h>

**20119 CHANGE HISTORY**

20120 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

20121 **NAME**

20122           inet\_lnaof — IPv4 address manipulation

20123 **SYNOPSIS**

20124           #include <arpa/inet.h>

20125           in\_addr\_t inet\_lnaof(struct in\_addr *in*);

20126 **DESCRIPTION**

20127           Refer to *inet\_addr()*.



20128 **NAME**

20129       inet\_makeaddr — IPv4 address manipulation

20130 **SYNOPSIS**

20131       #include &lt;arpa/inet.h&gt;

20132       struct in\_addr inet\_makeaddr(in\_addr\_t *net*, in\_addr\_t *lna*);20133 **DESCRIPTION**20134       Refer to *inet\_addr()*.

20135 **NAME**

20136       inet\_netof — IPv4 address manipulation

20137 **SYNOPSIS**

20138       #include <arpa/inet.h>

20139       in\_addr\_t inet\_netof(struct in\_addr *in*);

20140 **DESCRIPTION**

20141       Refer to *inet\_addr()*.

20142 **NAME**

20143       inet\_network — IPv4 address manipulation

20144 **SYNOPSIS**

20145       #include &lt;arpa/inet.h&gt;

20146       in\_addr\_t inet\_network(const char \*cp);

20147 **DESCRIPTION**20148       Refer to *inet\_addr()*.

20149 **NAME**

20150       inet\_ntoa — IPv4 address manipulation

20151 **SYNOPSIS**

20152       #include <arpa/inet.h>

20153       char \*inet\_ntoa(struct in\_addr *in*);

20154 **DESCRIPTION**

20155       Refer to *inet\_addr()*.

20156 **NAME**

20157 inet\_ntop, inet\_pton — convert IPv4 and IPv6 addresses between binary and text form

20158 **SYNOPSIS**

20159 IP6 `#include <arpa/inet.h>`

```
20160 const char *inet_ntop(int af, const void *restrict src,
20161 char *restrict dst, socklen_t size);
20162 int inet_pton(int af, const char *restrict src, void *restrict dst);
20163
```

20164 **Notes to Reviewers**

20165 *This section with side shading will not appear in the final copy. - Ed.*

20166 D3, XSH, ERN 330 (AI 2000-05-024) To supply editing instructions regarding making these  
20167 functions mandatory and shading IPv6-specific information.

20168 **DESCRIPTION**

20169 The *inet\_ntop()* function converts a numeric address into a text string suitable for presentation.  
20170 The *af* argument specifies the family of the address. This can be AF\_INET or AF\_INET6. The *src*  
20171 argument points to a buffer holding an IPv4 address if the *af* argument is AF\_INET, or an IPv6  
20172 address if the *af* argument is AF\_INET6. The *dst* argument points to a buffer where the function  
20173 stores the resulting text string; it shall not be NULL. The *size* argument specifies the size of this  
20174 buffer, which shall be large enough to hold the text string (INET\_ADDRSTRLEN characters for  
20175 IPv4, INET6\_ADDRSTRLEN characters for IPv6).

20176 The *inet\_pton()* function converts an address in its standard text presentation form into its  
20177 numeric binary form. The *af* argument specifies the family of the address. The AF\_INET and  
20178 AF\_INET6 address families are supported. The *src* argument points to the string being passed in.  
20179 The *dst* argument points to a buffer into which the function stores the numeric address; this shall  
20180 be large enough to hold the numeric address (32 bits for AF\_INET, 128 bits for AF\_INET6).

20181 If the *af* argument of *inet\_pton()* is AF\_INET, the *src* string shall be in the standard IPv4 dotted-  
20182 decimal form:

20183 `ddd.ddd.ddd.ddd`

20184 where "ddd" is a one to three digit decimal number between 0 and 255 (see *inet\_addr()*). The  
20185 *inet\_pton()* function does not accept other formats (such as the octal numbers, hexadecimal  
20186 numbers, and fewer than four numbers that *inet\_addr()* accepts).

20187 If the *af* argument of *inet\_pton()* is AF\_INET6, the *src* string shall be in one of the following  
20188 standard IPv6 text forms:

- 20189 1. The preferred form is "x:x:x:x:x:x:x:x", where the 'x's are the hexadecimal values  
20190 of the eight 16-bit pieces of the address. Leading zeros in individual fields can be omitted,  
20191 but there shall be at least one numeral in every field.
- 20192 2. A string of contiguous zero fields in the preferred form can be shown as "::". The "::"  
20193 can only appear once in an address. Unspecified addresses ("0:0:0:0:0:0:0:0") may  
20194 be represented simply as "::".
- 20195 3. A third form that is sometimes more convenient when dealing with a mixed environment  
20196 of IPv4 and IPv6 nodes is "x:x:x:x:x:x:d.d.d.d", where the 'x's are the  
20197 hexadecimal values of the six high-order 16-bit pieces of the address, and the 'd's are the  
20198 decimal values of the four low-order 8-bit pieces of the address (standard IPv4  
20199 representation).

20200 **Note:** A more extensive description of the standard representations of IPv6 addresses can  
20201 be found in RFC 2373.

20202 **RETURN VALUE**

20203 The *inet\_ntop()* function shall return a pointer to the buffer containing the text string if the  
20204 conversion succeeds, and NULL otherwise, and set *errno* to indicate the error.

20205 The *inet\_pton()* function shall return 1 if the conversion succeeds, with the address pointed to by  
20206 *dst* in network byte order. It shall return 0 if the input is not a valid IPv4 dotted-decimal string or  
20207 a valid IPv6 address string, or -1 with *errno* set to [EAFNOSUPPORT] if the *af* argument is  
20208 unknown.

20209 **ERRORS**

20210 The *inet\_ntop()* and *inet\_pton()* functions shall fail if:

20211 [EAFNOSUPPORT]

20212 The *af* argument is invalid.

20213 [ENOSPC] The size of the *inet\_ntop()* result buffer is inadequate.

20214 **EXAMPLES**

20215 None.

20216 **APPLICATION USAGE**

20217 None.

20218 **RATIONALE**

20219 None.

20220 **FUTURE DIRECTIONS**

20221 None.

20222 **SEE ALSO**

20223 The Base Definitions volume of IEEE Std. 1003.1-200x, <*arpa/inet.h*>

20224 **CHANGE HISTORY**

20225 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

20226 Marked as part of the IPv6 option.

20227 The **restrict** keyword is added to the *inet\_ntop()* and *inet\_pton()* prototypes for alignment with  
20228 the ISO/IEC 9899:1999 standard.

20229 **NAME**

20230           initstate, random, setstate, srandom — pseudorandom number functions

20231 **SYNOPSIS**

```
20232 xSI #include <stdlib.h>
20233 char *initstate(unsigned seed, char *state, size_t size);
20234 long random(void);
20235 char *setstate(const char *state);
20236 void srandom(unsigned seed);
20237
```

20238 **DESCRIPTION**

20239       The *random()* function uses a non-linear additive feedback random-number generator  
 20240       employing a default state array size of 31 **long** integers to return successive pseudo-random  
 20241       numbers in the range from 0 to  $2^{31}-1$ . The period of this random-number generator is  
 20242       approximately  $16 \times (2^{31}-1)$ . The size of the state array determines the period of the random-  
 20243       number generator. Increasing the state array size increases the period.

20244       With 256 bytes of state information, the period of the random-number generator is greater than  
 20245        $2^{69}$ .

20246       Like *rand()*, *random()* produces by default a sequence of numbers that can be duplicated by  
 20247       calling *srandom()* with 1 as the seed.

20248       The *srandom()* function initializes the current state array using the value of *seed*.

20249       The *initstate()* and *setstate()* functions handle restarting and changing random-number  
 20250       generators. The *initstate()* function allows a state array, pointed to by the *state* argument, to be  
 20251       initialized for future use. The *size* argument, which specifies the size in bytes of the state array, is  
 20252       used by *initstate()* to decide what type of random-number generator to use; the larger the state  
 20253       array, the more random the numbers. Values for the amount of state information are 8, 32, 64,  
 20254       128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one of  
 20255       these values. If *initstate()* is called with  $8 \leq \text{size} < 32$ , then *random()* uses a simple linear  
 20256       congruential random number generator. The *seed* argument specifies a starting point for the  
 20257       random-number sequence and provides for restarting at the same point. The *initstate()* function  
 20258       shall return a pointer to the previous state information array.

20259       If *initstate()* has not been called, then *random()* behaves as though *initstate()* had been called  
 20260       with *seed*=1 and *size*=128.

20261       Once a state has been initialized, *setstate()* allows switching between state arrays. The array  
 20262       defined by the *state* argument is used for further random-number generation until *initstate()* is  
 20263       called or *setstate()* is called again. The *setstate()* function shall return a pointer to the previous  
 20264       state array.

20265 **RETURN VALUE**

20266       If *initstate()* is called with *size* less than 8, it shall return NULL.

20267       The *random()* function shall return the generated pseudo-random number.

20268       The *srandom()* function shall return no value.

20269       Upon successful completion, *initstate()* and *setstate()* shall return a pointer to the previous state  
 20270       array; otherwise, a null pointer shall be returned.

20271 **ERRORS**

20272 No errors are defined.

20273 **EXAMPLES**

20274 None.

20275 **APPLICATION USAGE**

20276 After initialization, a state array can be restarted at a different point in one of two ways:

- 20277 1. The *initstate()* function can be used, with the desired seed, state array, and size of the  
20278 array.
- 20279 2. The *setstate()* function, with the desired state, can be used, followed by *srandom()* with the  
20280 desired seed. The advantage of using both of these functions is that the size of the state  
20281 array does not have to be saved once it is initialized.

20282 Although some implementations of *random()* have written messages to standard error, such  
20283 implementations do not conform to this volume of IEEE Std. 1003.1-200x.

20284 Issue 5 restores the historical behavior of this function.

20285 Threaded applications should use *rand\_r()*, *erand48()*, *nrand48()*, or *jrand48()* instead of  
20286 *random()* when an independent random number sequence in multiple threads is required.20287 **RATIONALE**

20288 None.

20289 **FUTURE DIRECTIONS**

20290 None.

20291 **SEE ALSO**20292 *drand48()*, *rand()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<stdlib.h>`20293 **CHANGE HISTORY**

20294 First released in Issue 4, Version 2.

20295 **Issue 5**

20296 Moved from X/OPEN UNIX extension to BASE.

20297 In the DESCRIPTION, the phrase “values smaller than 8” is replaced with “values greater than  
20298 or equal to 8, or less than 32”, “*size*<8” is replaced with “ $8 \leq \textit{size} < 32$ ”, and a new first paragraph  
20299 is added to the RETURN VALUE section. A note is added to the APPLICATION USAGE  
20300 indicating that these changes restore the historical behavior of the function.

20301 **Issue 6**

20302 In the DESCRIPTION, duplicate text “For values greater than or equal to 8 . . .” is removed.



20303 **NAME**

20304           insque, remque — insert or remove an element in a queue

20305 **SYNOPSIS**

```
20306 xsi #include <search.h>
20307 void insque(void *element, void *pred);
20308 void remque(void *element);
20309
```

20310 **DESCRIPTION**

20311       The *insque()* and *remque()* functions manipulate queues built from doubly-linked lists. The  
 20312       queue can be either circular or linear. An application using *insque()* or *remque()* shall ensure it  
 20313       defines a structure in which the first two members of the structure are pointers to the same type  
 20314       of structure, and any further members are application-specific. The first member of the structure  
 20315       is a forward pointer to the next entry in the queue. The second member is a backward pointer to  
 20316       the previous entry in the queue. If the queue is linear, the queue is terminated with null  
 20317       pointers. The names of the structure and of the pointer members are not subject to any special  
 20318       restriction.

20319       The *insque()* function inserts the element pointed to by *element* into a queue immediately after  
 20320       the element pointed to by *pred*.

20321       The *remque()* function removes the element pointed to by *element* from a queue.

20322       If the queue is to be used as a linear list, invoking *insque(&element, NULL)*, where *element* is the  
 20323       initial element of the queue, shall initialize the forward and backward pointers of *element* to null  
 20324       pointers.

20325       If the queue is to be used as a circular list, the application shall ensure it initializes the forward  
 20326       pointer and the backward pointer of the initial element of the queue to the element's own  
 20327       address.

20328 **RETURN VALUE**20329           The *insque()* and *remque()* functions do not return a value.20330 **ERRORS**

20331           No errors are defined.

20332 **EXAMPLES**20333           **Creating a Linear Linked List**

20334           The following example creates a linear linked list.

```
20335 #include <search.h>
20336 ...
20337 struct myque element1;
20338 struct myque element2;
20339 char *data1 = "DATA1";
20340 char *data2 = "DATA2";
20341 ...
20342 element1.data = data1;
20343 element2.data = data2;
20344 insque (&element1, NULL);
20345 insque (&element2, &element1);
```

20346 **Creating a Circular Linked List**

20347 The following example creates a circular linked list.

```

20348 #include <search.h>
20349 ...
20350 struct myque element1;
20351 struct myque element2;

20352 char *data1 = "DATA1";
20353 char *data2 = "DATA2";
20354 ...
20355 element1.data = data1;
20356 element2.data = data2;

20357 element1.fwd = &element1;
20358 element1.bck = &element1;

20359 insque (&element2, &element1);

```

20360 **Removing an Element**20361 The following example removes the element pointed to by *element1*.

```

20362 #include <search.h>
20363 ...
20364 struct myque element1;
20365 ...
20366 remque (&element1);

```

20367 **APPLICATION USAGE**

20368 The historical implementations of these functions described the arguments as being of type  
 20369 **struct qelem\*** rather than as being of type **void\*** as defined here. In those implementations,  
 20370 **struct qelem** was commonly defined in **<search.h>** as:

```

20371 struct qelem {
20372 struct qelem *q_forw;
20373 struct qelem *q_back;
20374 };

```

20375 Applications using these functions, however, were never able to use this structure directly since  
 20376 it provided no room for the actual data contained in the elements. Most applications defined  
 20377 structures that contained the two pointers as the initial elements and also provided space for, or  
 20378 pointers to, the object's data. Applications that used these functions to update more than one  
 20379 type of table also had the problem of specifying two or more different structures with the same  
 20380 name, if they literally used **struct qelem** as specified.

20381 As described here, the implementations were actually expecting a structure type where the first  
 20382 two members were forward and backward pointers to structures. With C compilers that didn't  
 20383 provide function prototypes, applications used structures as specified in the DESCRIPTION  
 20384 above and the compiler did what the application expected.

20385 If this method had been carried forward with an ISO C standard compiler and the historical  
 20386 function prototype, most applications would have to be modified to cast pointers to the  
 20387 structures actually used to be pointers to **struct qelem** to avoid compilation warnings. By  
 20388 specifying **void\*** as the argument type, applications do not need to change (unless they  
 20389 specifically referenced **struct qelem** and depended on it being defined in **<search.h>**).

20390 **RATIONALE**

20391           None.

20392 **FUTURE DIRECTIONS**

20393           None.

20394 **SEE ALSO**

20395           The Base Definitions volume of IEEE Std. 1003.1-200x, <search.h>

20396 **CHANGE HISTORY**

20397           First released in Issue 4, Version 2.

20398 **Issue 5**

20399           Moved from X/OPEN UNIX extension to BASE.

20400 **Issue 6**

20401           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

## 20402 NAME

20403        ioctl — control a STREAMS device (**STREAMS**)

## 20404 SYNOPSIS

20405 XSR        #include &lt;stropts.h&gt;

20406        int ioctl(int *fildev*, int *request*, ... /\* *arg* \*/);

20407

## 20408 DESCRIPTION

20409        The *ioctl()* function performs a variety of control functions on STREAMS devices. For non-  
 20410 STREAMS devices, the functions performed by this call are unspecified. The *request* argument  
 20411 and an optional third argument (with varying type) are passed to and interpreted by the  
 20412 appropriate part of the STREAM associated with *fildev*.

20413        The *fildev* argument is an open file descriptor that refers to a device.

20414        The *request* argument selects the control function to be performed and shall depend on the  
 20415 STREAMS device being addressed.

20416        The *arg* argument represents additional information that is needed by this specific STREAMS  
 20417 device to perform the requested function. The type of *arg* depends upon the particular control  
 20418 request, but it is either an integer or a pointer to a device-specific data structure.

20419        The *ioctl()* commands applicable to STREAMS, their arguments, and error conditions that apply  
 20420 to each individual command are described below.

20421        The following *ioctl()* commands, with error values indicated, are applicable to all STREAMS  
 20422 files:

20423        I\_PUSH        Pushes the module whose name is pointed to by *arg* onto the top of the  
 20424 current STREAM, just below the STREAM head. It then calls the *open()*  
 20425 function of the newly-pushed module.

20426        The *ioctl()* function with the I\_PUSH command shall fail if:

20427        [EINVAL]        Invalid module name. |

20428        [ENXIO]        Open function of new module failed. |

20429        [ENXIO]        Hangup received on *fildev*. |

20430        I\_POP        Removes the module just below the STREAM head of the STREAM pointed to  
 20431 by *fildev*. The *arg* argument should be 0 in an I\_POP request.

20432        The *ioctl()* function with the I\_POP command shall fail if:

20433        [EINVAL]        No module present in the STREAM. |

20434        [ENXIO]        Hangup received on *fildev*. |

20435        I\_LOOK        Retrieves the name of the module just below the STREAM head of the  
 20436 STREAM pointed to by *fildev*, and places it in a character string pointed to by  
 20437 *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long,  
 20438 where FMNAMESZ is defined in <stropts.h>.

20439        The *ioctl()* function with the I\_LOOK command shall fail if:

20440        [EINVAL]        No module present in the STREAM. |

20441        I\_FLUSH        This request flushes read and/or write queues, depending on the value of *arg*.  
 20442 Valid *arg* values are:

|       |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------|----------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 20443 | FLUSHR                                                                           | Flush all read queues.                                                                                                                                                                                                                                                                                                                                                                                                                |
| 20444 | FLUSHW                                                                           | Flush all write queues.                                                                                                                                                                                                                                                                                                                                                                                                               |
| 20445 | FLUSHRW                                                                          | Flush all read and all write queues.                                                                                                                                                                                                                                                                                                                                                                                                  |
| 20446 | The <i>ioctl()</i> function with the <code>L_FLUSH</code> command shall fail if: |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20447 | [EINVAL]                                                                         | Invalid <i>arg</i> value.                                                                                                                                                                                                                                                                                                                                                                                                             |
| 20448 | [EAGAIN] or [ENOSR]                                                              | Unable to allocate buffers for flush message.                                                                                                                                                                                                                                                                                                                                                                                         |
| 20449 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20450 | [ENXIO]                                                                          | Hangup received on <i>fildev</i> .                                                                                                                                                                                                                                                                                                                                                                                                    |
| 20451 | I_FLUSHBAND                                                                      | Flushes a particular band of messages. The <i>arg</i> argument points to a <b>bandinfo</b> structure. The <i>bi_flag</i> member may be one of FLUSHR, FLUSHW, or FLUSHRW as described above. The <i>bi_pri</i> member determines the priority band to be flushed.                                                                                                                                                                     |
| 20452 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20453 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20454 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20455 | I_SETSIG                                                                         | Requests that the STREAMS implementation send the SIGPOLL signal to the calling process when a particular event has occurred on the STREAM associated with <i>fildev</i> . I_SETSIG supports an asynchronous processing capability in STREAMS. The value of <i>arg</i> is a bitmask that specifies the events for which the process should be signaled. It is the bitwise-inclusive OR of any combination of the following constants: |
| 20456 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20457 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20458 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20459 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20460 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20461 | S_RDNORM                                                                         | A normal (priority band set to 0) message has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.                                                                                                                                                                                                                                                                     |
| 20462 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20463 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20464 | S_RDBAND                                                                         | A message with a non-zero priority band has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.                                                                                                                                                                                                                                                                       |
| 20465 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20466 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20467 | S_INPUT                                                                          | A message, other than a high-priority message, has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.                                                                                                                                                                                                                                                                |
| 20468 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20469 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20470 | S_HIPRI                                                                          | A high-priority message is present on a STREAM head read queue. A signal shall be generated even if the message is of zero length.                                                                                                                                                                                                                                                                                                    |
| 20471 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20472 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20473 | S_OUTPUT                                                                         | The write queue for normal data (priority band 0) just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) normal data downstream.                                                                                                                                                                                                                            |
| 20474 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20475 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20476 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20477 | S_WRNORM                                                                         | Same as S_OUTPUT.                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20478 | S_WRBAND                                                                         | The write queue for a non-zero priority band just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) priority data downstream.                                                                                                                                                                                                                               |
| 20479 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20480 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20481 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20482 | S_MSG                                                                            | A STREAMS signal message that contains the SIGPOLL signal has reached the front of the STREAM head read queue.                                                                                                                                                                                                                                                                                                                        |
| 20483 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20484 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20485 | S_ERROR                                                                          | Notification of an error condition has reached the STREAM head.                                                                                                                                                                                                                                                                                                                                                                       |
| 20486 |                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                       |

|       |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------|----------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 20487 |          | S_HANGUP  | Notification of a hangup has reached the STREAM head.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 20488 |          | S_BANDURG | When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.                                                                                                                                                                                                                                                                                                                  |
| 20489 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20490 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20491 |          |           | If <i>arg</i> is 0, the calling process shall be unregistered and shall not receive further SIGPOLL signals for the stream associated with <i>fildev</i> .                                                                                                                                                                                                                                                                                                               |
| 20492 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20493 |          |           | Processes that wish to receive SIGPOLL signals shall ensure that they explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same STREAM, each process shall be signaled when the event occurs.                                                                                                                                                                                              |
| 20494 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20495 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20496 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20497 |          |           | The <i>ioctl()</i> function with the I_SETSIG command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20498 |          | [EINVAL]  | The value of <i>arg</i> is invalid.                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 20499 |          | [EINVAL]  | The value of <i>arg</i> is 0 and the calling process is not registered to receive the SIGPOLL signal.                                                                                                                                                                                                                                                                                                                                                                    |
| 20500 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20501 |          | [EAGAIN]  | There were insufficient resources to store the signal request.                                                                                                                                                                                                                                                                                                                                                                                                           |
| 20502 | I_GETSIG |           | Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask in an <i>int</i> pointed to by <i>arg</i> , where the events are those specified in the description of I_SETSIG above.                                                                                                                                                                                                        |
| 20503 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20504 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20505 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20506 |          |           | The <i>ioctl()</i> function with the I_GETSIG command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20507 |          | [EINVAL]  | Process is not registered to receive the SIGPOLL signal.                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20508 | I_FIND   |           | This request compares the names of all modules currently present in the STREAM to the name pointed to by <i>arg</i> , and returns 1 if the named module is present in the STREAM, or returns 0 if the named module is not present.                                                                                                                                                                                                                                       |
| 20509 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20510 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20511 |          |           | The <i>ioctl()</i> function with the I_FIND command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20512 |          | [EINVAL]  | <i>arg</i> does not contain a valid module name.                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20513 | I_PEEK   |           | This request allows a process to retrieve the information in the first message on the STREAM head read queue without taking the message off the queue. It is analogous to <i>getmsg()</i> except that this command does not remove the message from the queue. The <i>arg</i> argument points to a <b>strpeek</b> structure.                                                                                                                                             |
| 20514 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20515 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20516 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20517 |          |           | The application shall ensure that the <i>maxlen</i> member in the <b>ctlbuf</b> and <b>databuf</b> structures is set to the number of bytes of control information and/or data information, respectively, to retrieve. The <i>flags</i> member may be marked RS_HIPRI or 0, as described by <i>getmsg()</i> . If the process sets <i>flags</i> to RS_HIPRI, for example, I_PEEK shall only look for a high-priority message on the STREAM head read queue.               |
| 20518 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20519 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20520 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20521 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20522 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20523 |          |           | I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the STREAM head read queue, or if the RS_HIPRI flag was set in <i>flags</i> and a high-priority message was not present on the STREAM head read queue. It does not wait for a message to arrive. On return, <b>ctlbuf</b> specifies information in the control buffer, <b>databuf</b> specifies information in the data buffer, and <i>flags</i> contains the value RS_HIPRI or 0. |
| 20524 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20525 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20526 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20527 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20528 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20529 | I_SRDOPT |           | Sets the read mode using the value of the argument <i>arg</i> . Read modes are described in <i>read()</i> . Valid <i>arg</i> flags are:                                                                                                                                                                                                                                                                                                                                  |
| 20530 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

|       |            |           |                                                                                               |
|-------|------------|-----------|-----------------------------------------------------------------------------------------------|
| 20531 |            | RNORM     | Byte-stream mode, the default.                                                                |
| 20532 |            | RMSGD     | Message-discard mode.                                                                         |
| 20533 |            | RMSGN     | Message-nondiscard mode.                                                                      |
| 20534 |            |           | The bitwise-inclusive OR of RMSGD and RMSGN shall return [EINVAL]. The                        |
| 20535 |            |           | bitwise-inclusive OR of RNORM and either RMSGD or RMSGN shall result in                       |
| 20536 |            |           | the other flag overriding RNORM which is the default.                                         |
| 20537 |            |           | In addition, treatment of control messages by the STREAM head may be                          |
| 20538 |            |           | changed by setting any of the following flags in <i>arg</i> :                                 |
| 20539 |            | RPROTNORM | Fail <i>read()</i> with [EBADMSG] if a message containing a                                   |
| 20540 |            |           | control part is at the front of the STREAM head read queue.                                   |
| 20541 |            | RPROTDAT  | Deliver the control part of a message as data when a                                          |
| 20542 |            |           | process issues a <i>read()</i> .                                                              |
| 20543 |            | RPROTDIS  | Discard the control part of a message, delivering any data                                    |
| 20544 |            |           | portion, when a process issues a <i>read()</i> .                                              |
| 20545 |            |           | The <i>ioctl()</i> function with the I_SRDOPT command shall fail if:                          |
| 20546 |            | [EINVAL]  | The <i>arg</i> argument is not valid.                                                         |
| 20547 | I_GRDOPT   |           | Returns the current read mode setting as, described above, in an <b>int</b> pointed to        |
| 20548 |            |           | by the argument <i>arg</i> . Read modes are described in <i>read()</i> .                      |
| 20549 | I_NREAD    |           | Counts the number of data bytes in the data part of the first message on the                  |
| 20550 |            |           | STREAM head read queue and places this value in the <b>int</b> pointed to by <i>arg</i> .     |
| 20551 |            |           | The return value for the command is the number of messages on the STREAM                      |
| 20552 |            |           | head read queue. For example, if 0 is returned in <i>arg</i> , but the <i>ioctl()</i> return  |
| 20553 |            |           | value is greater than 0, this indicates that a zero-length message is next on the             |
| 20554 |            |           | queue.                                                                                        |
| 20555 | I_FDINSERT |           | Creates a message from specified buffer(s), adds information about another                    |
| 20556 |            |           | STREAM, and sends the message downstream. The message contains a                              |
| 20557 |            |           | control part and an optional data part. The data and control parts to be sent                 |
| 20558 |            |           | are distinguished by placement in separate buffers, as described below. The                   |
| 20559 |            |           | <i>arg</i> argument points to a <b>strfdinsert</b> structure.                                 |
| 20560 |            |           | The application shall ensure that the <i>len</i> member in the <b>ctlbuf strbuf</b> structure |
| 20561 |            |           | is set to the size of a <b>t_uscalar_t</b> plus the number of bytes of control                |
| 20562 |            |           | information to be sent with the message. The <i>fildev</i> member specifies the file          |
| 20563 |            |           | descriptor of the other STREAM, and the <i>offset</i> member, which must be                   |
| 20564 |            |           | suitably aligned for use as a <b>t_uscalar_t</b> , specifies the offset from the start of     |
| 20565 |            |           | the control buffer where I_FDINSERT shall store a <b>t_uscalar_t</b> whose                    |
| 20566 |            |           | interpretation is specific to the STREAM end. The application shall ensure that               |
| 20567 |            |           | the <i>len</i> member in the <b>databuf strbuf</b> structure is set to the number of bytes of |
| 20568 |            |           | data information to be sent with the message, or to 0 if no data part is to be                |
| 20569 |            |           | sent.                                                                                         |
| 20570 |            |           | The <i>flags</i> member specifies the type of message to be created. A normal                 |
| 20571 |            |           | message is created if <i>flags</i> is set to 0, and a high-priority message is created if     |
| 20572 |            |           | <i>flags</i> is set to RS_HIPRI. For non-priority messages, I_FDINSERT shall block if         |
| 20573 |            |           | the STREAM write queue is full due to internal flow control conditions. For                   |
| 20574 |            |           | priority messages, I_FDINSERT does not block on this condition. For non-                      |
| 20575 |            |           | priority messages, I_FDINSERT does not block when the write queue is full                     |

20576 and O\_NONBLOCK is set. Instead, it fails and sets *errno* to [EAGAIN].

20577 I\_FDINSERT also blocks, unless prevented by lack of internal resources,  
 20578 waiting for the availability of message blocks in the STREAM, regardless of  
 20579 priority or whether O\_NONBLOCK has been specified. No partial message is  
 20580 sent.

20581 The *ioctl()* function with the I\_FDINSERT command shall fail if:

20582 [EAGAIN] A non-priority message is specified, the O\_NONBLOCK  
 20583 flag is set, and the STREAM write queue is full due to  
 20584 internal flow control conditions.

20585 [EAGAIN] or [ENOSR]  
 20586 Buffers cannot be allocated for the message that is to be  
 20587 created.

20588 [EINVAL] One of the following:

20589 — The *fildev* member of the **strfdinsert** structure is not a  
 20590 valid, open STREAM file descriptor.

20591 — The size of a **t\_uscalar\_t** plus *offset* is greater than the *len*  
 20592 member for the buffer specified through **ctlbuf**.

20593 — The *offset* member does not specify a properly-aligned  
 20594 location in the data buffer.

20595 — An undefined value is stored in *flags*.

20596 [ENXIO] Hangupt received on the STREAM identified by either the  
 20597 *fildev* argument or the *fildev* member of the **strfdinsert**  
 20598 structure.

20599 [ERANGE] The *len* member for the buffer specified through **databuf**  
 20600 does not fall within the range specified by the maximum  
 20601 and minimum packet sizes of the topmost STREAM module  
 20602 or the *len* member for the buffer specified through **databuf**  
 20603 is larger than the maximum configured size of the data part  
 20604 of a message; or the *len* member for the buffer specified  
 20605 through **ctlbuf** is larger than the maximum configured size  
 20606 of the control part of a message.

20607 I\_STR Constructs an internal STREAMS *ioctl()* message from the data pointed to by  
 20608 *arg*, and sends that message downstream.

20609 This mechanism is provided to send *ioctl()* requests to downstream modules  
 20610 and drivers. It allows information to be sent with *ioctl()*, and returns to the  
 20611 process any information sent upstream by the downstream recipient. I\_STR  
 20612 blocks until the system responds with either a positive or negative  
 20613 acknowledgement message, or until the request times out after some period of  
 20614 time. If the request times out, it fails with *errno* set to [ETIME].

20615 At most, one I\_STR can be active on a STREAM. Further I\_STR calls shall  
 20616 block until the active I\_STR completes at the STREAM head. The default  
 20617 timeout interval for these requests is 15 seconds. The O\_NONBLOCK flag has  
 20618 no effect on this call.

20619 To send requests downstream, the application shall ensure that *arg* points to a  
 20620 **strioc** structure.



|       |          |                                                                                                                                                                                                                                         |
|-------|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 20621 |          | The <i>ic_cmd</i> member is the internal <i>ioctl()</i> command intended for a downstream module or driver and <i>ic_timeout</i> is the number of seconds (−1=infinite, 0=use implementation-defined timeout interval, >0=as specified) |
| 20622 |          | an I_STR request shall wait for acknowledgement before timing out. <i>ic_len</i> is                                                                                                                                                     |
| 20623 |          | the number of bytes in the data argument, and <i>ic_dp</i> is a pointer to the data                                                                                                                                                     |
| 20624 |          | argument. The <i>ic_len</i> member has two uses: on input, it contains the length of                                                                                                                                                    |
| 20625 |          | the data argument passed in, and on return from the command, it contains the                                                                                                                                                            |
| 20626 |          | number of bytes being returned to the process (the buffer pointed to by <i>ic_dp</i>                                                                                                                                                    |
| 20627 |          | should be large enough to contain the maximum amount of data that any                                                                                                                                                                   |
| 20628 |          | module or the driver in the STREAM can return).                                                                                                                                                                                         |
| 20629 |          |                                                                                                                                                                                                                                         |
| 20630 |          |                                                                                                                                                                                                                                         |
| 20631 |          | The STREAM head shall convert the information pointed to by the <b>strioc</b>                                                                                                                                                           |
| 20632 |          | structure to an internal <i>ioctl()</i> command message and sends it downstream.                                                                                                                                                        |
| 20633 |          | The <i>ioctl()</i> function with the I_STR command shall fail if:                                                                                                                                                                       |
| 20634 |          | [EAGAIN] or [ENOSR]                                                                                                                                                                                                                     |
| 20635 |          | Unable to allocate buffers for the <i>ioctl()</i> message.                                                                                                                                                                              |
| 20636 |          | [EINVAL] The <i>ic_len</i> member is less than 0 or larger than the                                                                                                                                                                     |
| 20637 |          | maximum configured size of the data part of a message, or                                                                                                                                                                               |
| 20638 |          | <i>ic_timeout</i> is less than −1.                                                                                                                                                                                                      |
| 20639 |          | [ENXIO] Hangup received on <i>fil</i> des.                                                                                                                                                                                              |
| 20640 |          | [ETIME] A downstream <i>ioctl()</i> timed out before acknowledgement                                                                                                                                                                    |
| 20641 |          | was received.                                                                                                                                                                                                                           |
| 20642 |          | An I_STR can also fail while waiting for an acknowledgement if a message                                                                                                                                                                |
| 20643 |          | indicating an error or a hangup is received at the STREAM head. In addition,                                                                                                                                                            |
| 20644 |          | an error code can be returned in the positive or negative acknowledgement                                                                                                                                                               |
| 20645 |          | message, in the event the <i>ioctl()</i> command sent downstream fails. For these                                                                                                                                                       |
| 20646 |          | cases, I_STR fails with <i>errno</i> set to the value in the message.                                                                                                                                                                   |
| 20647 | I_SWROPT | Sets the write mode using the value of the argument <i>arg</i> . Valid bit settings for                                                                                                                                                 |
| 20648 |          | <i>arg</i> are:                                                                                                                                                                                                                         |
| 20649 |          | SNDZERO Send a zero-length message downstream when a <i>write()</i> of                                                                                                                                                                  |
| 20650 |          | 0 bytes occurs. To not send a zero-length message when a                                                                                                                                                                                |
| 20651 |          | <i>write()</i> of 0 bytes occurs, the application shall ensure that                                                                                                                                                                     |
| 20652 |          | this bit is not set in <i>arg</i> (for example, <i>arg</i> would be set to 0).                                                                                                                                                          |
| 20653 |          | The <i>ioctl()</i> function with the I_SWROPT command shall fail if:                                                                                                                                                                    |
| 20654 |          | [EINVAL] <i>arg</i> is not the above value.                                                                                                                                                                                             |
| 20655 | I_GWROPT | Returns the current write mode setting, as described above, in the <b>int</b> that is                                                                                                                                                   |
| 20656 |          | pointed to by the argument <i>arg</i> .                                                                                                                                                                                                 |
| 20657 | I_SENDFD | I_SENDFD creates a new reference to the open file description associated with                                                                                                                                                           |
| 20658 |          | the file descriptor <i>arg</i> , and writes a message on the STREAMS-based pipe                                                                                                                                                         |
| 20659 |          | <i>fil</i> des containing this reference, together with the user ID and group ID of the                                                                                                                                                 |
| 20660 |          | calling process.                                                                                                                                                                                                                        |
| 20661 |          | The <i>ioctl()</i> function with the I_SENDFD command shall fail if:                                                                                                                                                                    |
| 20662 |          | [EAGAIN] The sending STREAM is unable to allocate a message block                                                                                                                                                                       |
| 20663 |          | to contain the file pointer; or the read queue of the receiving                                                                                                                                                                         |
| 20664 |          | STREAM head is full and cannot accept the message sent by                                                                                                                                                                               |
| 20665 |          | I_SENDFD.                                                                                                                                                                                                                               |

|       |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------|----------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 20666 |          | [EBADF]             | The <i>arg</i> argument is not a valid, open file descriptor.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 20667 |          | [EINVAL]            | The <i>fildev</i> argument is not connected to a STREAM pipe.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 20668 |          | [ENXIO]             | Hangup received on <i>fildev</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 20669 | I_RECVFD |                     | Retrieves the reference to an open file description from a message written to a STREAMS-based pipe using the I_SENDFD command, and allocates a new file descriptor in the calling process that refers to this open file description. The <i>arg</i> argument is a pointer to a <b>strrecvfd</b> data structure as defined in <b>&lt;stropts.h&gt;</b> .                                                                                                                                                                                                                                                                                                         |
| 20670 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20671 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20672 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20673 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20674 |          |                     | The <i>fd</i> member is a file descriptor. The <i>uid</i> and <i>gid</i> members are the effective user ID and effective group ID, respectively, of the sending process.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 20675 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20676 |          |                     | If O_NONBLOCK is not set, I_RECVFD blocks until a message is present at the STREAM head. If O_NONBLOCK is set, I_RECVFD fails with <i>errno</i> set to [EAGAIN] if no message is present at the STREAM head.                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 20677 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20678 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20679 |          |                     | If the message at the STREAM head is a message sent by an I_SENDFD, a new file descriptor is allocated for the open file descriptor referenced in the message. The new file descriptor is placed in the <i>fd</i> member of the <b>strrecvfd</b> structure pointed to by <i>arg</i> .                                                                                                                                                                                                                                                                                                                                                                           |
| 20680 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20681 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20682 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20683 |          |                     | The <i>ioctl()</i> function with the I_RECVFD command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20684 |          | [EAGAIN]            | A message is not present at the STREAM head read queue and the O_NONBLOCK flag is set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20685 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20686 |          | [EBADMSG]           | The message at the STREAM head read queue is not a message containing a passed file descriptor.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20687 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20688 |          | [EMFILE]            | The process has the maximum number of file descriptors currently open that it is allowed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20689 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20690 |          | [ENXIO]             | Hangup received on <i>fildev</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 20691 | I_LIST   |                     | This request allows the process to list all the module names on the STREAM, up to and including the topmost driver name. If <i>arg</i> is a null pointer, the return value is the number of modules, including the driver, that are on the STREAM pointed to by <i>fildev</i> . This lets the process allocate enough space for the module names. Otherwise, it should point to a <b>str_list</b> structure.                                                                                                                                                                                                                                                    |
| 20692 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20693 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20694 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20695 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20696 |          |                     | The <i>sl_nmods</i> member indicates the number of entries the process has allocated in the array. Upon return, the <i>sl_modlist</i> member of the <b>str_list</b> structure contains the list of module names, and the number of entries that have been filled into the <i>sl_modlist</i> array is found in the <i>sl_nmods</i> member (the number includes the number of modules including the driver). The return value from <i>ioctl()</i> is 0. The entries are filled in starting at the top of the STREAM and continuing downstream until either the end of the STREAM is reached, or the number of requested modules ( <i>sl_nmods</i> ) is satisfied. |
| 20697 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20698 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20699 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20700 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20701 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20702 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20703 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20704 |          |                     | The <i>ioctl()</i> function with the I_LIST command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 20705 |          | [EINVAL]            | The <i>sl_nmods</i> member is less than 1.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 20706 |          | [EAGAIN] or [ENOSR] |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20707 |          |                     | Unable to allocate buffers.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20708 | I_ATMARK |                     | This request allows the process to see if the message at the head of the STREAM head read queue is marked by some module downstream. The <i>arg</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 20709 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

|       |             |                                                                                           |
|-------|-------------|-------------------------------------------------------------------------------------------|
| 20710 |             | argument determines how the checking is done when there may be multiple                   |
| 20711 |             | marked messages on the STREAM head read queue. It may take on the                         |
| 20712 |             | following values:                                                                         |
| 20713 |             | ANYMARK      Check if the message is marked.                                              |
| 20714 |             | LASTMARK     Check if the message is the last one marked on the queue.                    |
| 20715 |             | The bitwise-inclusive OR of the flags ANYMARK and LASTMARK is                             |
| 20716 |             | permitted.                                                                                |
| 20717 |             | The return value is 1 if the mark condition is satisfied; otherwise, the value is         |
| 20718 |             | 0.                                                                                        |
| 20719 |             | The <i>ioctl()</i> function with the L_ATMARK command shall fail if:                      |
| 20720 |             | [EINVAL]      Invalid <i>arg</i> value.                                                   |
| 20721 | I_CKBAND    | Check if the message of a given priority band exists on the STREAM head                   |
| 20722 |             | read queue. This returns 1 if a message of the given priority exists, 0 if no such        |
| 20723 |             | message exists, or -1 on error. <i>arg</i> should be of type <b>int</b> .                 |
| 20724 |             | The <i>ioctl()</i> function with the L_CKBAND command shall fail if:                      |
| 20725 |             | [EINVAL]      Invalid <i>arg</i> value.                                                   |
| 20726 | I_GETBAND   | Return the priority band of the first message on the STREAM head read queue               |
| 20727 |             | in the integer referenced by <i>arg</i> .                                                 |
| 20728 |             | The <i>ioctl()</i> function with the L_GETBAND command shall fail if:                     |
| 20729 |             | [ENODATA]    No message on the STREAM head read queue.                                    |
| 20730 | I_CANPUT    | Check if a certain band is writable. <i>arg</i> is set to the priority band in question.  |
| 20731 |             | The return value is 0 if the band is flow-controlled, 1 if the band is writable, or       |
| 20732 |             | -1 on error.                                                                              |
| 20733 |             | The <i>ioctl()</i> function with the L_CANPUT command shall fail if:                      |
| 20734 |             | [EINVAL]      Invalid <i>arg</i> value.                                                   |
| 20735 | I_SETCLTIME | This request allows the process to set the time the STREAM head shall delay               |
| 20736 |             | when a STREAM is closing and there is data on the write queues. Before                    |
| 20737 |             | closing each module or driver, if there is data on its write queue, the STREAM            |
| 20738 |             | head shall delay for the specified amount of time to allow the data to drain. If,         |
| 20739 |             | after the delay, data is still present, it shall be flushed. The <i>arg</i> argument is a |
| 20740 |             | pointer to an integer specifying the number of milliseconds to delay, rounded             |
| 20741 |             | up to the nearest valid value. If I_SETCLTIME is not performed on a STREAM,               |
| 20742 |             | an implementation-defined default timeout interval is used.                               |
| 20743 |             | The <i>ioctl()</i> function with the L_SETCLTIME command shall fail if:                   |
| 20744 |             | [EINVAL]      Invalid <i>arg</i> value.                                                   |
| 20745 | I_GETCLTIME | This request returns the close time delay in the integer pointed to by <i>arg</i> .       |

20746 **Multiplexed STREAMS Configurations**

20747 The following commands are used for connecting and disconnecting multiplexed STREAMS  
20748 configurations. These commands use an implementation-defined default timeout interval.

20749 **I\_LINK** Connects two STREAMs, where *fildev* is the file descriptor of the STREAM  
20750 connected to the multiplexing driver, and *arg* is the file descriptor of the  
20751 STREAM connected to another driver. The STREAM designated by *arg* is  
20752 connected below the multiplexing driver. I\_LINK requires the multiplexing  
20753 driver to send an acknowledgement message to the STREAM head regarding  
20754 the connection. This call returns a multiplexer ID number (an identifier used  
20755 to disconnect the multiplexer; see I\_UNLINK) on success, and -1 on failure.

20756 The *ioctl()* function with the I\_LINK command shall fail if:

20757 [ENXIO] Hangup received on *fildev*.

20758 [ETIME] Timeout before acknowledgement message was received at  
20759 STREAM head.

20760 [EAGAIN] or [ENOSR]

20761 Unable to allocate STREAMS storage to perform the  
20762 I\_LINK.

20763 [EBADF] The *arg* argument is not a valid, open file descriptor.

20764 [EINVAL] The *fildev* argument does not support multiplexing; or *arg* is  
20765 not a STREAM or is already connected downstream from a  
20766 multiplexer; or the specified I\_LINK operation would  
20767 connect the STREAM head in more than one place in the  
20768 multiplexed STREAM.

20769 An I\_LINK can also fail while waiting for the multiplexing driver to  
20770 acknowledge the request, if a message indicating an error or a hangup is  
20771 received at the STREAM head of *fildev*. In addition, an error code can be  
20772 returned in the positive or negative acknowledgement message. For these  
20773 cases, I\_LINK fails with *errno* set to the value in the message.

20774 **I\_UNLINK** Disconnects the two STREAMs specified by *fildev* and *arg*. *fildev* is the file  
20775 descriptor of the STREAM connected to the multiplexing driver. The *arg*  
20776 argument is the multiplexer ID number that was returned by the I\_LINK  
20777 *ioctl()* command when a STREAM was connected downstream from the  
20778 multiplexing driver. If *arg* is MUXID\_ALL, then all STREAMs that were  
20779 connected to *fildev* are disconnected. As in I\_LINK, this command requires  
20780 acknowledgement.

20781 The *ioctl()* function with the I\_UNLINK command shall fail if:

20782 [ENXIO] Hangup received on *fildev*.

20783 [ETIME] Timeout before acknowledgement message was received at  
20784 STREAM head.

20785 [EAGAIN] or [ENOSR]

20786 Unable to allocate buffers for the acknowledgement  
20787 message.

20788 [EINVAL] Invalid multiplexer ID number.

20789 An I\_UNLINK can also fail while waiting for the multiplexing driver to  
 20790 acknowledge the request if a message indicating an error or a hangup is  
 20791 received at the STREAM head of *filde*s. In addition, an error code can be  
 20792 returned in the positive or negative acknowledgement message. For these  
 20793 cases, I\_UNLINK fails with *errno* set to the value in the message.

20794 I\_PLINK Creates a *persistent connection* between two STREAMs, where *filde*s is the file  
 20795 descriptor of the STREAM connected to the multiplexing driver, and *arg* is the  
 20796 file descriptor of the STREAM connected to another driver. This call creates a  
 20797 persistent connection which can exist even if the file descriptor *filde*s  
 20798 associated with the upper STREAM to the multiplexing driver is closed. The  
 20799 STREAM designated by *arg* gets connected via a persistent connection below  
 20800 the multiplexing driver. I\_PLINK requires the multiplexing driver to send an  
 20801 acknowledgement message to the STREAM head. This call returns a  
 20802 multiplexer ID number (an identifier that may be used to disconnect the  
 20803 multiplexer; see I\_PUNLINK) on success, and -1 on failure.

20804 The *ioctl*() function with the I\_PLINK command shall fail if:

20805 [ENXIO] Hangup received on *filde*s.

20806 [ETIME] Timeout before acknowledgement message was received at  
 20807 STREAM head.

20808 [EAGAIN] or [ENOSR]  
 20809 Unable to allocate STREAMS storage to perform the  
 20810 I\_PLINK.

20811 [EBADF] The *arg* argument is not a valid, open file descriptor.

20812 [EINVAL] The *filde*s argument does not support multiplexing; or *arg* is  
 20813 not a STREAM or is already connected downstream from a  
 20814 multiplexer; or the specified I\_PLINK operation would  
 20815 connect the STREAM head in more than one place in the  
 20816 multiplexed STREAM.

20817 An I\_PLINK can also fail while waiting for the multiplexing driver to  
 20818 acknowledge the request, if a message indicating an error or a hangup is  
 20819 received at the STREAM head of *filde*s. In addition, an error code can be  
 20820 returned in the positive or negative acknowledgement message. For these  
 20821 cases, I\_PLINK fails with *errno* set to the value in the message.

20822 I\_PUNLINK Disconnects the two STREAMs specified by *filde*s and *arg* from a persistent  
 20823 connection. The *filde*s argument is the file descriptor of the STREAM  
 20824 connected to the multiplexing driver. The *arg* argument is the multiplexer ID  
 20825 number that was returned by the I\_PLINK *ioctl*() command when a STREAM  
 20826 was connected downstream from the multiplexing driver. If *arg* is  
 20827 MUXID\_ALL, then all STREAMs which are persistent connections to *filde*s  
 20828 are disconnected. As in I\_PLINK, this command requires the multiplexing driver  
 20829 to acknowledge the request.

20830 The *ioctl*() function with the I\_PUNLINK command shall fail if:

20831 [ENXIO] Hangup received on *filde*s.

20832 [ETIME] Timeout before acknowledgement message was received at  
 20833 STREAM head.

20834 [EAGAIN] or [ENOSR]  
 20835 Unable to allocate buffers for the acknowledgement  
 20836 message.

20837 [EINVAL] Invalid multiplexer ID number.

20838 An I\_PUNLINK can also fail while waiting for the multiplexing driver to  
 20839 acknowledge the request if a message indicating an error or a hangup is  
 20840 received at the STREAM head of *fildev*. In addition, an error code can be  
 20841 returned in the positive or negative acknowledgement message. For these  
 20842 cases, I\_PUNLINK fails with *errno* set to the value in the message.

#### 20843 RETURN VALUE

20844 Upon successful completion, *ioctl()* shall return a value other than  $-1$  that depends upon the  
 20845 STREAMS device control function. Otherwise, it shall return  $-1$  and set *errno* to indicate the  
 20846 error.

#### 20847 ERRORS

20848 Under the following general conditions, *ioctl()* shall fail if:

20849 [EBADF] The *fildev* argument is not a valid open file descriptor.

20850 [EINTR] A signal was caught during the *ioctl()* operation.

20851 [EINVAL] The STREAM or multiplexer referenced by *fildev* is linked (directly or  
 20852 indirectly) downstream from a multiplexer.

20853 If an underlying device driver detects an error, then *ioctl()* shall fail if:

20854 [EINVAL] The *request* or *arg* argument is not valid for this device.

20855 [EIO] Some physical I/O error has occurred.

20856 [ENOTTY] The *fildev* argument is not associated with a STREAMS device that accepts  
 20857 control functions.

20858 [ENXIO] The *request* and *arg* arguments are valid for this device driver, but the service  
 20859 requested cannot be performed on this particular sub-device.

20860 [ENODEV] The *fildev* argument refers to a valid STREAMS device, but the corresponding  
 20861 device driver does not support the *ioctl()* function.

20862 If a STREAM is connected downstream from a multiplexer, any *ioctl()* command except  
 20863 I\_UNLINK and I\_PUNLINK shall set *errno* to [EINVAL].

#### 20864 EXAMPLES

20865 None.

#### 20866 APPLICATION USAGE

20867 The implementation-defined timeout interval for STREAMS has historically been 15 seconds.

#### 20868 RATIONALE

20869 None.

#### 20870 FUTURE DIRECTIONS

20871 None.

#### 20872 SEE ALSO

20873 *close()*, *fcntl()*, *getmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *read()*, *sigaction()*, *write()*, the Base  
 20874 Definitions volume of IEEE Std. 1003.1-200x, <**stropts.h**>, Section 2.6 (on page 539)

20875 **CHANGE HISTORY**

20876 First released in Issue 4, Version 2.

20877 **Issue 5**

20878 Moved from X/OPEN UNIX extension to BASE.

20879 **Issue 6**

20880 The Open Group corrigenda item U028/4 has been applied, correcting text in the I\_FDINSERT, [EINVAL] case to refer to *ctlbuf*.

20882 This function is marked as part of the XSI STREAMS Option Group.

20883 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20884 **NAME**

20885           isalnum — test for an alphanumeric character

20886 **SYNOPSIS**

20887           #include <ctype.h>

20888           int isalnum(int c);

20889 **DESCRIPTION**

20890 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
20891 conflict between the requirements described here and the ISO C standard is unintentional. This  
20892 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

20893       The *isalnum()* function tests whether *c* is a character of class **alpha** or **digit** in the program's  
20894 current locale; see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale.

20895       In all cases *c* is an **int**, the value of which the application shall ensure is representable as an  
20896 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the  
20897 behavior is undefined.

20898 **RETURN VALUE**

20899       The *isalnum()* function shall return non-zero if *c* is an alphanumeric character; otherwise, it shall  
20900 return 0.

20901 **ERRORS**

20902       No errors are defined.

20903 **EXAMPLES**

20904       None.

20905 **APPLICATION USAGE**

20906       To ensure applications portability, especially across natural languages, only this function and  
20907 those listed in the SEE ALSO section should be used for character classification.

20908 **RATIONALE**

20909       None.

20910 **FUTURE DIRECTIONS**

20911       None.

20912 **SEE ALSO**

20913       *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*,  
20914 *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>, <stdio.h>, the Base  
20915 Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

20916 **CHANGE HISTORY**

20917       First released in Issue 1. Derived from Issue 1 of the SVID.

20918 **Issue 4**

20919       The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no  
20920 functional differences between this issue and Issue 3. Operation in the C locale is no longer  
20921 described explicitly on this reference page.

20922 **Issue 6**

20923       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.



20924 **NAME**

20925 isalpha — test for an alphabetic character

20926 **SYNOPSIS**

20927 #include <ctype.h>

20928 int isalpha(int c);

20929 **DESCRIPTION**

20930 **cx** The functionality described on this reference page is aligned with the ISO C standard. Any  
20931 conflict between the requirements described here and the ISO C standard is unintentional. This  
20932 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

20933 The *isalpha()* function shall test whether *c* is a character of class **alpha** in the program's current  
20934 locale; see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale.

20935 In all cases *c* is an **int**, the value of which the application shall ensure is representable as an  
20936 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the  
20937 behavior is undefined.

20938 **RETURN VALUE**

20939 The *isalpha()* function shall return non-zero if *c* is an alphabetic character; otherwise, it shall  
20940 return 0.

20941 **ERRORS**

20942 No errors are defined.

20943 **EXAMPLES**

20944 None.

20945 **APPLICATION USAGE**

20946 To ensure applications portability, especially across natural languages, only this function and  
20947 those listed in the SEE ALSO section should be used for character classification.

20948 **RATIONALE**

20949 None.

20950 **FUTURE DIRECTIONS**

20951 None.

20952 **SEE ALSO**

20953 *isalnum()*, *isctrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
20954 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>, <stdio.h>,  
20955 the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

20956 **CHANGE HISTORY**

20957 First released in Issue 1. Derived from Issue 1 of the SVID.

20958 **Issue 4**

20959 The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no  
20960 functional differences between this issue and Issue 3. Operation in the C locale is no longer  
20961 described explicitly on this reference page.

20962 **Issue 6**

20963 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20964 **NAME**

20965           isascii — test for a 7-bit US-ASCII character

20966 **SYNOPSIS**

20967 xSI       #include &lt;ctype.h&gt;

20968           int isascii(int c);

20969

20970 **DESCRIPTION**20971           The *isascii()* function tests whether *c* is a 7-bit US-ASCII character code.20972           The *isascii()* function is defined on all integer values.20973 **RETURN VALUE**20974           The *isascii()* function shall return non-zero if *c* is a 7-bit US-ASCII character code between 0 and

20975           octal 0177 inclusive; otherwise, it shall return 0.

20976 **ERRORS**

20977           No errors are defined.

20978 **EXAMPLES**

20979           None.

20980 **APPLICATION USAGE**

20981           None.

20982 **RATIONALE**

20983           None.

20984 **FUTURE DIRECTIONS**

20985           None.

20986 **SEE ALSO**

20987           The Base Definitions volume of IEEE Std. 1003.1-200x, &lt;ctype.h&gt;

20988 **CHANGE HISTORY**

20989           First released in Issue 1. Derived from Issue 1 of the SVID.

20990 **NAME**20991 isastream — test a file descriptor (**STREAMS**)20992 **SYNOPSIS**20993 XSR `#include <stropts.h>`20994 `int isastream(int fildev);`

20995

20996 **DESCRIPTION**20997 The *isastream()* function shall test whether *fildev*, an open file descriptor, is associated with a  
20998 STREAMS-based file.20999 **RETURN VALUE**21000 Upon successful completion, *isastream()* shall return 1 if *fildev* refers to a STREAMS-based file  
21001 and 0 if not. Otherwise, *isastream()* shall return -1 and set *errno* to indicate the error.21002 **ERRORS**21003 The *isastream()* function shall fail if:21004 [EBADF] The *fildev* argument is not a valid open file descriptor.21005 **EXAMPLES**

21006 None.

21007 **APPLICATION USAGE**

21008 None.

21009 **RATIONALE**

21010 None.

21011 **FUTURE DIRECTIONS**

21012 None.

21013 **SEE ALSO**

21014 The Base Definitions volume of IEEE Std. 1003.1-200x, &lt;stropts.h&gt;

21015 **CHANGE HISTORY**

21016 First released in Issue 4, Version 2.

21017 **Issue 5**

21018 Moved from X/OPEN UNIX extension to BASE.

21019 **NAME**

21020           isatty — test for a terminal device

21021 **SYNOPSIS**

21022           #include <unistd.h>

21023           int isatty(int *fildev*);

21024 **DESCRIPTION**

21025           The *isatty()* function shall test whether *fildev*, an open file descriptor, is associated with a  
21026           terminal device.

21027 **RETURN VALUE**

21028           The *isatty()* function shall return 1 if *fildev* is associated with a terminal; otherwise, it shall return  
21029           0 and may set *errno* to indicate the error.

21030 **ERRORS**

21031           The *isatty()* function may fail if:

21032           [EBADF]           The *fildev* argument is not a valid open file descriptor.

21033           [ENOTTY]         The *fildev* argument is not associated with a terminal.

21034 **EXAMPLES**

21035           None.

21036 **APPLICATION USAGE**

21037           The *isatty()* function does not necessarily indicate that a human being is available for interaction  
21038           via *fildev*. It is quite possible that non-terminal devices are connected to the communications  
21039           line.

21040 **RATIONALE**

21041           None.

21042 **FUTURE DIRECTIONS**

21043           None.

21044 **SEE ALSO**

21045           The Base Definitions volume of IEEE Std. 1003.1-200x, <unistd.h>

21046 **CHANGE HISTORY**

21047           First released in Issue 1. Derived from Issue 1 of the SVID.

21048 **Issue 4**

21049           The <unistd.h> header is added to the SYNOPSIS section.

21050           In the RETURN VALUE section, the sentence indicating that this function may set *errno* is  
21051           marked as an extension.

21052           The errors [EBADF] and [ENOTTY] are marked as extensions.

21053 **Issue 6**

21054           The following new requirements on POSIX implementations derive from alignment with the  
21055           Single UNIX Specification:

- 21056           • The optional setting of *errno* to indicate an error is added.
- 21057           • The [EBADF] and [ENOTTY] optional error conditions are added.

21058 **NAME**

21059           isblank — test for a blank character

21060 **SYNOPSIS**

21061           #include <ctype.h>

21062           int isblank(int *c*);

21063 **DESCRIPTION**

21064 *cx*       The functionality described on this reference page is aligned with the ISO C standard. Any  
21065           conflict between the requirements described here and the ISO C standard is unintentional. This  
21066           volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21067           The *isblank()* function shall test whether *c* is a character of class **blank** in the program's current  
21068           locale; see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale. In all cases *c*  
21069           is a type **int**, the value of which the application shall ensure is a character representable as an  
21070           **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the  
21071           behavior is undefined.

21072 **RETURN VALUE**

21073           The *isblank()* function shall return non-zero if *c* is a <blank> character; otherwise, it shall return  
21074           0.

21075 **ERRORS**

21076           No errors are defined.

21077 **EXAMPLES**

21078           None.

21079 **APPLICATION USAGE**

21080           To ensure applications portability, especially across natural languages, only this function and  
21081           those listed in the SEE ALSO section should be used for character classification.

21082 **RATIONALE**

21083           None.

21084 **FUTURE DIRECTIONS**

21085           None.

21086 **SEE ALSO**

21087           *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
21088           *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>

21089 **CHANGE HISTORY**

21090           First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21091 **NAME**

21092 isctrl — test for a control character

21093 **SYNOPSIS**

21094 #include &lt;ctype.h&gt;

21095 int isctrl(int c);

21096 **DESCRIPTION**

21097 *cx* The functionality described on this reference page is aligned with the ISO C standard. Any  
21098 conflict between the requirements described here and the ISO C standard is unintentional. This  
21099 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21100 The *isctrl()* function shall test whether *c* is a character of class **cntrl** in the program's current  
21101 locale; see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale.

21102 In all cases *c* is a type **int**, the value of which the application shall ensure is a character  
21103 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
21104 any other value, the behavior is undefined.

21105 **RETURN VALUE**21106 The *isctrl()* function shall return non-zero if *c* is a control character; otherwise, it shall return 0.21107 **ERRORS**

21108 No errors are defined.

21109 **EXAMPLES**

21110 None.

21111 **APPLICATION USAGE**

21112 To ensure applications portability, especially across natural languages, only this function and  
21113 those listed in the SEE ALSO section should be used for character classification.

21114 **RATIONALE**

21115 None.

21116 **FUTURE DIRECTIONS**

21117 None.

21118 **SEE ALSO**

21119 *isalnum()*, *isalpha()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
21120 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>, the Base  
21121 Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

21122 **CHANGE HISTORY**

21123 First released in Issue 1. Derived from Issue 1 of the SVID.

21124 **Issue 4**

21125 The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no  
21126 functional differences between this issue and Issue 3. Operation in the C locale is no longer  
21127 described explicitly on this reference page.

21128 **Issue 6**

21129 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21130 **NAME**

21131            isdigit — test for a decimal digit

21132 **SYNOPSIS**

21133            #include <ctype.h>

21134            int isdigit(int c);

21135 **DESCRIPTION**

21136 **cx**        The functionality described on this reference page is aligned with the ISO C standard. Any  
21137            conflict between the requirements described here and the ISO C standard is unintentional. This  
21138            volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21139            The *isdigit()* function shall test whether *c* is a character of class **digit** in the program's current  
21140            locale; see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale.

21141            In all cases *c* is an **int**, the value of which the application shall ensure is a character representable  
21142            as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value,  
21143            the behavior is undefined.

21144 **RETURN VALUE**

21145            The *isdigit()* function shall return non-zero if *c* is a decimal digit; otherwise, it shall return 0.

21146 **ERRORS**

21147            No errors are defined.

21148 **EXAMPLES**

21149            None.

21150 **APPLICATION USAGE**

21151            To ensure applications portability, especially across natural languages, only this function and  
21152            those listed in the SEE ALSO section should be used for character classification.

21153 **RATIONALE**

21154            None.

21155 **FUTURE DIRECTIONS**

21156            None.

21157 **SEE ALSO**

21158            *isalnum()*, *isalpha()*, *iscntrl()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
21159            *isxdigit()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>

21160 **CHANGE HISTORY**

21161            First released in Issue 1. Derived from Issue 1 of the SVID.

21162 **Issue 4**

21163            The text of the DESCRIPTION is revised, although there are no functional differences between  
21164            this issue and Issue 3.

21165 **Issue 6**

21166            The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21167 **NAME**

21168       isfdtype — determine whether a file descriptor refers to a socket

21169 **SYNOPSIS**

21170       #include <sys/stat.h>

21171       int isfdtype(int *fildev*, int *fdtype*);

21172 **DESCRIPTION**

21173       The *isfdtype()* function shall determine whether the descriptor *fildev* has the properties identified  
21174       by the value of *fdtype*, returning 1 if so, and 0 if not.

21175       If *fdtype* has the value S\_IFSOCK:

- 21176       • The *isfdtype()* function shall return 1 if the descriptor refers to a socket.
- 21177       • It is implementation-defined whether *isfdtype()* shall return 1 if the descriptor refers to a  
21178       pipe.
- 21179       • The function shall return 0 for descriptors that refer neither to a socket nor to a pipe.

21180 **RETURN VALUE**

21181       Upon successful completion, the *isfdtype()* function shall return a value of 1 or 0 indicating  
21182       whether the descriptor is of the indicated type. Otherwise, it shall return a value of -1 and set  
21183       *errno* to indicate the error.

21184 **ERRORS**

21185       If any of the following conditions occur, the *isfdtype()* function shall return -1 and set *errno* to  
21186       the corresponding value:

21187       [EBADF]       The *fildev* argument is not a valid file descriptor.

21188 **EXAMPLES**

21189       None.

21190 **APPLICATION USAGE**

21191       None.

21192 **RATIONALE**

21193       None.

21194 **FUTURE DIRECTIONS**

21195       None.

21196 **SEE ALSO**

21197       *isatty()*, *socket()*, *stat()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/stat.h>

21198 **CHANGE HISTORY**

21199       First released in Issue 6. Derived from IEEE Std. 1003.1g-2000.



21200 **NAME**

21201 isfinite — test for finite value

21202 **SYNOPSIS**

21203 #include &lt;math.h&gt;

21204 int isfinite(real-floating x);

21205 **DESCRIPTION**

21206 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
21207 conflict between the requirements described here and the ISO C standard is unintentional. This  
21208 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21209 The *isfinite()* macro shall determine whether its argument has a finite value (zero, subnormal, or  
21210 normal, and not infinite or NaN). First, an argument represented in a format wider than its  
21211 semantic type is converted to its semantic type. Then determination is based on the type of the  
21212 argument.

21213 **RETURN VALUE**21214 The *isfinite()* macro shall return a non-zero value if and only if its argument has a finite value.21215 **ERRORS**

21216 No errors are defined.

21217 **EXAMPLES**

21218 None.

21219 **APPLICATION USAGE**

21220 None.

21221 **RATIONALE**

21222 None.

21223 **FUTURE DIRECTIONS**

21224 None.

21225 **SEE ALSO**

21226 *fpclassify()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*, the Base Definitions volume of  
21227 IEEE Std. 1003.1-200x <math.h>

21228 **CHANGE HISTORY**

21229 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21230 **NAME**

21231 isgraph — test for a visible character

21232 **SYNOPSIS**

21233 #include &lt;ctype.h&gt;

21234 int isgraph(int c);

21235 **DESCRIPTION**

21236 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
21237 conflict between the requirements described here and the ISO C standard is unintentional. This  
21238 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21239 The *isgraph()* function shall test whether *c* is a character of class **graph** in the program's current  
21240 locale; see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale.

21241 In all cases *c* is an **int**, the value of which the application shall ensure is a character representable  
21242 as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value,  
21243 the behavior is undefined.

21244 **RETURN VALUE**

21245 The *isgraph()* function shall return non-zero if *c* is a character with a visible representation;  
21246 otherwise, it shall return 0.

21247 **ERRORS**

21248 No errors are defined.

21249 **EXAMPLES**

21250 None.

21251 **APPLICATION USAGE**

21252 To ensure applications portability, especially across natural languages, only this function and  
21253 those listed in the SEE ALSO section should be used for character classification.

21254 **RATIONALE**

21255 None.

21256 **FUTURE DIRECTIONS**

21257 None.

21258 **SEE ALSO**

21259 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*,  
21260 *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>, the Base Definitions  
21261 volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

21262 **CHANGE HISTORY**

21263 First released in Issue 1. Derived from Issue 1 of the SVID.

21264 **Issue 4**

21265 The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no  
21266 functional differences between this issue and Issue 3. Operation in the C locale is no longer  
21267 described explicitly on this reference page.

21268 **Issue 6**

21269 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21270 **NAME**

21271 `isgreater` — test if  $x$  greater than  $y$

21272 **SYNOPSIS**

21273 `#include <math.h>`

21274 `int isgreater(real-floating  $x$ , real-floating  $y$ );`

21275 **DESCRIPTION**

21276 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21279 The `isgreater()` macro shall determine whether its first argument is greater than its second argument. The value of `isgreater( $x$ ,  $y$ )` shall be equal to  $(x) > (y)$ ; however, unlike  $(x) > (y)$ , `isgreater( $x$ ,  $y$ )` shall not raise the invalid floating-point exception when  $x$  and  $y$  are unordered.

21282 **RETURN VALUE**

21283 Upon successful completion, the `isgreater()` macro shall return the value of  $(x) > (y)$ .

21284 If  $x$  or  $y$  is NaN, 0 shall be returned.

21285 **ERRORS**

21286 No errors are defined.

21287 **EXAMPLES**

21288 None.

21289 **APPLICATION USAGE**

21290 The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators may raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for NaNs without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating** indicates that the argument shall be an expression of **real-floating** type.

21298 **RATIONALE**

21299 None.

21300 **FUTURE DIRECTIONS**

21301 None.

21302 **SEE ALSO**

21303 `isgreaterequal()`, `isless()`, `islessequal()`, `islessgreater()`, `isunordered()`, the Base Definitions volume of IEEE Std. 1003.1-200x `<math.h>`

21305 **CHANGE HISTORY**

21306 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21307 **NAME**

21308           isgreaterqual — test if  $x$  greater than or equal to  $y$

21309 **SYNOPSIS**

21310           #include <math.h>

21311           int isgreaterqual(real-floating  $x$ , real-floating  $y$ );

21312 **DESCRIPTION**

21313 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21314 conflict between the requirements described here and the ISO C standard is unintentional. This  
21315 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21316       The *isgreaterqual()* macro shall determine whether its first argument is greater than or equal to  
21317 its second argument. The value of *isgreaterqual*( $x$ ,  $y$ ) shall be equal to  $(x) >= (y)$ ; however, unlike  
21318  $(x) >= (y)$ , *isgreaterqual*( $x$ ,  $y$ ) shall not raise the invalid floating-point exception when  $x$  and  $y$  are  
21319 unordered.

21320 **RETURN VALUE**

21321       Upon successful completion, the *isgreaterqual()* macro shall return the value of  $(x) >= (y)$ .

21322       If  $x$  or  $y$  is NaN, 0 shall be returned.

21323 **ERRORS**

21324       No errors are defined.

21325 **EXAMPLES**

21326       None.

21327 **APPLICATION USAGE**

21328       The relational and equality operators support the usual mathematical relationships between  
21329 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
21330 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
21331 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
21332 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
21333 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
21334 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
21335 indicates that the argument shall be an expression of **real-floating** type.

21336 **RATIONALE**

21337       None.

21338 **FUTURE DIRECTIONS**

21339       None.

21340 **SEE ALSO**

21341       *isgreater()*, *isless()*, *islessequal()*, *islessgreater()*, *isunordered()*, the Base Definitions volume of  
21342 IEEE Std. 1003.1-200x <math.h>

21343 **CHANGE HISTORY**

21344       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21345 **NAME**

21346           isinf — test for infinity

21347 **SYNOPSIS**

21348           #include &lt;math.h&gt;

21349           int isinf(real-floating x);

21350 **DESCRIPTION**

21351 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
21352       conflict between the requirements described here and the ISO C standard is unintentional. This  
21353       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21354       The *isinf()* macro shall determine whether its argument value is an infinity (positive or  
21355       negative). First, an argument represented in a format wider than its semantic type is converted  
21356       to its semantic type. Then determination is based on the type of the argument.

21357 **RETURN VALUE**21358       The *isinf()* macro shall return a non-zero value if and only if its argument has an infinite value.21359 **ERRORS**

21360       No errors are defined.

21361 **EXAMPLES**

21362       None.

21363 **APPLICATION USAGE**

21364       None.

21365 **RATIONALE**

21366       None.

21367 **FUTURE DIRECTIONS**

21368       None.

21369 **SEE ALSO**

21370       *fpclassify()*, *isfinite()*, *isnan()*, *isnormal()*, *signbit()*, the Base Definitions volume of  
21371       IEEE Std. 1003.1-200x <math.h>

21372 **CHANGE HISTORY**

21373       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21374 **NAME**

21375 isless — test if *x* is less than *y*

21376 **SYNOPSIS**

21377 #include <math.h>

21378 int isless(real-floating *x*, real-floating *y*);

21379 **DESCRIPTION**

21380 *cx* The functionality described on this reference page is aligned with the ISO C standard. Any  
21381 conflict between the requirements described here and the ISO C standard is unintentional. This  
21382 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21383 The *isless()* macro shall determine whether its first argument is less than its second argument.  
21384 The value of *isless(x, y)* shall be equal to  $(x) < (y)$ ; however, unlike  $(x) < (y)$ , *isless(x, y)* shall not  
21385 raise the invalid floating-point exception when *x* and *y* are unordered.

21386 **RETURN VALUE**

21387 Upon successful completion, the *isless()* macro shall return the value of  $(x) < (y)$ .

21388 If *x* or *y* is NaN, 0 shall be returned.

21389 **ERRORS**

21390 No errors are defined.

21391 **EXAMPLES**

21392 None.

21393 **APPLICATION USAGE**

21394 The relational and equality operators support the usual mathematical relationships between  
21395 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
21396 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
21397 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
21398 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
21399 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
21400 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
21401 indicates that the argument shall be an expression of **real-floating** type.

21402 **RATIONALE**

21403 None.

21404 **FUTURE DIRECTIONS**

21405 None.

21406 **SEE ALSO**

21407 *isgreater()*, *isgreaterequal()*, *islessequal()*, *islessgreater()*, *isunordered()*, the Base Definitions volume  
21408 of IEEE Std. 1003.1-200x, <math.h>

21409 **CHANGE HISTORY**

21410 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21411 **NAME**

21412 islessequal — test if *x* is less than or equal to *y*

21413 **SYNOPSIS**

21414 #include <math.h>

21415 int islessequal(real-floating *x*, real-floating *y*);

21416 **DESCRIPTION**

21417 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
21418 conflict between the requirements described here and the ISO C standard is unintentional. This  
21419 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21420 The *islessequal()* macro shall determine whether its first argument is less than or equal to its  
21421 second argument. The value of *islessequal(x, y)* shall be equal to  $(x) \leq (y)$ ; however, unlike  
21422  $(x) \leq (y)$ , *islessequal(x, y)* shall not raise the invalid floating-point exception when *x* and *y* are  
21423 unordered.

21424 **RETURN VALUE**

21425 Upon successful completion, the *islessequal()* macro shall return the value of  $(x) \leq (y)$ .

21426 If *x* or *y* is NaN, 0 shall be returned.

21427 **ERRORS**

21428 No errors are defined.

21429 **EXAMPLES**

21430 None.

21431 **APPLICATION USAGE**

21432 The relational and equality operators support the usual mathematical relationships between  
21433 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
21434 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
21435 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
21436 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
21437 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
21438 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
21439 indicates that the argument shall be an expression of **real-floating** type.

21440 **RATIONALE**

21441 None.

21442 **FUTURE DIRECTIONS**

21443 None.

21444 **SEE ALSO**

21445 *isgreater()*, *isgreaterequal()*, *isless()*, *islessgreater()*, *isunordered()*, the Base Definitions volume of  
21446 IEEE Std. 1003.1-200x <math.h>

21447 **CHANGE HISTORY**

21448 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21449 **NAME**

21450 islessgreater — test if  $x$  is less than or greater than  $y$

21451 **SYNOPSIS**

21452 #include <math.h>

21453 int islessgreater(real-floating  $x$ , real-floating  $y$ );

21454 **DESCRIPTION**

21455 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
21456 conflict between the requirements described here and the ISO C standard is unintentional. This  
21457 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21458 The *islessgreater()* macro shall determine whether its first argument is less than or greater than  
21459 its second argument. The *islessgreater*( $x$ ,  $y$ ) macro is similar to  $(x) < (y) \ || \ (x) > (y)$ ; however,  
21460 *islessgreater*( $x$ ,  $y$ ) shall not raise the invalid floating-point exception when  $x$  and  $y$  are unordered  
21461 (nor shall it evaluate  $x$  and  $y$  twice).

21462 **RETURN VALUE**

21463 Upon successful completion, the *islessgreater()* macro shall return the value of  
21464  $(x) < (y) \ || \ (x) > (y)$ .

21465 If  $x$  or  $y$  is NaN, 0 shall be returned.

21466 **ERRORS**

21467 No errors are defined.

21468 **EXAMPLES**

21469 None.

21470 **APPLICATION USAGE**

21471 The relational and equality operators support the usual mathematical relationships between  
21472 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
21473 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
21474 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
21475 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
21476 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
21477 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
21478 indicates that the argument shall be an expression of **real-floating** type.

21479 **RATIONALE**

21480 None.

21481 **FUTURE DIRECTIONS**

21482 None.

21483 **SEE ALSO**

21484 *isgreater()*, *isgreaterequal()*, *isless()*, *islessequal()*, *isunordered()*, the Base Definitions volume of  
21485 IEEE Std. 1003.1-200x <math.h>

21486 **CHANGE HISTORY**

21487 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.



21488 **NAME**

21489           islower — test for a lowercase letter

21490 **SYNOPSIS**

21491           #include &lt;ctype.h&gt;

21492           int islower(int c);

21493 **DESCRIPTION**

21494 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21495 conflict between the requirements described here and the ISO C standard is unintentional. This  
21496 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21497       The *islower()* function shall test whether *c* is a character of class **lower** in the program's current  
21498 locale; see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale.

21499       In all cases *c* is an **int**, the value of which the application shall ensure is a character representable  
21500 as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value,  
21501 the behavior is undefined.

21502 **RETURN VALUE**21503       The *islower()* function shall return non-zero if *c* is a lowercase letter; otherwise, it shall return 0.21504 **ERRORS**

21505       No errors are defined.

21506 **EXAMPLES**21507           **Testing for a Lowercase Letter**

21508       The following example tests whether the value is a lowercase letter, based on the locale of the  
21509 user, then uses it as part of a key value.

```
21510 #include <ctype.h>
21511 #include <stdlib.h>
21512 #include <locale.h>
21513 ...
21514 char *keyst;
21515 int elementlen, len;
21516 char c;
21517 ...
21518 setlocale(LC_ALL, "");
21519 ...
21520 len = 0;
21521 while (len < elementlen) {
21522 c = (char) (rand() % 256);
21523 ...
21524 if (islower(c))
21525 keyst[len++] = c;
21526 }
21527 ...
```

21528 **APPLICATION USAGE**

21529       To ensure applications portability, especially across natural languages, only this function and  
21530 those listed in the SEE ALSO section should be used for character classification.

21531 **RATIONALE**

21532 None.

21533 **FUTURE DIRECTIONS**

21534 None.

21535 **SEE ALSO**

21536 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
21537 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>, the Base  
21538 Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

21539 **CHANGE HISTORY**

21540 First released in Issue 1. Derived from Issue 1 of the SVID.

21541 **Issue 4**

21542 The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no  
21543 functional differences between this issue and Issue 3. Operation in the C locale is no longer  
21544 described explicitly on this reference page.

21545 **Issue 6**

21546 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21547 **NAME**21548            **isnan** — test for a NaN21549 **SYNOPSIS**

21550            #include &lt;math.h&gt;

21551            int isnan(real-floating x);

21552 **DESCRIPTION**

21553 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
21554            conflict between the requirements described here and the ISO C standard is unintentional. This  
21555            volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21556            The *isnan()* macro shall determine whether its argument value is a NaN. First, an argument  
21557            represented in a format wider than its semantic type is converted to its semantic type. Then  
21558            determination is based on the type of the argument.

21559 **RETURN VALUE**21560            The *isnan()* macro shall return a non-zero value if and only if its argument has a NaN value.21561 **ERRORS**

21562            No errors are defined.

21563 **EXAMPLES**

21564            None.

21565 **APPLICATION USAGE**

21566            None.

21567 **RATIONALE**

21568            None.

21569 **FUTURE DIRECTIONS**

21570            None.

21571 **SEE ALSO**

21572            *fpclassify()*, *isfinite()*, *isinf()*, *isnormal()*, *signbit()*, the Base Definitions volume of  
21573            IEEE Std. 1003.1-200x, <math.h>

21574 **CHANGE HISTORY**

21575            First released in Issue 3.

21576 **Issue 4**

21577            The words “not supporting NaN” are added to the APPLICATION USAGE section.

21578 **Issue 5**

21579            The DESCRIPTION is updated to indicate the return value when NaN is not supported. This  
21580            text was previously published in the APPLICATION USAGE section.

21581 **Issue 6**

21582            Entry re-written for alignment with the ISO/IEC 9899:1999 standard.

21583 **NAME**

21584           isnormal — test for a normal value

21585 **SYNOPSIS**

21586           #include <math.h>

21587           int isnormal(real-floating x);

21588 **DESCRIPTION**

21589 **cx**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21590       conflict between the requirements described here and the ISO C standard is unintentional. This  
21591       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21592       The *isnormal()* macro shall determine whether its argument value is normal (neither zero,  
21593       subnormal, infinite, nor NaN). First, an argument represented in a format wider than its  
21594       semantic type is converted to its semantic type. Then determination is based on the type of the  
21595       argument.

21596 **RETURN VALUE**

21597       The *isnormal()* macro shall return a non-zero value if and only if its argument has a normal  
21598       value.

21599 **ERRORS**

21600       No errors are defined.

21601 **EXAMPLES**

21602       None.

21603 **APPLICATION USAGE**

21604       None.

21605 **RATIONALE**

21606       None.

21607 **FUTURE DIRECTIONS**

21608       None.

21609 **SEE ALSO**

21610       *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *signbit()*, the Base Definitions volume of  
21611       IEEE Std. 1003.1-200x, <math.h>

21612 **CHANGE HISTORY**

21613       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21614 **NAME**

21615           isprint — test for a printing character

21616 **SYNOPSIS**

21617           #include <ctype.h>

21618           int isprint(int c);

21619 **DESCRIPTION**

21620 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21621       conflict between the requirements described here and the ISO C standard is unintentional. This  
21622       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21623       The *isprint()* function shall test whether *c* is a character of class **print** in the program's current  
21624       locale; see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale.

21625       In all cases *c* is an **int**, the value of which the application shall ensure is a character representable  
21626       as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value,  
21627       the behavior is undefined.

21628 **RETURN VALUE**

21629       The *isprint()* function shall return non-zero if *c* is a printing character; otherwise, it shall return 0.

21630 **ERRORS**

21631       No errors are defined.

21632 **EXAMPLES**

21633       None.

21634 **APPLICATION USAGE**

21635       To ensure applications portability, especially across natural languages, only this function and  
21636       those listed in the SEE ALSO section should be used for character classification.

21637 **RATIONALE**

21638       None.

21639 **FUTURE DIRECTIONS**

21640       None.

21641 **SEE ALSO**

21642       *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *ispunct()*, *isspace()*, *isupper()*,  
21643       *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>, the Base  
21644       Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

21645 **CHANGE HISTORY**

21646       First released in Issue 1. Derived from Issue 1 of the SVID.

21647 **Issue 4**

21648       The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no  
21649       functional differences between this issue and Issue 3. Operation in the C locale is no longer  
21650       described explicitly on this reference page.

21651 **Issue 6**

21652       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21653 **NAME**

21654           ispunct — test for a punctuation character

21655 **SYNOPSIS**

21656           #include <ctype.h>

21657           int ispunct(int c);

21658 **DESCRIPTION**

21659 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21660 conflict between the requirements described here and the ISO C standard is unintentional. This  
21661 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21662       The *ispunct()* function shall test whether *c* is a character of class **punct** in the program's current  
21663 locale; see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale.

21664       In all cases *c* is an **int**, the value of which the application shall ensure is a character representable  
21665 as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value,  
21666 the behavior is undefined.

21667 **RETURN VALUE**

21668       The *ispunct()* function shall return non-zero if *c* is a punctuation character; otherwise, it shall  
21669 return 0.

21670 **ERRORS**

21671       No errors are defined.

21672 **EXAMPLES**

21673       None.

21674 **APPLICATION USAGE**

21675       To ensure applications portability, especially across natural languages, only this function and  
21676 those listed in the SEE ALSO section should be used for character classification.

21677 **RATIONALE**

21678       None.

21679 **FUTURE DIRECTIONS**

21680       None.

21681 **SEE ALSO**

21682       *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *isspace()*, *isupper()*, *isxdigit()*,  
21683 *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>, the Base Definitions  
21684 volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

21685 **CHANGE HISTORY**

21686       First released in Issue 1. Derived from Issue 1 of the SVID.

21687 **Issue 4**

21688       The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no  
21689 functional differences between this issue and Issue 3. Operation in the C locale is no longer  
21690 described explicitly on this reference page.

21691 **Issue 6**

21692       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21693 **NAME**

21694           isspace — test for a white-space character

21695 **SYNOPSIS**

21696           #include <ctype.h>

21697           int isspace(int c);

21698 **DESCRIPTION**

21699 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21700 conflict between the requirements described here and the ISO C standard is unintentional. This  
21701 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21702       The *isspace()* function shall test whether *c* is a character of class **space** in the program's current  
21703 locale; see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale.

21704       In all cases *c* is an **int**, the value of which the application shall ensure is a character representable  
21705 as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value,  
21706 the behavior is undefined.

21707 **RETURN VALUE**

21708       The *isspace()* function shall return non-zero if *c* is a white-space character; otherwise, it shall  
21709 return 0.

21710 **ERRORS**

21711       No errors are defined.

21712 **EXAMPLES**

21713       None.

21714 **APPLICATION USAGE**

21715       To ensure applications portability, especially across natural languages, only this function and  
21716 those listed in the SEE ALSO section should be used for character classification.

21717 **RATIONALE**

21718       None.

21719 **FUTURE DIRECTIONS**

21720       None.

21721 **SEE ALSO**

21722       *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isupper()*,  
21723 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>, the Base  
21724 Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

21725 **CHANGE HISTORY**

21726       First released in Issue 1. Derived from Issue 1 of the SVID.

21727 **Issue 4**

21728       The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no  
21729 functional differences between this issue and Issue 3. Operation in the C locale is no longer  
21730 described explicitly on this reference page.

21731 **Issue 6**

21732       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21733 **NAME**

21734 isunordered — test if arguments are unordered

21735 **SYNOPSIS**

21736 #include <math.h>

21737 int isunordered(real-floating x, real-floating y);

21738 **DESCRIPTION**

21739 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
21740 conflict between the requirements described here and the ISO C standard is unintentional. This  
21741 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21742 The *isunordered()* macro shall determine whether its arguments are unordered.

21743 **RETURN VALUE**

21744 Upon successful completion, the *isunordered()* macro shall return 1 if its arguments are  
21745 unordered, and 0 otherwise.

21746 If *x* or *y* is NaN, 0 shall be returned.

21747 **ERRORS**

21748 No errors are defined.

21749 **EXAMPLES**

21750 None.

21751 **APPLICATION USAGE**

21752 The relational and equality operators support the usual mathematical relationships between  
21753 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
21754 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
21755 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
21756 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
21757 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
21758 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
21759 indicates that the argument shall be an expression of **real-floating** type.

21760 **RATIONALE**

21761 None.

21762 **FUTURE DIRECTIONS**

21763 None.

21764 **SEE ALSO**

21765 *isgreater()*, *isgreaterequal()*, *isless()*, *islessequal()*, *islessgreater()*, the Base Definitions volume of  
21766 IEEE Std. 1003.1-200x, <math.h>

21767 **CHANGE HISTORY**

21768 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.



21769 **NAME**

21770 isupper — test for an uppercase letter

21771 **SYNOPSIS**

21772 #include <ctype.h>

21773 int isupper(int c);

21774 **DESCRIPTION**

21775 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
21776 conflict between the requirements described here and the ISO C standard is unintentional. This  
21777 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21778 The *isupper()* function shall test whether *c* is a character of class **upper** in the program's current  
21779 locale; see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale.

21780 In all cases *c* is an **int**, the value of which the application shall ensure is a character representable  
21781 as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value,  
21782 the behavior is undefined.

21783 **RETURN VALUE**

21784 The *isupper()* function shall return non-zero if *c* is an uppercase letter; otherwise, it shall return 0.

21785 **ERRORS**

21786 No errors are defined.

21787 **EXAMPLES**

21788 None.

21789 **APPLICATION USAGE**

21790 To ensure applications portability, especially across natural languages, only this function and  
21791 those listed in the SEE ALSO section should be used for character classification.

21792 **RATIONALE**

21793 None.

21794 **FUTURE DIRECTIONS**

21795 None.

21796 **SEE ALSO**

21797 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isxdigit()*,  
21798 *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>, the Base Definitions  
21799 volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

21800 **CHANGE HISTORY**

21801 First released in Issue 1. Derived from Issue 1 of the SVID.

21802 **Issue 4**

21803 The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no  
21804 functional differences between this issue and Issue 3. Operation in the C locale is no longer  
21805 described explicitly on this reference page.

21806 **Issue 6**

21807 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21808 **NAME**

21809           iswalnum — test for an alphanumeric wide-character code

21810 **SYNOPSIS**

21811           #include <wctype.h>

21812           int iswalnum(wint\_t *wc*);

21813 **DESCRIPTION**

21814 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21815 conflict between the requirements described here and the ISO C standard is unintentional. This  
21816 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21817       The *iswalnum()* function shall test whether *wc* is a wide-character code representing a character  
21818 of class **alpha** or **digit** in the program's current locale; see the Base Definitions volume of  
21819 IEEE Std. 1003.1-200x, Chapter 7, Locale.

21820       In all cases *wc* is a **wint\_t**, the value of which the application shall ensure is a wide-character  
21821 code corresponding to a valid character in the current locale, or equal to the value of the macro  
21822 WEOF. If the argument has any other value, the behavior is undefined.

21823 **RETURN VALUE**

21824       The *iswalnum()* function shall return non-zero if *wc* is an alphanumeric wide-character code;  
21825 otherwise, it shall return 0.

21826 **ERRORS**

21827       No errors are defined.

21828 **EXAMPLES**

21829       None.

21830 **APPLICATION USAGE**

21831       To ensure applications portability, especially across natural languages, only this function and  
21832 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21833 **RATIONALE**

21834       None.

21835 **FUTURE DIRECTIONS**

21836       None.

21837 **SEE ALSO**

21838       *iswalphabet()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
21839 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
21840 IEEE Std. 1003.1-200x, <wctype.h>, <wchar.h>, <stdio.h>, the Base Definitions volume of  
21841 IEEE Std. 1003.1-200x, Chapter 7, Locale

21842 **CHANGE HISTORY**

21843       First released as a World-wide Portability Interface in Issue 4.

21844 **Issue 5**

21845       The following change has been made in this issue for alignment with  
21846 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21847       • The SYNOPSIS has been changed to indicate that this function and associated data types are  
21848       now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21849 **Issue 6**

21850

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21851 **NAME**

21852           iswalpha — test for an alphabetic wide-character code

21853 **SYNOPSIS**

21854           #include <wctype.h>

21855           int iswalpha(wint\_t *wc*);

21856 **DESCRIPTION**

21857 *cx*       The functionality described on this reference page is aligned with the ISO C standard. Any  
21858 conflict between the requirements described here and the ISO C standard is unintentional. This  
21859 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21860       The *iswalpha()* function shall test whether *wc* is a wide-character code representing a character of  
21861 class **alpha** in the program's current locale; see the Base Definitions volume of  
21862 IEEE Std. 1003.1-200x, Chapter 7, Locale.

21863       In all cases *wc* is a **wint\_t**, the value of which the application shall ensure is a wide-character  
21864 code corresponding to a valid character in the current locale, or equal to the value of the macro  
21865 WEOF. If the argument has any other value, the behavior is undefined.

21866 **RETURN VALUE**

21867       The *iswalpha()* function shall return non-zero if *wc* is an alphabetic wide-character code;  
21868 otherwise, it shall return 0.

21869 **ERRORS**

21870       No errors are defined.

21871 **EXAMPLES**

21872       None.

21873 **APPLICATION USAGE**

21874       To ensure applications portability, especially across natural languages, only this function and  
21875 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21876 **RATIONALE**

21877       None.

21878 **FUTURE DIRECTIONS**

21879       None.

21880 **SEE ALSO**

21881       *iswalnum()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
21882 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
21883 IEEE Std. 1003.1-200x, <**wctype.h**>, <**wchar.h**>, <**stdio.h**>, the Base Definitions volume of  
21884 IEEE Std. 1003.1-200x, Chapter 7, Locale

21885 **CHANGE HISTORY**

21886       First released in Issue 4.

21887 **Issue 5**

21888       The following change has been made in this issue for alignment with  
21889 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21890       • The SYNOPSIS has been changed to indicate that this function and associated data types are  
21891       now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21892 **Issue 6**

21893

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21894 **NAME**

21895           iswblank — test for a blank wide-character code

21896 **SYNOPSIS**

21897           #include <wctype.h>

21898           int iswblank(wint\_t *wc*);

21899 **DESCRIPTION**

21900 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21901 conflict between the requirements described here and the ISO C standard is unintentional. This  
21902 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21903       The *iswblank()* function shall test whether *wc* is a wide-character code representing a character of  
21904 class **blank** in the program's current locale; see the Base Definitions volume of  
21905 IEEE Std. 1003.1-200x, Chapter 7, Locale. In all cases, *wc* is a **wint\_t**, the value of which the  
21906 application shall ensure is a wide-character code corresponding to a valid character in the  
21907 current locale, or equal to the value of the macro WEOF. If the argument has any other value, the  
21908 behavior is undefined.

21909 **RETURN VALUE**

21910       The *iswblank()* function shall return non-zero if *wc* is a <blank> wide-character code; otherwise,  
21911 it shall return 0.

21912 **ERRORS**

21913       No errors are defined.

21914 **EXAMPLES**

21915       None.

21916 **APPLICATION USAGE**

21917       To ensure applications portability, especially across natural languages, only this function and  
21918 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21919 **RATIONALE**

21920       None.

21921 **FUTURE DIRECTIONS**

21922       None.

21923 **SEE ALSO**

21924       *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
21925 *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
21926 IEEE Std. 1003.1-200x, <wchar.h>, <wctype.h>, <stdio.h>

21927 **CHANGE HISTORY**

21928       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21929 **NAME**

21930 iswcntrl — test for a control wide-character code

21931 **SYNOPSIS**

21932 #include &lt;wctype.h&gt;

21933 int iswcntrl(wint\_t *wc*);21934 **DESCRIPTION**

21935 *cx* The functionality described on this reference page is aligned with the ISO C standard. Any  
 21936 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21937 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21938 The *iswcntrl()* function shall test whether *wc* is a wide-character code representing a character of  
 21939 class **control** in the program's current locale; see the Base Definitions volume of  
 21940 IEEE Std. 1003.1-200x, Chapter 7, Locale.

21941 In all cases *wc* is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 21942 code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21943 WEOF. If the argument has any other value, the behavior is undefined.

21944 **RETURN VALUE**

21945 The *iswcntrl()* function shall return non-zero if *wc* is a control wide-character code; otherwise, it  
 21946 shall return 0.

21947 **ERRORS**

21948 No errors are defined.

21949 **EXAMPLES**

21950 None.

21951 **APPLICATION USAGE**

21952 To ensure applications portability, especially across natural languages, only this function and  
 21953 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21954 **RATIONALE**

21955 None.

21956 **FUTURE DIRECTIONS**

21957 None.

21958 **SEE ALSO**

21959 *iswalnum()*, *iswalpha()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
 21960 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 21961 IEEE Std. 1003.1-200x, <wctype.h>, <wchar.h>, the Base Definitions volume of  
 21962 IEEE Std. 1003.1-200x, Chapter 7, Locale

21963 **CHANGE HISTORY**

21964 First released in Issue 4.

21965 **Issue 5**

21966 The following change has been made in this issue for alignment with  
 21967 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21968 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21969 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21970 **Issue 6**

21971

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.



21972 **NAME**

21973 iswctype — test character for a specified class

21974 **SYNOPSIS**

21975 #include &lt;wctype.h&gt;

21976 int iswctype(wint\_t *wc*, wctype\_t *charclass*);21977 **DESCRIPTION**

21978 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 21979 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21980 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

21981 The *iswctype()* function shall determine whether the wide-character code *wc* has the character  
 21982 class *charclass*, returning true or false. The *iswctype()* function is defined on WEOF and wide-  
 21983 character codes corresponding to the valid character encodings in the current locale. If the *wc*  
 21984 argument is not in the domain of the function, the result is undefined. If the value of *charclass* is  
 21985 invalid (that is, not obtained by a call to *wctype()* or *charclass* is invalidated by a subsequent call  
 21986 to *setlocale()* that has affected category *LC\_CTYPE*) the result is unspecified.

21987 **RETURN VALUE**

21988 The *iswctype()* function shall return non-zero (true) if and only if *wc* has the property described  
 21989 **CX** by *charclass*. If *charclass* is 0, *iswctype()* shall return 0.

21990 **ERRORS**

21991 No errors are defined.

21992 **EXAMPLES**21993 **Testing for a Valid Character**

```
21994 #include <wctype.h>
21995 ...
21996 int yes_or_no;
21997 wint_t wc;
21998 wctype_t valid_class;
21999 ...
22000 if ((valid_class=wctype("vowel")) == (wctype_t)0)
22001 /* Invalid character class. */
22002 yes_or_no=iswctype(wc,valid_class);
```

22003 **APPLICATION USAGE**

22004 The twelve strings "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower",  
 22005 "print", "punct", "space", "upper", and "xdigit" are reserved for the standard  
 22006 character classes. In the table below, the functions in the left column are equivalent to the  
 22007 functions in the right column.

|       |                     |                                      |
|-------|---------------------|--------------------------------------|
| 22008 | <i>iswalnum(wc)</i> | <i>iswctype(wc, wctype("alnum"))</i> |
| 22009 | <i>iswalpha(wc)</i> | <i>iswctype(wc, wctype("alpha"))</i> |
| 22010 | <i>iswblank(wc)</i> | <i>iswctype(wc, wctype("blank"))</i> |
| 22011 | <i>iswcntrl(wc)</i> | <i>iswctype(wc, wctype("cntrl"))</i> |
| 22012 | <i>iswdigit(wc)</i> | <i>iswctype(wc, wctype("digit"))</i> |
| 22013 | <i>iswgraph(wc)</i> | <i>iswctype(wc, wctype("graph"))</i> |
| 22014 | <i>iswlower(wc)</i> | <i>iswctype(wc, wctype("lower"))</i> |
| 22015 | <i>iswprint(wc)</i> | <i>iswctype(wc, wctype("print"))</i> |
| 22016 | <i>iswpunct(wc)</i> | <i>iswctype(wc, wctype("punct"))</i> |
| 22017 | <i>iswspace(wc)</i> | <i>iswctype(wc, wctype("space"))</i> |

22018            `iswupper(wc)`      `iswctype(wc, wctype("upper"))`  
22019            `iswxdigit(wc)`      `iswctype(wc, wctype("xdigit"))`

22020 **RATIONALE**

22021            None.

22022 **FUTURE DIRECTIONS**

22023            None.

22024 **SEE ALSO**

22025            `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`,  
22026            `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `wctype()`, the Base Definitions volume of  
22027            IEEE Std. 1003.1-200x, `<wctype.h>`, `<wchar.h>`

22028 **CHANGE HISTORY**

22029            First released as World-wide Portability Interfaces in Issue 4.

22030 **Issue 5**

22031            The following change has been made in this issue for alignment with  
22032            ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 22033            • The SYNOPSIS has been changed to indicate that this function and associated data types are  
22034            now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

22035 **NAME**

22036 iswdigit — test for a decimal digit wide-character code

22037 **SYNOPSIS**

22038 #include &lt;wctype.h&gt;

22039 int iswdigit(wint\_t *wc*);22040 **DESCRIPTION**

22041 *cx* The functionality described on this reference page is aligned with the ISO C standard. Any  
 22042 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22043 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

22044 The *iswdigit()* function shall test whether *wc* is a wide-character code representing a character of  
 22045 class **digit** in the program's current locale; see the Base Definitions volume of  
 22046 IEEE Std. 1003.1-200x, Chapter 7, Locale.

22047 In all cases *wc* is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 22048 code corresponding to a valid character in the current locale, or equal to the value of the macro  
 22049 WEOF. If the argument has any other value, the behavior is undefined.

22050 **RETURN VALUE**

22051 The *iswdigit()* function shall return non-zero if *wc* is a decimal digit wide-character code;  
 22052 otherwise, it shall return 0.

22053 **ERRORS**

22054 No errors are defined.

22055 **EXAMPLES**

22056 None.

22057 **APPLICATION USAGE**

22058 To ensure applications portability, especially across natural languages, only this function and  
 22059 those listed in the SEE ALSO section should be used for classification of wide-character codes.

22060 **RATIONALE**

22061 None.

22062 **FUTURE DIRECTIONS**

22063 None.

22064 **SEE ALSO**

22065 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
 22066 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 22067 IEEE Std. 1003.1-200x, <wctype.h>, <wchar.h>

22068 **CHANGE HISTORY**

22069 First released in Issue 4.

22070 **Issue 5**

22071 The following change has been made in this issue for alignment with  
 22072 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 22073 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 22074 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

22075 **Issue 6**

22076 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22077 **NAME**

22078 iswgraph — test for a visible wide-character code

22079 **SYNOPSIS**

22080 #include <wctype.h>

22081 int iswgraph(wint\_t wc);

22082 **DESCRIPTION**

22083 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
22084 conflict between the requirements described here and the ISO C standard is unintentional. This  
22085 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

22086 The *iswgraph()* function shall test whether *wc* is a wide-character code representing a character  
22087 of class **graph** in the program's current locale; see the Base Definitions volume of  
22088 IEEE Std. 1003.1-200x, Chapter 7, Locale.

22089 In all cases *wc* is a **wint\_t**, the value of which the application shall ensure is a wide-character  
22090 code corresponding to a valid character in the current locale, or equal to the value of the macro  
22091 WEOF. If the argument has any other value, the behavior is undefined.

22092 **RETURN VALUE**

22093 The *iswgraph()* function shall return non-zero if *wc* is a wide-character code with a visible  
22094 representation; otherwise, it shall return 0.

22095 **ERRORS**

22096 No errors are defined.

22097 **EXAMPLES**

22098 None.

22099 **APPLICATION USAGE**

22100 To ensure applications portability, especially across natural languages, only this function and  
22101 those listed in the SEE ALSO section should be used for classification of wide-character codes.

22102 **RATIONALE**

22103 None.

22104 **FUTURE DIRECTIONS**

22105 None.

22106 **SEE ALSO**

22107 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
22108 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
22109 IEEE Std. 1003.1-200x, <wctype.h>, <wchar.h>, the Base Definitions volume of  
22110 IEEE Std. 1003.1-200x, Chapter 7, Locale

22111 **CHANGE HISTORY**

22112 First released in Issue 4.

22113 **Issue 5**

22114 The following change has been made in this issue for alignment with  
22115 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 22116 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
22117 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

22118 **Issue 6**

22119

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22120 **NAME**

22121 iswlower — test for a lowercase letter wide-character code

22122 **SYNOPSIS**

22123 #include <wctype.h>

22124 int iswlower(wint\_t wc);

22125 **DESCRIPTION**

22126 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
22127 conflict between the requirements described here and the ISO C standard is unintentional. This  
22128 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

22129 The *iswlower()* function shall test whether *wc* is a wide-character code representing a character  
22130 of class **lower** in the program's current locale; see the Base Definitions volume of  
22131 IEEE Std. 1003.1-200x, Chapter 7, Locale.

22132 In all cases *wc* is a **wint\_t**, the value of which the application shall ensure is a wide-character  
22133 code corresponding to a valid character in the current locale, or equal to the value of the macro  
22134 WEOF. If the argument has any other value, the behavior is undefined.

22135 **RETURN VALUE**

22136 The *iswlower()* function shall return non-zero if *wc* is a lowercase letter wide-character code;  
22137 otherwise, it shall return 0.

22138 **ERRORS**

22139 No errors are defined.

22140 **EXAMPLES**

22141 None.

22142 **APPLICATION USAGE**

22143 To ensure applications portability, especially across natural languages, only this function and  
22144 those listed in the SEE ALSO section should be used for classification of wide-character codes.

22145 **RATIONALE**

22146 None.

22147 **FUTURE DIRECTIONS**

22148 None.

22149 **SEE ALSO**

22150 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswprint()*, *iswpunct()*,  
22151 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
22152 IEEE Std. 1003.1-200x, <wctype.h>, <wchar.h>, the Base Definitions volume of  
22153 IEEE Std. 1003.1-200x, Chapter 7, Locale

22154 **CHANGE HISTORY**

22155 First released in Issue 4.

22156 **Issue 5**

22157 The following change has been made in this issue for alignment with  
22158 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 22159 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
22160 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

22161 **Issue 6**

22162

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22163 **NAME**

22164           iswprint — test for a printing wide-character code

22165 **SYNOPSIS**

22166           #include <wctype.h>

22167           int iswprint(wint\_t *wc*);

22168 **DESCRIPTION**

22169 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
22170 conflict between the requirements described here and the ISO C standard is unintentional. This  
22171 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

22172       The *iswprint()* function shall test whether *wc* is a wide-character code representing a character of  
22173 class **print** in the program's current locale; see the Base Definitions volume of  
22174 IEEE Std. 1003.1-200x, Chapter 7, Locale.

22175       In all cases *wc* is a **wint\_t**, the value of which the application shall ensure is a wide-character  
22176 code corresponding to a valid character in the current locale, or equal to the value of the macro  
22177 WEOF. If the argument has any other value, the behavior is undefined.

22178 **RETURN VALUE**

22179       The *iswprint()* function shall return non-zero if *wc* is a printing wide-character code; otherwise, it  
22180 shall return 0.

22181 **ERRORS**

22182       No errors are defined.

22183 **EXAMPLES**

22184       None.

22185 **APPLICATION USAGE**

22186       To ensure applications portability, especially across natural languages, only this function and  
22187 those listed in the SEE ALSO section should be used for classification of wide-character codes.

22188 **RATIONALE**

22189       None.

22190 **FUTURE DIRECTIONS**

22191       None.

22192 **SEE ALSO**

22193       *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswpunct()*,  
22194 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
22195 IEEE Std. 1003.1-200x, <wctype.h>, <wchar.h>, the Base Definitions volume of  
22196 IEEE Std. 1003.1-200x, Chapter 7, Locale

22197 **CHANGE HISTORY**

22198       First released in Issue 4.

22199 **Issue 5**

22200       The following change has been made in this issue for alignment with  
22201 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 22202       • The SYNOPSIS has been changed to indicate that this function and associated data types are  
22203       now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.



22204 **Issue 6**

22205

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

## 22206 NAME

22207 iswpunct — test for a punctuation wide-character code

## 22208 SYNOPSIS

22209 #include <wctype.h>

22210 int iswpunct(wint\_t *wc*);

## 22211 DESCRIPTION

22212 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
22213 conflict between the requirements described here and the ISO C standard is unintentional. This  
22214 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

22215 The *iswpunct()* function shall test whether *wc* is a wide-character code representing a character  
22216 of class **punct** in the program's current locale; see the Base Definitions volume of  
22217 IEEE Std. 1003.1-200x, Chapter 7, Locale.

22218 In all cases *wc* is a **wint\_t**, the value of which the application shall ensure is a wide-character  
22219 code corresponding to a valid character in the current locale, or equal to the value of the macro  
22220 WEOF. If the argument has any other value, the behavior is undefined.

## 22221 RETURN VALUE

22222 The *iswpunct()* function shall return non-zero if *wc* is a punctuation wide-character code;  
22223 otherwise, it shall return 0.

## 22224 ERRORS

22225 No errors are defined.

## 22226 EXAMPLES

22227 None.

## 22228 APPLICATION USAGE

22229 To ensure applications portability, especially across natural languages, only this function and  
22230 those listed in the SEE ALSO section should be used for classification of wide-character codes.

## 22231 RATIONALE

22232 None.

## 22233 FUTURE DIRECTIONS

22234 None.

## 22235 SEE ALSO

22236 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
22237 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
22238 IEEE Std. 1003.1-200x, <wctype.h>, <wchar.h>, the Base Definitions volume of  
22239 IEEE Std. 1003.1-200x, Chapter 7, Locale

## 22240 CHANGE HISTORY

22241 First released in Issue 4.

## 22242 Issue 5

22243 The following change has been made in this issue for alignment with  
22244 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 22245 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
22246 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

22247 **Issue 6**

22248

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22249 **NAME**

22250 iswspace — test for a white-space wide-character code

22251 **SYNOPSIS**

22252 #include &lt;wctype.h&gt;

22253 int iswspace(wint\_t wc);

22254 **DESCRIPTION**

22255 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
22256 conflict between the requirements described here and the ISO C standard is unintentional. This  
22257 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

22258 The *iswspace()* function shall test whether *wc* is a wide-character code representing a character of  
22259 class **space** in the program's current locale; see the Base Definitions volume of  
22260 IEEE Std. 1003.1-200x, Chapter 7, Locale.

22261 In all cases *wc* is a **wint\_t**, the value of which the application shall ensure is a wide-character  
22262 code corresponding to a valid character in the current locale, or equal to the value of the macro  
22263 WEOF. If the argument has any other value, the behavior is undefined.

22264 **RETURN VALUE**

22265 The *iswspace()* function shall return non-zero if *wc* is a white-space wide-character code;  
22266 otherwise, it shall return 0.

22267 **ERRORS**

22268 No errors are defined.

22269 **EXAMPLES**

22270 None.

22271 **APPLICATION USAGE**

22272 To ensure applications portability, especially across natural languages, only this function and  
22273 those listed in the SEE ALSO section should be used for classification of wide-character codes.

22274 **RATIONALE**

22275 None.

22276 **FUTURE DIRECTIONS**

22277 None.

22278 **SEE ALSO**

22279 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
22280 *iswpunct()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
22281 IEEE Std. 1003.1-200x, <wctype.h>, <wchar.h>, the Base Definitions volume of  
22282 IEEE Std. 1003.1-200x, Chapter 7, Locale

22283 **CHANGE HISTORY**

22284 First released in Issue 4.

22285 **Issue 5**

22286 The following change has been made in this issue for alignment with  
22287 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 22288 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
22289 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

22290 **Issue 6**

22291

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22292 **NAME**

22293 iswupper — test for an uppercase letter wide-character code

22294 **SYNOPSIS**

22295 #include <wctype.h>

22296 int iswupper(wint\_t wc);

22297 **DESCRIPTION**

22298 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
22299 conflict between the requirements described here and the ISO C standard is unintentional. This  
22300 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

22301 The *iswupper()* function shall test whether *wc* is a wide-character code representing a character  
22302 of class **upper** in the program's current locale; see the Base Definitions volume of  
22303 IEEE Std. 1003.1-200x, Chapter 7, Locale.

22304 In all cases *wc* is a **wint\_t**, the value of which the application shall ensure is a wide-character  
22305 code corresponding to a valid character in the current locale, or equal to the value of the macro  
22306 WEOF. If the argument has any other value, the behavior is undefined.

22307 **RETURN VALUE**

22308 The *iswupper()* function shall return non-zero if *wc* is an uppercase letter wide-character code;  
22309 otherwise, it shall return 0.

22310 **ERRORS**

22311 No errors are defined.

22312 **EXAMPLES**

22313 None.

22314 **APPLICATION USAGE**

22315 To ensure applications portability, especially across natural languages, only this function and  
22316 those listed in the SEE ALSO section should be used for classification of wide-character codes.

22317 **RATIONALE**

22318 None.

22319 **FUTURE DIRECTIONS**

22320 None.

22321 **SEE ALSO**

22322 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
22323 *iswpunct()*, *iswspace()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
22324 IEEE Std. 1003.1-200x, <wctype.h>, <wchar.h>, the Base Definitions volume of  
22325 IEEE Std. 1003.1-200x, Chapter 7, Locale

22326 **CHANGE HISTORY**

22327 First released in Issue 4.

22328 **Issue 5**

22329 The following change has been made in this issue for alignment with  
22330 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 22331 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
22332 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

22333 **Issue 6**

22334

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22335 **NAME**

22336 iswxdigit — test for a hexadecimal digit wide-character code

22337 **SYNOPSIS**

22338 #include <wctype.h>

22339 int iswxdigit(wint\_t wc);

22340 **DESCRIPTION**

22341 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
22342 conflict between the requirements described here and the ISO C standard is unintentional. This  
22343 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

22344 The *iswxdigit()* function shall test whether *wc* is a wide-character code representing a character  
22345 of class **xdigit** in the program's current locale; see the Base Definitions volume of  
22346 IEEE Std. 1003.1-200x, Chapter 7, Locale.

22347 In all cases *wc* is a **wint\_t**, the value of which the application shall ensure is a wide-character  
22348 code corresponding to a valid character in the current locale, or equal to the value of the macro  
22349 WEOF. If the argument has any other value, the behavior is undefined.

22350 **RETURN VALUE**

22351 The *iswxdigit()* function shall return non-zero if *wc* is a hexadecimal digit wide-character code;  
22352 otherwise, it shall return 0.

22353 **ERRORS**

22354 No errors are defined.

22355 **EXAMPLES**

22356 None.

22357 **APPLICATION USAGE**

22358 To ensure applications portability, especially across natural languages, only this function and  
22359 those listed in the SEE ALSO section should be used for classification of wide-character codes.

22360 **RATIONALE**

22361 None.

22362 **FUTURE DIRECTIONS**

22363 None.

22364 **SEE ALSO**

22365 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
22366 *iswpunct()*, *iswspace()*, *iswupper()*, *setlocale()*, the Base Definitions volume of  
22367 IEEE Std. 1003.1-200x, <wctype.h>, <wchar.h>

22368 **CHANGE HISTORY**

22369 First released in Issue 4.

22370 **Issue 5**

22371 The following change has been made in this issue for alignment with  
22372 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 22373 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
22374 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

22375 **Issue 6**

22376 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.



22377 **NAME**

22378 isxdigit — test for a hexadecimal digit

22379 **SYNOPSIS**

22380 #include &lt;ctype.h&gt;

22381 int isxdigit(int c);

22382 **DESCRIPTION**

22383 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 22384 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22385 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

22386 The *isxdigit()* function shall test whether *c* is a character of class **xdigit** in the program's current  
 22387 locale; see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale.

22388 In all cases *c* is an **int**, the value of which the application shall ensure is a character representable  
 22389 as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value,  
 22390 the behavior is undefined.

22391 **RETURN VALUE**

22392 The *isxdigit()* function shall return non-zero if *c* is a hexadecimal digit; otherwise, it shall return  
 22393 0.

22394 **ERRORS**

22395 No errors are defined.

22396 **EXAMPLES**

22397 None.

22398 **APPLICATION USAGE**

22399 To ensure applications portability, especially across natural languages, only this function and  
 22400 those listed in the SEE ALSO section should be used for character classification.

22401 **RATIONALE**

22402 None.

22403 **FUTURE DIRECTIONS**

22404 None.

22405 **SEE ALSO**

22406 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
 22407 the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>

22408 **CHANGE HISTORY**

22409 First released in Issue 1. Derived from Issue 1 of the SVID.

22410 **Issue 4**

22411 The text of the DESCRIPTION is revised, although there are no functional differences between  
 22412 this issue and Issue 3.

22413 **Issue 6**

22414 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22415 **NAME**

22416 j0, j1, jn — Bessel functions of the first kind

22417 **SYNOPSIS**

```
22418 xSI #include <math.h>
22419 double j0(double x);
22420 double j1(double x);
22421 double jn(int n, double x);
22422
```

22423 **DESCRIPTION**

22424 The  $j0()$ ,  $j1()$ , and  $jn()$  functions shall compute Bessel functions of  $x$  of the first kind of orders 0,  
 22425 1, and  $n$  respectively.

22426 An application wishing to check for error situations should set *errno* to 0 before calling  $j0()$ ,  $j1()$ ,  
 22427 or  $jn()$ . If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

22428 **RETURN VALUE**

22429 Upon successful completion,  $j0()$ ,  $j1()$ , and  $jn()$  shall return the relevant Bessel value of  $x$  of the  
 22430 first kind.

22431 If the  $x$  argument is too large in magnitude, 0 shall be returned and *errno* may be set to  
 22432 [ERANGE].

22433 If  $x$  is NaN, NaN shall be returned and *errno* may be set to [EDOM].

22434 If the correct result would cause underflow, 0 shall be returned and *errno* may be set to  
 22435 [ERANGE].

22436 **ERRORS**

22437 The  $j0()$ ,  $j1()$ , and  $jn()$  functions may fail if:

22438 [EDOM] The value of  $x$  is NaN.

22439 [ERANGE] The value of  $x$  was too large in magnitude, or underflow occurred.

22440 No other errors shall occur.

22441 **EXAMPLES**

22442 None.

22443 **APPLICATION USAGE**

22444 None.

22445 **RATIONALE**

22446 None.

22447 **FUTURE DIRECTIONS**

22448 None.

22449 **SEE ALSO**

22450 *isnan()*, *y0()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

22451 **CHANGE HISTORY**

22452 First released in Issue 1. Derived from Issue 1 of the SVID.

22453 **Issue 4**

22454 References to *matherr()* are removed.

22455 The RETURN VALUE and ERRORS sections are substantially rewritten to rationalize error  
 22456 handling in the mathematics functions.

22457 **Issue 5**

22458

22459

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

22460 **NAME**

22461       jrand48 — generate a uniformly distributed pseudo-random long signed integer

22462 **SYNOPSIS**

22463 XSI       #include <stdlib.h>

22464       long jrand48(unsigned short xsubi[3]);

22465

22466 **DESCRIPTION**

22467       Refer to *drand48()*.

22468 **NAME**

22469 kill — send a signal to a process or a group of processes

22470 **SYNOPSIS**

22471 #include &lt;signal.h&gt;

22472 int kill(pid\_t pid, int sig);

22473 **DESCRIPTION**

22474 The *kill()* function shall send a signal to a process or a group of processes specified by *pid*. The  
 22475 signal to be sent is specified by *sig* and is either one from the list given in <**signal.h**> or 0. If *sig* is  
 22476 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can  
 22477 be used to check the validity of *pid*.

22478 For a process to have permission to send a signal to a process designated by *pid*, unless the  
 22479 sending process has appropriate privileges, the application shall ensure that the real or effective  
 22480 user ID of the sending process matches the real or saved set-user-ID of the receiving process.

22481 If *pid* is greater than 0, *sig* shall be sent to the process whose process ID is equal to *pid*.

22482 If *pid* is 0, *sig* shall be sent to all processes (excluding an unspecified set of system processes)  
 22483 whose process group ID is equal to the process group ID of the sender, and for which the  
 22484 process has permission to send a signal.

22485 If *pid* is -1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) for  
 22486 which the process has permission to send that signal.

22487 If *pid* is negative, but not -1, *sig* shall be sent to all processes (excluding an unspecified set of  
 22488 system processes) whose process group ID is equal to the absolute value of *pid*, and for which  
 22489 the process has permission to send a signal.

22490 If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not blocked for  
 22491 the calling thread and if no other thread has *sig* unblocked or is waiting in a *sigwait()* function  
 22492 for *sig*, either *sig* or at least one pending unblocked signal shall be delivered to the sending  
 22493 thread before *kill()* returns.

22494 The user ID tests described above shall not be applied when sending SIGCONT to a process that  
 22495 is a member of the same session as the sending process.

22496 An implementation that provides extended security controls may impose further  
 22497 implementation-defined restrictions on the sending of signals, including the null signal. In  
 22498 particular, the system may deny the existence of some or all of the processes specified by *pid*.

22499 The *kill()* function is successful if the process has permission to send *sig* to any of the processes  
 22500 specified by *pid*. If *kill()* fails, no signal shall be sent.

22501 **RETURN VALUE**

22502 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 22503 indicate the error.

22504 **ERRORS**

22505 The *kill()* function shall fail if:

22506 [EINVAL] The value of the *sig* argument is an invalid or unsupported signal number.

22507 [EPERM] The process does not have permission to send the signal to any receiving  
 22508 process.

22509 [ESRCH] No process or process group can be found corresponding to that specified by  
 22510 *pid*.

22511 **EXAMPLES**

22512 None.

22513 **APPLICATION USAGE**

22514 None.

22515 **RATIONALE**

22516 The semantics for permission checking for *kill()* differed between System V and most other  
22517 implementations, such as Version 7 or 4.3 BSD. The semantics chosen for this volume of  
22518 IEEE Std. 1003.1-200x agree with System V. Specifically, a set-user-ID process cannot protect  
22519 itself against signals (or at least not against SIGKILL) unless it changes its real user ID. This  
22520 choice allows the user who starts an application to send it signals even if it changes its effective  
22521 user ID. The other semantics give more power to an application that wants to protect itself from  
22522 the user who ran it.

22523 Some implementations provide semantic extensions to the *kill()* function when the absolute  
22524 value of *pid* is greater than some maximum, or otherwise special, value. Negative values are a  
22525 flag to *kill()*. Since most implementations return [ESRCH] in this case, this behavior is not  
22526 included in this volume of IEEE Std. 1003.1-200x, although a conforming implementation could  
22527 provide such an extension.

22528 The implementation-defined processes to which a signal cannot be sent may include the  
22529 scheduler or *init*.

22530 There was initially strong sentiment to specify that, if *pid* specifies that a signal be sent to the  
22531 calling process and that signal is not blocked, that signal would be delivered before *kill()*  
22532 returns. This would permit a process to call *kill()* and be guaranteed that the call never return.  
22533 However, historical implementations that provide only the *signal()* function make only the  
22534 weaker guarantee in this volume of IEEE Std. 1003.1-200x, because they only deliver one signal  
22535 each time a process enters the kernel. Modifications to such implementations to support the  
22536 *sigaction()* function generally require entry to the kernel following return from a signal-catching  
22537 function, in order to restore the signal mask. Such modifications have the effect of satisfying the  
22538 stronger requirement, at least when *sigaction()* is used, but not necessarily when *signal()* is used.  
22539 The developers of this volume of IEEE Std. 1003.1-200x considered making the stronger  
22540 requirement except when *signal()* is used, but felt this would be unnecessarily complex.  
22541 Implementors are encouraged to meet the stronger requirement whenever possible. In practice,  
22542 the weaker requirement is the same, except in the rare case when two signals arrive during a  
22543 very short window. This reasoning also applies to a similar requirement for *sigprocmask()*.

22544 In 4.2 BSD, the SIGCONT signal can be sent to any descendant process regardless of user-ID  
22545 security checks. This allows a job control shell to continue a job even if processes in the job have  
22546 altered their user IDs (as in the *su* command). In keeping with the addition of the concept of  
22547 sessions, similar functionality is provided by allowing the SIGCONT signal to be sent to any  
22548 process in the same session regardless of user ID security checks. This is less restrictive than BSD  
22549 in the sense that ancestor processes (in the same session) can now be the recipient. It is more  
22550 restrictive than BSD in the sense that descendant processes that form new sessions are now  
22551 subject to the user ID checks. A similar relaxation of security is not necessary for the other job  
22552 control signals since those signals are typically sent by the terminal driver in recognition of  
22553 special characters being typed; the terminal driver bypasses all security checks.

22554 In secure implementations, a process may be restricted from sending a signal to a process having  
22555 a different security label. In order to prevent the existence or nonexistence of a process from  
22556 being used as a covert channel, such processes should appear nonexistent to the sender; that is,  
22557 [ESRCH] should be returned, rather than [EPERM], if *pid* refers only to such processes.

22558 Existing implementations vary on the result of a *kill()* with *pid* indicating an inactive process (a  
 22559 terminated process that has not been waited for by its parent). Some indicate success on such a  
 22560 call (subject to permission checking), while others give an error of [ESRCH]. Since the definition  
 22561 of process lifetime in this volume of IEEE Std. 1003.1-200x covers inactive processes, the  
 22562 [ESRCH] error as described is inappropriate in this case. In particular, this means that an  
 22563 application cannot have a parent process check for termination of a particular child with *kill()*.  
 22564 (Usually this is done with the null signal; this can be done reliably with *waitpid()*.)

22565 There is some belief that the name *kill()* is misleading, since the function is not always intended  
 22566 to cause process termination. However, the name is common to all historical implementations,  
 22567 and any change would be in conflict with the goal of minimal changes to existing application  
 22568 code.

#### 22569 FUTURE DIRECTIONS

22570 None.

#### 22571 SEE ALSO

22572 *getpid()*, *raise()*, *setsid()*, *sigaction()*, *sigqueue()*, the Base Definitions volume of  
 22573 IEEE Std. 1003.1-200x, `<signal.h>`, `<sys/types.h>`

#### 22574 CHANGE HISTORY

22575 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 22576 Issue 4

22577 The `<sys/types.h>` header is now marked as optional (OH); this header need not be included on  
 22578 XSI-conformant systems.

22579 The DESCRIPTION is clarified in various places.

22580 The following change is incorporated for alignment with the FIPS requirements:

- 22581 • In the DESCRIPTION, the second paragraph is reworded to indicate that the saved set-user-  
 22582 ID of the calling process is checked in place of its effective user ID. This functionality is  
 22583 marked as an extension.

#### 22584 Issue 5

22585 The DESCRIPTION is updated for alignment with POSIX Threads Extension.

#### 22586 Issue 6

22587 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

22588 The following new requirements on POSIX implementations derive from alignment with the  
 22589 Single UNIX Specification:

- 22590 • In the DESCRIPTION, the second paragraph is reworded to indicate that the saved set-user-  
 22591 ID of the calling process is checked in place of its effective user ID. This is a FIPS  
 22592 requirement.
- 22593 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 22594 required for conforming implementations of previous POSIX specifications, it was not  
 22595 required for UNIX applications.
- 22596 • The behavior when *pid* is `-1` is now specified. It was previously explicitly unspecified in the  
 22597 POSIX.1-1988 standard.

22598 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22599 **NAME**

22600 killpg — send a signal to a process group

22601 **SYNOPSIS**

22602 XSI #include <signal.h>

22603 int killpg(pid\_t pgrp, int sig);

22604

22605 **DESCRIPTION**

22606 The *killpg()* function shall send the signal specified by *sig* to the process group specified by *pgrp*.

22607 If *pgrp* is greater than 1, *killpg(pgrp, sig)* shall be equivalent to *kill(-pgrp, sig)*. If *pgrp* is less than or  
22608 equal to 1, the behavior of *killpg()* is undefined.

22609 **RETURN VALUE**

22610 Refer to *kill()*.

22611 **ERRORS**

22612 Refer to *kill()*.

22613 **EXAMPLES**

22614 None.

22615 **APPLICATION USAGE**

22616 None.

22617 **RATIONALE**

22618 None.

22619 **FUTURE DIRECTIONS**

22620 None.

22621 **SEE ALSO**

22622 *getpgid()*, *getpid()*, *kill()*, *raise()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |  
22623 <signal.h>

22624 **CHANGE HISTORY**

22625 First released in Issue 4, Version 2.

22626 **Issue 5**

22627 Moved from X/OPEN UNIX extension to BASE.



22628 **NAME**

22629       l64a — convert a 32-bit integer to a radix-64 ASCII string

22630 **SYNOPSIS**

22631 xSI     #include &lt;stdlib.h&gt;

22632       char \*l64a(long value);

22633

22634 **DESCRIPTION**22635       Refer to *a64l()*.

22636 **NAME**

22637 labs, llabs — return a long integer absolute value

22638 **SYNOPSIS**

22639 #include <stdlib.h>

22640 long labs(long *i*);

22641 long long llabs(long long *i*);

22642 **DESCRIPTION**

22643 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
22644 conflict between the requirements described here and the ISO C standard is unintentional. This  
22645 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

22646 These functions shall compute the absolute value of the **long** integer operand, *i*. If the result  
22647 cannot be represented, the behavior is undefined.

22648 **RETURN VALUE**

22649 These functions shall return the absolute value of the **long** integer operand.

22650 **ERRORS**

22651 No errors are defined.

22652 **EXAMPLES**

22653 None.

22654 **APPLICATION USAGE**

22655 None.

22656 **RATIONALE**

22657 None.

22658 **FUTURE DIRECTIONS**

22659 None.

22660 **SEE ALSO**

22661 *abs()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>

22662 **CHANGE HISTORY**

22663 First released in Issue 4. Derived from the ISO C standard.

22664 **Issue 6**

22665 The *llabs()* function is added for alignment with the ISO/IEC 9899:1999 standard.

22666 **NAME**

22667 lchown — change the owner and group of a symbolic link

22668 **SYNOPSIS**

22669 XSI #include &lt;unistd.h&gt;

22670 int lchown(const char \*path, uid\_t owner, gid\_t group);

22671

22672 **DESCRIPTION**

22673 The *lchown()* function shall have the same effect as *chown()* except in the case where the named  
 22674 file is a symbolic link. In this case, *lchown()* shall change the ownership of the symbolic link file  
 22675 itself, while *chown()* changes the ownership of the file or directory to which the symbolic link  
 22676 refers.

22677 **RETURN VALUE**

22678 Upon successful completion, *lchown()* shall return 0. Otherwise, it shall return  $-1$  and set *errno* to  
 22679 indicate an error.

22680 **ERRORS**22681 The *lchown()* function shall fail if:22682 [EACCES] Search permission is denied on a component of the path prefix of *path*.

22683 [EINVAL] The owner or group ID is not a value supported by the implementation.

22684 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 22685 argument.

22686 [ENAMETOOLONG]

22687 The length of a path name exceeds {PATH\_MAX} or a path name component  
 22688 is longer than {NAME\_MAX}.

22689 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.22690 [ENOTDIR] A component of the path prefix of *path* is not a directory.

22691 [EOPNOTSUPP] The *path* argument names a symbolic link and the implementation does not  
 22692 support setting the owner or group of a symbolic link.

22693 [EPERM] The effective user ID does not match the owner of the file and the process  
 22694 does not have appropriate privileges.

22695 [EROFS] The file resides on a read-only file system.

22696 The *lchown()* function may fail if:

22697 [EIO] An I/O error occurred while reading or writing to the file system.

22698 [EINTR] A signal was caught during execution of the function.

22699 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 22700 resolution of the *path* argument.

22701 [ENAMETOOLONG]

22702 Path name resolution of a symbolic link produced an intermediate result  
 22703 whose length exceeds {PATH\_MAX}.

22704 **EXAMPLES**22705 **Changing the Current Owner of a File**

22706 The following example shows how to change the ownership of the symbolic link named  
22707 **/modules/pass1** to the user ID associated with “jones” and the group ID associated with “cnd”.

22708 The numeric value for the user ID is obtained by using the *getpwnam()* function. The numeric  
22709 value for the group ID is obtained by using the *getgrnam()* function.

```
22710 #include <sys/types.h>
22711 #include <unistd.h>
22712 #include <pwd.h>
22713 #include <grp.h>

22714 struct passwd *pwd;
22715 struct group *grp;
22716 char *path = "/modules/pass1";
22717 ...
22718 pwd = getpwnam("jones");
22719 grp = getgrnam("cnd");
22720 lchown(path, pwd->pw_uid, grp->gr_gid);
```

22721 **APPLICATION USAGE**

22722 None.

22723 **RATIONALE**

22724 None.

22725 **FUTURE DIRECTIONS**

22726 None.

22727 **SEE ALSO**

22728 *chown()*, *symlink()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<unistd.h>**

22729 **CHANGE HISTORY**

22730 First released in Issue 4, Version 2.

22731 **Issue 5**

22732 Moved from X/OPEN UNIX extension to BASE.

22733 **Issue 6**

22734 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
22735 [ELOOP] error condition is added.

22736 **NAME**

22737        lcong48 — seed a uniformly distributed pseudo-random signed long integer generator

22738 **SYNOPSIS**

22739 xSI     #include <stdlib.h>

22740        void lcong48(unsigned short param[7]);

22741

22742 **DESCRIPTION**

22743        Refer to *drand48()*.

22744 **NAME**

22745 ldexp, ldexpf, ldexpl — load exponent of a floating point number

22746 **SYNOPSIS**

22747 #include <math.h>

22748 double ldexp(double x, int exp);

22749 float ldexpf(float x, int exp);

22750 long double ldexpl(long double x, int exp);

22751 **DESCRIPTION**

22752 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 22753 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22754 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

22755 These functions shall compute the quantity  $x * 2^{exp}$ .

22756 An application wishing to check for error situations should set *errno* to 0 before calling *ldexp()*.

22757 If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

22758 **RETURN VALUE**

22759 Upon successful completion, these functions shall return *x* multiplied by 2, raised to the power  
 22760 *exp*.

22761 **XSI** If the value of *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

22762 If *ldexp()* would cause overflow,  $\pm$ HUGE\_VAL shall be returned (according to the sign of *x*), and  
 22763 *errno* shall be set to [ERANGE].

22764 If *ldexp()* would cause underflow, 0 shall be returned and *errno* may be set to [ERANGE].

22765 **ERRORS**

22766 These functions shall fail if:

22767 [ERANGE] The value to be returned would have caused overflow.

22768 These functions may fail if:

22769 **XSI** [EDOM] The argument *x* is NaN.

22770 [ERANGE] The value to be returned would have caused underflow.

22771 No other errors shall occur.

22772 **EXAMPLES**

22773 None.

22774 **APPLICATION USAGE**

22775 None.

22776 **RATIONALE**

22777 None.

22778 **FUTURE DIRECTIONS**

22779 None.

22780 **SEE ALSO**

22781 *frexp()*, *isnan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

22782 **CHANGE HISTORY**

22783 First released in Issue 1. Derived from Issue 1 of the SVID.

22784 **Issue 4**

22785 References to *matherr()* are removed.

22786 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the  
22787 ISO C standard and to rationalize error handling in the mathematics functions.

22788 The return value specified for [EDOM] is marked as an extension.

22789 **Issue 5**

22790 The DESCRIPTION is updated to indicate how an application should check for an error. This  
22791 text was previously published in the APPLICATION USAGE section.

22792 **Issue 6**

22793 The *ldexpf()* and *ldexpl()* functions are added for alignment with the ISO/IEC 9899:1999  
22794 standard.

22795 **NAME**

22796 ldiv, lldiv — compute quotient and remainder of a long division

22797 **SYNOPSIS**

22798 #include <stdlib.h>

22799 ldiv\_t ldiv(long *numer*, long *denom*);

22800 lldiv\_t lldiv(long long *numer*, long long *denom*);

22801 **DESCRIPTION**

22802 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
22803 conflict between the requirements described here and the ISO C standard is unintentional. This  
22804 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

22805 These functions shall compute the quotient and remainder of the division of the numerator  
22806 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the **long**  
22807 integer of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be  
22808 represented, the behavior is undefined; otherwise, *quot* \* *denom* + *rem* shall equal *numer*.

22809 **RETURN VALUE**

22810 These functions shall return a structure of type **ldiv\_t**, comprising both the quotient and the  
22811 remainder. The structure includes the following members, in any order:

22812 long quot; /\* Quotient \*/

22813 long rem; /\* Remainder \*/

22814 **ERRORS**

22815 No errors are defined.

22816 **EXAMPLES**

22817 None.

22818 **APPLICATION USAGE**

22819 None.

22820 **RATIONALE**

22821 None.

22822 **FUTURE DIRECTIONS**

22823 None.

22824 **SEE ALSO**

22825 *div()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>

22826 **CHANGE HISTORY**

22827 First released in Issue 4. Derived from the ISO C standard.

22828 **Issue 6**

22829 The *lldiv()* function is added for alignment with the ISO/IEC 9899:1999 standard.



22830 **NAME**

22831 lfind — find entry in a linear search table

22832 **SYNOPSIS**

22833 XSI #include &lt;search.h&gt;

22834 void \*lfind(const void \*key, const void \*base, size\_t \*nelp,  
22835 size\_t width, int (\*compar)(const void \*, const void \*));

22836

22837 **DESCRIPTION**22838 Refer to *lsearch()*.

22839 **NAME**

22840 lgamma, lgammaf, lgammal — log gamma function

22841 **SYNOPSIS**

```
22842 xSI #include <math.h>
22843 double lgamma(double x);
22844 float lgammaf(float x);
22845 long double lgammal(long double x);
22846 extern int signgam;
22847
```

22848 **DESCRIPTION**

22849 These functions shall compute  $\log_e|\Gamma(x)|$  where  $\Gamma(x)$  is defined as  $\int_0^{\infty} e^{-t} t^{x-1} dt$ . The sign of  
 22850  $\Gamma(x)$  is returned in the external integer *signgam*. The argument *x* need not be a non-positive  
 22851 integer ( $\Gamma(x)$  is defined over the reals, except the non-positive integers).  
 22852

22853 An application wishing to check for error situations should set *errno* to 0 before calling *lgamma()*.  
 22854 If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

22855 These functions need not be reentrant. A function that is not required to be reentrant is not  
 22856 required to be thread-safe.

22857 **RETURN VALUE**

22858 Upon successful completion, these functions shall return the logarithmic gamma of *x*.

22859 If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

22860 If *x* is a non-positive integer, either HUGE\_VAL or NaN shall be returned and *errno* may be set to  
 22861 [ERANGE].

22862 If the correct value would cause overflow, *lgamma()* shall return HUGE\_VAL and may set *errno*  
 22863 to [ERANGE].

22864 If the correct value would cause underflow, *lgamma()* shall return 0 and may set *errno* to  
 22865 [ERANGE].

22866 **ERRORS**

22867 These functions may fail if:

22868 [EDOM] The value of *x* is NaN.

22869 [ERANGE] The value of *x* is a non-positive integer, or the value to be returned would  
 22870 have caused overflow or underflow.

22871 No other errors shall occur.

22872 **EXAMPLES**

22873 None.

22874 **APPLICATION USAGE**

22875 None.

22876 **RATIONALE**

22877 None.

22878 **FUTURE DIRECTIONS**

22879 None.

22880 **SEE ALSO**

22881 *exp()*, *isnan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

22882 **CHANGE HISTORY**

22883 First released in Issue 3.

22884 **Issue 4**

22885 This page no longer points to *gamma()*, but contains all information relating to *lgamma()*.

22886 The RETURN VALUE and ERRORS sections are substantially rewritten to rationalize error handling in the mathematics functions.

22888 **Issue 5**

22889 The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

22891 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

22892 **Issue 6**

22893 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 22894 • The *lgammaf()* and *lgammal()* functions are added.
  - 22895 • The RETURN VALUE and ERRORS sections are updated so that when *x* is a non-positive integer, *errno* may be set to [ERANGE]; previously, this was [EDOM].
- 22896

22897 **NAME**

22898 link — link to a file

22899 **SYNOPSIS**

22900 #include &lt;unistd.h&gt;

22901 int link(const char \*path1, const char \*path2);

22902 **DESCRIPTION**22903 The *link()* function shall create a new link (directory entry) for the existing file, *path1*.22904 The *path1* argument points to a path name naming an existing file. The *path2* argument points to  
22905 a path name naming the new directory entry to be created. The *link()* function shall atomically  
22906 create a new link for the existing file and the link count of the file shall be incremented by one.22907 If *path1* names a directory, *link()* shall fail unless the process has appropriate privileges and the  
22908 implementation supports using *link()* on directories.22909 Upon successful completion, *link()* shall mark for update the *st\_ctime* field of the file. Also, the  
22910 *st\_ctime* and *st\_mtime* fields of the directory that contains the new entry shall be marked for  
22911 update.22912 If *link()* fails, no link shall be created and the link count of the file shall remain unchanged.22913 The implementation may require that the calling process has permission to access the existing  
22914 file.22915 **RETURN VALUE**22916 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
22917 indicate the error.22918 **ERRORS**22919 The *link()* function shall fail if:22920 [EACCES] A component of either path prefix denies search permission, or the requested  
22921 link requires writing in a directory that denies write permission, or the calling  
22922 process does not have permission to access the existing file and this is  
22923 required by the implementation.22924 [EEXIST] The *path2* argument resolves to an existing file or refers to a symbolic link.22925 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path1* or  
22926 *path2* argument.22927 [EMLINK] The number of links to the file named by *path1* would exceed {LINK\_MAX}.

22928 [ENAMETOOLONG]

22929 The length of the *path1* or *path2* argument exceeds {PATH\_MAX} or a path  
22930 name component is longer than {NAME\_MAX}.22931 [ENOENT] A component of either path prefix does not exist; the file named by *path1* does  
22932 not exist; or *path1* or *path2* points to an empty string.

22933 [ENOSPC] The directory to contain the link cannot be extended.

22934 [ENOTDIR] A component of either path prefix is not a directory.

22935 [EPERM] The file named by *path1* is a directory and either the calling process does not  
22936 have appropriate privileges or the implementation prohibits using *link()* on  
22937 directories.

|       |                |                                                                                                                                                                             |
|-------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 22938 | [EROFS]        | The requested link requires writing in a directory on a read-only file system.                                                                                              |
| 22939 | [EXDEV]        | The link named by <i>path2</i> and the file named by <i>path1</i> are on different file systems and the implementation does not support links between file systems,         |
| 22940 | XSR            | or <i>path1</i> refers to a named STREAM.                                                                                                                                   |
| 22941 |                |                                                                                                                                                                             |
| 22942 |                | The <i>link()</i> function may fail if:                                                                                                                                     |
| 22943 | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path1</i> or <i>path2</i> argument.                                                     |
| 22944 |                |                                                                                                                                                                             |
| 22945 | [ENAMETOOLONG] | As a result of encountering a symbolic link in resolution of the <i>path1</i> or <i>path2</i> argument, the length of the substituted path name string exceeded {PATH_MAX}. |
| 22946 |                |                                                                                                                                                                             |
| 22947 |                |                                                                                                                                                                             |
| 22948 |                |                                                                                                                                                                             |

## 22949 EXAMPLES

### 22950 Creating a Link to a File

22951 The following example shows how to create a link to a file named `/home/cnd/mod1` by creating a  
22952 new directory entry named `/modules/pass1`.

```
22953 #include <unistd.h>
22954
22954 char *path1 = "/home/cnd/mod1";
22955 char *path2 = "/modules/pass1";
22956 int status;
22957 ...
22958 status = link (path1, path2);
```

### 22959 Creating a Link to a File Within a Program

22960 In the following program example, the *link()* function is used to link the `/etc/passwd` file  
22961 (defined as `PASSWDFILE`) to a file named `/etc/opasswd` (defined as `SAVEFILE`), which is used  
22962 to save the current password file. Then, after removing the current password file (defined as  
22963 `PASSWDFILE`), the new password file is saved as the current password file using the *link()*  
22964 function again.

```
22965 #include <unistd.h>
22966
22966 #define LOCKFILE "/etc/ptmp"
22967 #define PASSWDFILE "/etc/passwd"
22968 #define SAVEFILE "/etc/opasswd"
22969 ...
22970 /* Save current password file */
22971 link (PASSWDFILE, SAVEFILE);
22972
22972 /* Remove current password file. */
22973 unlink (PASSWDFILE);
22974
22974 /* Save new password file as current password file. */
22975 link (LOCKFILE, PASSWDFILE);
```

### 22976 APPLICATION USAGE

22977 Some implementations do allow links between file systems.

22978 **RATIONALE**

22979 Linking to a directory is restricted to the superuser in most historical implementations because  
 22980 this capability may produce loops in the file hierarchy or otherwise corrupt the file system. This  
 22981 volume of IEEE Std. 1003.1-200x continues that philosophy by prohibiting *link()* and *unlink()*  
 22982 from doing this. Other functions could do it if the implementor designed such an extension.

22983 Some historical implementations allow linking of files on different file systems. Wording was  
 22984 added to explicitly allow this optional behavior.

22985 The exception for cross-file system links is intended to apply only to links that are  
 22986 programmatically indistinguishable from “hard” links.

22987 **FUTURE DIRECTIONS**

22988 None.

22989 **SEE ALSO**

22990 *symlink()*, *unlink()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>

22991 **CHANGE HISTORY**

22992 First released in Issue 1. Derived from Issue 1 of the SVID.

22993 **Issue 4**

22994 The <**unistd.h**> header is added to the SYNOPSIS section.

22995 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 22996 • The type of arguments *path1* and *path2* are changed from **char\*** to **const char\***.

22997 The following change is incorporated for alignment with the FIPS requirements:

- 22998 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path  
 22999 name component is larger than {NAME\_MAX} is now defined as mandatory and marked as  
 23000 an extension.

23001 **Issue 4, Version 2**

23002 The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- 23003 • The [ELOOP] error is returned if too many symbolic links are encountered during path name  
 23004 resolution.
- 23005 • The [EXDEV] error may also be returned if *path1* refers to a named STREAM.
- 23006 • A second [ENAMETOOLONG] condition is defined that may report excessive length of an  
 23007 intermediate result of path name resolution of a symbolic link.

23008 **Issue 6**

23009 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 23010 • The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.  
 23011 This is since behavior may vary from one file system to another.

23012 The following new requirements on POSIX implementations derive from alignment with the  
 23013 Single UNIX Specification:

- 23014 • The [ELOOP] mandatory error condition is added.
- 23015 • A second [ENAMETOOLONG] is added as an optional error condition.

23016 The following changes were made to align with the IEEE P1003.1a draft standard:

- 23017 • An explanation is added of action when *path2* refers to a symbolic link.

23018

- The [ELOOP] optional error condition is added.

## 23019 NAME

23020 lio\_listio — list directed I/O (**REALTIME**)

## 23021 SYNOPSIS

23022 AIO #include &lt;aio.h&gt;

```
23023 int lio_listio(int mode, struct aiocb *restrict const list[restrict],
23024 int nent, struct sigevent *restrict sig);
23025
```

## 23026 DESCRIPTION

23027 The *lio\_listio()* function allows the calling process to initiate a list of I/O requests with a single  
23028 function call.

23029 The *mode* argument takes one of the values LIO\_WAIT or LIO\_NOWAIT declared in <aio.h> and  
23030 determines whether the function returns when the I/O operations have been completed, or as  
23031 soon as the operations have been queued. If the *mode* argument is LIO\_WAIT, the function waits  
23032 until all I/O is complete and the *sig* argument is ignored.

23033 If the *mode* argument is LIO\_NOWAIT, the function shall return immediately, and asynchronous  
23034 notification shall occur, according to the *sig* argument, when all the I/O operations complete. If  
23035 *sig* is NULL, then no asynchronous notification shall occur. If *sig* is not NULL, asynchronous  
23036 notification occurs as specified in Section 2.4.1 (on page 528) when all the requests in *list* have  
23037 completed.

23038 The I/O requests enumerated by *list* are submitted in an unspecified order.

23039 The *list* argument is an array of pointers to **aiocb** structures. The array contains *nent* elements.  
23040 The array may contain NULL elements, which shall be ignored.

23041 The *aio\_lio\_opcode* field of each **aiocb** structure specifies the operation to be performed. The  
23042 supported operations are LIO\_READ, LIO\_WRITE, and LIO\_NOP; these symbols are defined in  
23043 <aio.h>. The LIO\_NOP operation causes the list entry to be ignored. If the *aio\_lio\_opcode*  
23044 element is equal to LIO\_READ, then an I/O operation is submitted as if by a call to *aio\_read()*  
23045 with the *aiocbp* equal to the address of the **aiocb** structure. If the *aio\_lio\_opcode* element is equal  
23046 to LIO\_WRITE, then an I/O operation is submitted as if by a call to *aio\_write()* with the *aiocbp*  
23047 equal to the address of the **aiocb** structure.

23048 The *aio\_fildes* member specifies the file descriptor on which the operation is to be performed.

23049 The *aio\_buf* member specifies the address of the buffer to or from which the data is transferred.

23050 The *aio\_nbytes* member specifies the number of bytes of data to be transferred.

23051 The members of the **aiocb** structure further describe the I/O operation to be performed, in a  
23052 manner identical to that of the corresponding **aiocb** structure when used by the *aio\_read()* and  
23053 *aio\_write()* functions.

23054 The *nent* argument specifies how many elements are members of the list; that is, the length of the  
23055 array.

23056 The behavior of this function is altered according to the definitions of synchronized I/O data  
23057 integrity completion and synchronized I/O file integrity completion if synchronized I/O is  
23058 enabled on the file associated with *aio\_fildes*.

23059 For regular files, no data transfer shall occur past the offset maximum established in the open  
23060 file description associated with *aiocbp->aio\_fildes*.



## 23061 RETURN VALUE

23062 If the *mode* argument has the value LIO\_NOWAIT, the *lio\_listio()* function shall return the value  
 23063 zero if the I/O operations are successfully queued; otherwise, the function shall return the value  
 23064 -1 and set *errno* to indicate the error.

23065 If the *mode* argument has the value LIO\_WAIT, the *lio\_listio()* function shall return the value  
 23066 zero when all the indicated I/O has completed successfully. Otherwise, *lio\_listio()* shall return a  
 23067 value of -1 and set *errno* to indicate the error.

23068 In either case, the return value only indicates the success or failure of the *lio\_listio()* call itself,  
 23069 not the status of the individual I/O requests. In some cases one or more of the I/O requests  
 23070 contained in the list may fail. Failure of an individual request does not prevent completion of  
 23071 any other individual request. To determine the outcome of each I/O request, the application  
 23072 shall examine the error status associated with each **aiocb** control block. The error statuses so  
 23073 returned are identical to those returned as the result of an *aio\_read()* or *aio\_write()* function.

## 23074 ERRORS

23075 The *lio\_listio()* function shall fail if:

23076 [EAGAIN] The resources necessary to queue all the I/O requests were not available. The  
 23077 application may check the error status for each **aiocb** to determine the  
 23078 individual request(s) that failed.

23079 [EAGAIN] The number of entries indicated by *nent* would cause the system-wide limit  
 23080 {AIO\_MAX} to be exceeded.

23081 [EINVAL] The *mode* argument is not a proper value, or the value of *nent* was greater than  
 23082 {AIO\_LISTIO\_MAX}.

23083 [EINTR] A signal was delivered while waiting for all I/O requests to complete during  
 23084 an LIO\_WAIT operation. Note that, since each I/O operation invoked by  
 23085 *lio\_listio()* may possibly provoke a signal when it completes, this error return  
 23086 may be caused by the completion of one (or more) of the very I/O operations  
 23087 being awaited. Outstanding I/O requests are not canceled, and the application  
 23088 shall examine each list element to determine whether the request was  
 23089 initiated, canceled, or completed.

23090 [EIO] One or more of the individual I/O operations failed. The application may  
 23091 check the error status for each **aiocb** structure to determine the individual  
 23092 request(s) that failed.

23093 In addition to the errors returned by the *lio\_listio()* function, if the *lio\_listio()* function succeeds  
 23094 or fails with errors of [EAGAIN], [EINTR], or [EIO], then some of the I/O specified by the list  
 23095 may have been initiated. If the *lio\_listio()* function fails with an error code other than [EAGAIN],  
 23096 [EINTR], or [EIO], no operations from the list shall have been initiated. The I/O operation  
 23097 indicated by each list element can encounter errors specific to the individual read or write  
 23098 function being performed. In this event, the error status for each **aiocb** control block contains the  
 23099 associated error code. The error codes that can be set are the same as would be set by a *read()* or  
 23100 *write()* function, with the following additional error codes possible:

23101 [EAGAIN] The requested I/O operation was not queued due to resource limitations.

23102 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit  
 23103 *aio\_cancel()* request.

23104 [EFBIG] The *aiocbp->aio\_lio\_opcode* is LIO\_WRITE, the file is a regular file, *aiocbp->*  
 23105 *aio\_nbytes* is greater than 0, and the *aiocbp->aio\_offset* is greater than or equal  
 23106 to the offset maximum in the open file description associated with *aiocbp-*

23107 >*aio\_fildes*.

23108 [EINPROGRESS] The requested I/O is in progress.

23109 [EOVERFLOW] The *aiochp->aio\_lio\_opcode* is LIO\_READ, the file is a regular file, *aiochp->aio\_nbytes* is greater than 0, and the *aiochp->aio\_offset* is before the end-of-file and is greater than or equal to the offset maximum in the open file description associated with *aiochp->aio\_fildes*.

#### 23113 EXAMPLES

23114 None.

#### 23115 APPLICATION USAGE

23116 None.

#### 23117 RATIONALE

23118 Although it may appear that there are inconsistencies in the specified circumstances for error codes, the [EIO] error condition applies when any circumstance relating to an individual operation makes that operation fail. This might be due to a badly formulated request (for example, the *aio\_lio\_opcode* field is invalid, and *aio\_error()* returns [EINVAL]) or might arise from application behavior (for example, the file descriptor is closed before the operation is initiated, and *aio\_error()* returns [EBADF]).

23124 The limitation on the set of error codes returned when operations from the list shall have been initiated enables applications to know when operations have been started and whether *aio\_error()* is valid for a specific operation.

#### 23127 FUTURE DIRECTIONS

23128 None.

#### 23129 SEE ALSO

23130 *aio\_read()*, *aio\_write()*, *aio\_error()*, *aio\_return()*, *aio\_cancel()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*, *read()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**aio.h**>

#### 23132 CHANGE HISTORY

23133 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

23134 Large File Summit extensions are added.

#### 23135 Issue 6

23136 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

23138 The *lio\_listio()* function is marked as part of the Asynchronous Input and Output option.

23139 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

23141 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs past the offset maximum established in the open file description associated with *aiochp->aio\_fildes*. This change is to support large files.

23144 • The [EBIG] and [EOVERFLOW] error conditions are defined. This change is to support large files.

23146 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23147 The **restrict** keyword is added to the *lio\_listio()* prototype for alignment with the ISO/IEC 9899:1999 standard.

23149 **NAME**

23150 `listen` — listen for socket connections and limit the queue of incoming connections

23151 **SYNOPSIS**

23152 `#include <sys/socket.h>`

23153 `int listen(int socket, int backlog);`

23154 **DESCRIPTION**

23155 The `listen()` function shall mark a connection-mode socket, specified by the `socket` argument, as  
23156 accepting connections.

23157 The `backlog` argument provides a hint to the implementation which the implementation shall use  
23158 to limit the number of outstanding connections in the socket's listen queue. Implementations  
23159 may impose a limit on `backlog` and silently reduce the specified value. Normally, a larger `backlog`  
23160 argument value shall result in a larger or equal length of the listen queue. Implementations shall  
23161 support values of `backlog` up to `SOMAXCONN`, defined in `<sys/socket.h>`.

23162 The implementation may include incomplete connections in its listen queue. The limits on the  
23163 number of incomplete connections and completed connections queued may be different.

23164 The implementation may have an upper limit on the length of the listen queue—either global or  
23165 per accepting socket. If `backlog` exceeds this limit, the length of the listen queue is set to the limit.

23166 If `listen()` is called with a `backlog` argument value that is less than 0, the function behaves as if it  
23167 had been called with a `backlog` argument value of 0.

23168 A `backlog` argument of 0 may allow the socket to accept connections, in which case the length of  
23169 the listen queue may be set to an implementation-defined minimum value.

23170 The socket in use may require the process to have appropriate privileges to use the `listen()`  
23171 function.

23172 **RETURN VALUE**

23173 Upon successful completions, `listen()` shall return 0; otherwise, `-1` shall be returned and `errno` set  
23174 to indicate the error.

23175 **ERRORS**

23176 The `listen()` function shall fail if:

23177 [EBADF] The `socket` argument is not a valid file descriptor.

23178 [EDESTADDRREQ]

23179 The socket is not bound to a local address, and the protocol does not support  
23180 listening on an unbound socket.

23181 [EINVAL] The `socket` is already connected.

23182 [ENOTSOCK] The `socket` argument does not refer to a socket.

23183 [EOPNOTSUPP] The socket protocol does not support `listen()`.

23184 The `listen()` function may fail if:

23185 [EACCES] The calling process does not have the appropriate privileges.

23186 [EINVAL] The `socket` has been shut down.

23187 [ENOBUFS] Insufficient resources are available in the system to complete the call.

23188 **EXAMPLES**

23189 None.

23190 **APPLICATION USAGE**

23191 None.

23192 **RATIONALE**

23193 None.

23194 **FUTURE DIRECTIONS**

23195 None.

23196 **SEE ALSO**

23197 *accept()*, *connect()*, *socket()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/socket.h> |

23198 **CHANGE HISTORY**

23199 First released in Issue 6. Derived from the XNS, Issue 5.2 specification. |

23200 The DESCRIPTION is updated to describe the relationship of SOMAXCONN and the *backlog* |

23201 argument.

23202 **NAME**

23203 llrint, llrintf, llrintl, lrint, lrintf, lrintl — round to nearest integer value using current rounding  
23204 direction

23205 **SYNOPSIS**

```
23206 #include <math.h>

23207 long long llrint(double x);
23208 long long llrintf(float x);
23209 long long llrintl(long double x);
23210 long lrint(double x);
23211 long lrintf(float x);
23212 long lrintl(long double x);
```

23213 **DESCRIPTION**

23214 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
23215 conflict between the requirements described here and the ISO C standard is unintentional. This  
23216 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

23217 These functions shall round their argument to the nearest integer value, rounding according to  
23218 the current rounding direction.

23219 An application wishing to check for error situations should set *errno* to 0 before calling these  
23220 functions. If *errno* is non-zero on return, an error has occurred.

23221 **RETURN VALUE**

23222 Upon successful completion, these functions shall return the rounded integer value.

23223 If the rounded value is outside the range of the return type, the numeric result is unspecified.

23224 If the magnitude of *x* is too large, the numeric result is unspecified and *errno* may be set to  
23225 [ERANGE].

23226 **ERRORS**

23227 These functions may fail if:

23228 [ERANGE] The magnitude of *x* is too large.

23229 **EXAMPLES**

23230 None.

23231 **APPLICATION USAGE**

23232 None.

23233 **RATIONALE**

23234 None.

23235 **FUTURE DIRECTIONS**

23236 None.

23237 **SEE ALSO**

23238 The Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

23239 **CHANGE HISTORY**

23240 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23241 **NAME**

23242 lround, llroundf, llroundl, lround, lroundf, lroundl — round to nearest integer value

23243 **SYNOPSIS**

23244 #include <math.h>

23245 long long llround(double x);

23246 long long llroundf(float x);

23247 long long llroundl(long double x);

23248 long lround(double x);

23249 long lroundf(float x);

23250 long lroundl(long double x);

23251 **DESCRIPTION**

23252 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
23253 conflict between the requirements described here and the ISO C standard is unintentional. This  
23254 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

23255 These functions shall round their argument to the nearest integer value, rounding halfway cases  
23256 away from zero, regardless of the current rounding direction.

23257 An application wishing to check for error situations should set *errno* to 0 before calling these  
23258 functions. If *errno* is non-zero on return, an error has occurred.

23259 **RETURN VALUE**

23260 Upon successful completion, these functions shall return the rounded integer value.

23261 If the rounded value is outside the range of the return type, the numeric result is unspecified.

23262 If the magnitude of *x* is too large, the numeric result is unspecified and *errno* may be set to  
23263 [ERANGE].

23264 **ERRORS**

23265 These functions may fail if:

23266 [ERANGE] The magnitude of *x* is too large.

23267 **EXAMPLES**

23268 None.

23269 **APPLICATION USAGE**

23270 None.

23271 **RATIONALE**

23272 None.

23273 **FUTURE DIRECTIONS**

23274 None.

23275 **SEE ALSO**

23276 The Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

23277 **CHANGE HISTORY**

23278 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23279 **NAME**

23280 localeconv — return locale-specific information

23281 **SYNOPSIS**

23282 #include &lt;locale.h&gt;

23283 struct lconv \*localeconv(void);

23284 **DESCRIPTION**

23285 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 23286 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23287 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

23288 The *localeconv()* function shall set the components of an object with the type **struct lconv** with  
 23289 the values appropriate for the formatting of numeric quantities (monetary and otherwise)  
 23290 according to the rules of the current locale.

23291 The members of the structure with type **char\*** are pointers to strings, any of which (except  
 23292 **decimal\_point**) can point to " ", to indicate that the value is not available in the current locale or  
 23293 is of zero length. The members with type **char** are non-negative numbers, any of which can be  
 23294 {CHAR\_MAX} to indicate that the value is not available in the current locale.

23295 The members include the following:

23296 **char \*decimal\_point**

23297 The radix character used to format non-monetary quantities.

23298 **char \*thousands\_sep**

23299 The character used to separate groups of digits before the decimal-point character in  
 23300 formatted non-monetary quantities.

23301 **char \*grouping**

23302 A string whose elements taken as one-byte integer values indicate the size of each group of  
 23303 digits in formatted non-monetary quantities.

23304 **char \*int\_curr\_symbol**

23305 The international currency symbol applicable to the current locale. The first three  
 23306 characters contain the alphabetic international currency symbol in accordance with those  
 23307 specified in the ISO 4217:1995 standard. The fourth character (immediately preceding the  
 23308 null byte) is the character used to separate the international currency symbol from the  
 23309 monetary quantity.

23310 **char \*currency\_symbol**

23311 The local currency symbol applicable to the current locale.

23312 **char \*mon\_decimal\_point**

23313 The radix character used to format monetary quantities.

23314 **char \*mon\_thousands\_sep**

23315 The separator for groups of digits before the decimal-point in formatted monetary  
 23316 quantities.

23317 **char \*mon\_grouping**

23318 A string whose elements taken as one-byte integer values indicate the size of each group of  
 23319 digits in formatted monetary quantities.

23320 **char \*positive\_sign**

23321 The string used to indicate a non-negative valued formatted monetary quantity.

|           |                                                                                                          |
|-----------|----------------------------------------------------------------------------------------------------------|
| 23322     | <b>char *negative_sign</b>                                                                               |
| 23323     | The string used to indicate a negative valued formatted monetary quantity.                               |
| 23324     | <b>char int_frac_digits</b>                                                                              |
| 23325     | The number of fractional digits (those after the decimal-point) to be displayed in an                    |
| 23326     | internationally formatted monetary quantity.                                                             |
| 23327     | <b>char frac_digits</b>                                                                                  |
| 23328     | The number of fractional digits (those after the decimal-point) to be displayed in a                     |
| 23329     | formatted monetary quantity.                                                                             |
| 23330     | <b>char p_cs_precedes</b>                                                                                |
| 23331     | Set to 1 if the <b>currency_symbol</b> or <b>int_curr_symbol</b> precedes the value for a non-negative   |
| 23332     | formatted monetary quantity. Set to 0 if the symbol succeeds the value.                                  |
| 23333     | <b>char p_sep_by_space</b>                                                                               |
| 23334     | Set to 0 if no space separates the <b>currency_symbol</b> or <b>int_curr_symbol</b> from the value for a |
| 23335     | non-negative formatted monetary quantity. Set to 1 if a space separates the symbol from the              |
| 23336 XSI | value; and set to 2 if a space separates the symbol and the sign string, if adjacent.                    |
| 23337     | <b>char n_cs_precedes</b>                                                                                |
| 23338     | Set to 1 if the <b>currency_symbol</b> or <b>int_curr_symbol</b> precedes the value for a negative       |
| 23339     | formatted monetary quantity. Set to 0 if the symbol succeeds the value.                                  |
| 23340     | <b>char n_sep_by_space</b>                                                                               |
| 23341     | Set to 0 if no space separates the <b>currency_symbol</b> or <b>int_curr_symbol</b> from the value for a |
| 23342     | negative formatted monetary quantity. Set to 1 if a space separates the symbol from the                  |
| 23343 XSI | value; and set to 2 if a space separates the symbol and the sign string, if adjacent.                    |
| 23344     | <b>char p_sign_posn</b>                                                                                  |
| 23345     | Set to a value indicating the positioning of the <b>positive_sign</b> for a non-negative formatted       |
| 23346     | monetary quantity.                                                                                       |
| 23347     | <b>char n_sign_posn</b>                                                                                  |
| 23348     | Set to a value indicating the positioning of the <b>negative_sign</b> for a negative formatted           |
| 23349     | monetary quantity.                                                                                       |
| 23350     | <b>char int_p_cs_precedes</b>                                                                            |
| 23351     | Set to 1 or 0 if the <b>int_currency_symbol</b> respectively precedes or succeeds the value for a        |
| 23352     | non-negative internationally formatted monetary quantity.                                                |
| 23353     | <b>char int_n_cs_precedes</b>                                                                            |
| 23354     | Set to 1 or 0 if the <b>int_currency_symbol</b> respectively precedes or succeeds the value for a        |
| 23355     | negative internationally formatted monetary quantity.                                                    |
| 23356     | <b>char int_p_sep_by_space</b>                                                                           |
| 23357     | Set to a value indicating the separation of the <b>int_currency_symbol</b> , the sign string, and the    |
| 23358     | value for a non-negative internationally formatted monetary quantity.                                    |
| 23359     | <b>char int_n_sep_by_space</b>                                                                           |
| 23360     | Set to a value indicating the separation of the <b>int_currency_symbol</b> , the sign string, and the    |
| 23361     | value for a negative internationally formatted monetary quantity.                                        |
| 23362     | <b>char int_p_sign_posn</b>                                                                              |
| 23363     | Set to a value indicating the positioning of the <b>positive_sign</b> for a non-negative                 |
| 23364     | internationally formatted monetary quantity.                                                             |
| 23365     | <b>char int_n_sign_posn</b>                                                                              |
| 23366     | Set to a value indicating the positioning of the <b>negative_sign</b> for a negative internationally     |



- 23367 formatted monetary quantity.
- 23368 The elements of **grouping** and **mon\_grouping** are interpreted according to the following:
- 23369 {CHAR\_MAX} No further grouping is to be performed.
- 23370 0 The previous element is to be repeatedly used for the remainder of the digits.
- 23371 *other* The integer value is the number of digits that comprise the current group. The
- 23372 next element is examined to determine the size of the next group of digits
- 23373 before the current group.
- 23374 The values of **p\_sep\_by\_space**, **n\_sep\_by\_space**, **int\_p\_sep\_by\_space**, and **int\_n\_sep\_by\_space**
- 23375 are interpreted according to the following:
- 23376 0 No space separates the currency symbol and value.
- 23377 1 If the currency symbol and sign string are adjacent, a space separates them from the value;
- 23378 otherwise, a space separates the currency symbol from the value.
- 23379 2 If the currency symbol and sign string are adjacent, a space separates them; otherwise, a
- 23380 space separates the sign string from the value.
- 23381 The values of **p\_sign\_posn**, **n\_sign\_posn**, **int\_p\_sign\_posn**, and **int\_n\_sign\_posn** are
- 23382 interpreted according to the following:
- 23383 0 Parentheses surround the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 23384 1 The sign string precedes the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 23385 2 The sign string succeeds the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 23386 3 The sign string immediately precedes the **currency\_symbol** or **int\_curr\_symbol**.
- 23387 4 The sign string immediately succeeds the **currency\_symbol** or **int\_curr\_symbol**.
- 23388 The implementation shall behave as if no function in this volume of IEEE Std. 1003.1-200x calls
- 23389 *localeconv()*.
- 23390 cx The *localeconv()* function need not be reentrant. A function that is not required to be reentrant is
- 23391 not required to be thread-safe.

#### 23392 RETURN VALUE

- 23393 The *localeconv()* function shall return a pointer to the filled-in object. The application shall not
- 23394 modify the structure pointed to by the return value which may be overwritten by a subsequent
- 23395 call to *localeconv()*. In addition, calls to *setlocale()* with the categories *LC\_ALL*, *LC\_MONETARY*,
- 23396 or *LC\_NUMERIC* may overwrite the contents of the structure.

#### 23397 ERRORS

- 23398 No errors are defined.

#### 23399 EXAMPLES

- 23400 None.

#### 23401 APPLICATION USAGE

- 23402 The following table illustrates the rules which may be used by four countries to format monetary
- 23403 quantities.

23404  
23405  
23406  
23407  
23408  
23409

| Country     | Positive Format | Negative Format | International Format |
|-------------|-----------------|-----------------|----------------------|
| Italy       | L.1.230         | -L.1.230        | ITL.1.230            |
| Netherlands | F 1.234,56      | F -1.234,56     | NLG 1.234,56         |
| Norway      | kr1.234,56      | kr1.234,56-     | NOK 1.234,56         |
| Switzerland | SFrS.1,234.56   | SFrS.1,234.56C  | CHF 1,234.56         |

23410 For these four countries, the respective values for the monetary members of the structure  
23411 returned by *localeconv()* are:

23412  
23413  
23414  
23415  
23416  
23417  
23418  
23419  
23420  
23421  
23422  
23423  
23424  
23425  
23426  
23427  
23428  
23429  
23430  
23431  
23432  
23433

|                           | Italy   | Netherlands | Norway | Switzerland |
|---------------------------|---------|-------------|--------|-------------|
| <b>int_curr_symbol</b>    | "ITL. " | "NLG "      | "NOK " | "CHF "      |
| <b>currency_symbol</b>    | "L. "   | "F "        | "kr "  | "SFrS. "    |
| <b>mon_decimal_point</b>  | " "     | ","         | ","    | ."          |
| <b>mon_thousands_sep</b>  | ". "    | ". "        | ". "   | ". "        |
| <b>mon_grouping</b>       | "\3 "   | "\3 "       | "\3 "  | "\3 "       |
| <b>positive_sign</b>      | " "     | " "         | " "    | " "         |
| <b>negative_sign</b>      | "- "    | "- "        | "- "   | "C "        |
| <b>int_frac_digits</b>    | 0       | 2           | 2      | 2           |
| <b>frac_digits</b>        | 0       | 2           | 2      | 2           |
| <b>p_cs_precedes</b>      | 1       | 1           | 1      | 1           |
| <b>p_sep_by_space</b>     | 0       | 1           | 0      | 0           |
| <b>n_cs_precedes</b>      | 1       | 1           | 1      | 1           |
| <b>n_sep_by_space</b>     | 0       | 1           | 0      | 0           |
| <b>p_sign_posn</b>        | 1       | 1           | 1      | 1           |
| <b>n_sign_posn</b>        | 1       | 4           | 2      | 2           |
| <b>int_p_cs_precedes</b>  | 1       | 1           | 1      | 1           |
| <b>int_n_cs_precedes</b>  | 1       | 1           | 1      | 1           |
| <b>int_p_sep_by_space</b> | 0       | 0           | 0      | 0           |
| <b>int_n_sep_by_space</b> | 0       | 0           | 0      | 0           |
| <b>int_p_sign_posn</b>    | 1       | 1           | 1      | 1           |
| <b>int_n_sign_posn</b>    | 1       | 4           | 4      | 2           |

23434 **RATIONALE**  
23435 None.

23436 **FUTURE DIRECTIONS**  
23437 None.

23438 **SEE ALSO**  
23439 *isalpha()*, *isascii()*, *nl\_langinfo()*, *printf()*, *scanf()*, *setlocale()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*,  
23440 *strcpy()*, *strftime()*, *strlen()*, *strpbrk()*, *strspn()*, *strtok()*, *strxfrm()*, *strtod()*, the Base Definitions  
23441 volume of IEEE Std. 1003.1-200x, <langinfo.h>, <locale.h>

23442 **CHANGE HISTORY**  
23443 First released in Issue 4. Derived from the ANSI C standard.

23444 **Issue 6**  
23445 A note indicating that this function need not be reentrant is added to the DESCRIPTION.  
23446 The RETURN VALUE section is rewritten to avoid use of the term ‘‘must’’.  
23447 This reference page is updated for alignment with the ISO/IEC 9899: 1999 standard.

23448 **NAME**

23449 localtime, localtime\_r — convert a time value to a broken-down local time

23450 **SYNOPSIS**

23451 #include &lt;time.h&gt;

23452 struct tm \*localtime(const time\_t \*timer);

23453 TSF struct tm \*localtime\_r(const time\_t \*restrict timer,

23454 struct tm \*restrict result);

23455

23456 **DESCRIPTION**

23457 CX For *localtime()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

23460 The *localtime()* function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time, expressed as a local time. The function corrects for the timezone and any seasonal time adjustments. Local timezone information is used as though *localtime()* calls *tzset()*.

23464 CX The *localtime()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

23466 TSF The *localtime\_r()* function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time stored in the structure to which *result* points. The *localtime\_r()* function shall also return a pointer to that same structure.

23469 Unlike *localtime()*, the reentrant version is not required to set *tzname*.

23470 **RETURN VALUE**23471 The *localtime()* function shall return a pointer to the broken-down time structure.

23472 TSF Upon successful completion, *localtime\_r()* shall return a pointer to the structure pointed to by the argument *result*.

23474 **ERRORS**

23475 No errors are defined.

23476 **EXAMPLES**23477 **Getting the Local Date and Time**

23478 The following example uses the *time()* function to calculate the time elapsed, in seconds, since January 1, 1970 0:00 UTC (the Epoch), *localtime()* to convert that value to a broken-down time, and *asctime()* to convert the broken-down time values into a printable string.

23481 #include &lt;stdio.h&gt;

23482 #include &lt;time.h&gt;

23483 main()

23484 {

23485 time\_t result;

23486 result = time(NULL);

23487 printf("%s%ld secs since the Epoch\n",

23488 asctime(localtime(&amp;result)),

23489 (long)result);

23490 return(0);

23491 }

23492 This example writes the current time to *stdout* in a form like this:

```
23493 Wed Jun 26 10:32:15 1996
23494 835810335 secs since the Epoch
```

### 23495 Getting the Modification Time for a File

23496 The following example gets the modification time for a file. The *localtime()* function converts the **time\_t** value of the last modification date, obtained by a previous call to *stat()*, into a **tm** structure that contains the year, month, day, and so on.

```
23499 #include <time.h>
23500 ...
23501 struct stat statbuf;
23502 ...
23503 tm = localtime(&statbuf.st_mtime);
23504 ...
```

### 23505 Timing an Event

23506 The following example gets the current time, converts it to a string using *localtime()* and *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to an event being timed.

```
23509 #include <time.h>
23510 #include <stdio.h>
23511 ...
23512 time_t now;
23513 int minutes_to_event;
23514 ...
23515 time(&now);
23516 printf("The time is ");
23517 fputs(asctime(localtime(&now)), stdout);
23518 printf("There are still %d minutes to the event.\n",
23519 minutes_to_event);
23520 ...
```

### 23521 APPLICATION USAGE

23522 The *asctime()*, *ctime()*, *getdate()*, *gettimeofday()*, *gmtime()*, and *localtime()* functions return values in one of two static objects: a broken-down time structure and an array of **char**. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

23526 The *localtime\_r()* function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

### 23528 RATIONALE

23529 None.

### 23530 FUTURE DIRECTIONS

23531 None.

### 23532 SEE ALSO

23533 *asctime()*, *clock()*, *ctime()*, *difftime()*, *getdate()*, *gettimeofday()*, *gmtime()*, *mktime()*, *strftime()*,  
23534 *strptime()*, *time()*, *utime()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**time.h**>

23535 **CHANGE HISTORY**

23536 First released in Issue 1. Derived from Issue 1 of the SVID.

23537 **Issue 4**

23538 The APPLICATION USAGE section is expanded to provide a more complete description of how  
23539 static areas are used by the *\*time()* functions.

23540 The following change is incorporated for alignment with the ISO C standard:

- 23541 • The *timer* argument is now a type **const time\_t**.

23542 **Issue 5**

23543 A note indicating that the *localtime()* function need not be reentrant is added to the  
23544 DESCRIPTION.

23545 The *localtime\_r()* function is included for alignment with the POSIX Threads Extension.

23546 **Issue 6**

23547 The *localtime\_r()* function is marked as part of the Thread-Safe Functions option.

23548 Extensions beyond the ISO C standard are now marked.

23549 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
23550 its avoidance of possibly using a static data area.

23551 The **restrict** keyword is added to the *localtime\_r()* prototype for alignment with the  
23552 ISO/IEC 9899:1999 standard.

## 23553 NAME

23554 lockf — record locking on files

## 23555 SYNOPSIS

23556 XSI #include &lt;unistd.h&gt;

23557 int lockf(int *fildes*, int *function*, off\_t *size*);

23558

## 23559 DESCRIPTION

23560 The *lockf()* function allows sections of a file to be locked with advisory-mode locks. Calls to  
 23561 *lockf()* from other threads which attempt to lock the locked file section shall either return an  
 23562 error value or block until the section becomes unlocked. All the locks for a process are removed  
 23563 when the process terminates. Record locking with *lockf()* is supported for regular files and may  
 23564 be supported for other files.

23565 The *fildes* argument is an open file descriptor. The application shall ensure that the file descriptor  
 23566 has been opened with write-only permission (O\_WRONLY) or with read/write permission  
 23567 (O\_RDWR) to establish a lock with this function.

23568 The *function* argument is a control value which specifies the action to be taken. The permissible  
 23569 values for *function* are defined in <unistd.h> as follows:

23570

23571

| Function | Description                                  |
|----------|----------------------------------------------|
| F_ULOCK  | Unlock locked sections.                      |
| F_LOCK   | Lock a section for exclusive use.            |
| F_TLOCK  | Test and lock a section for exclusive use.   |
| F_TEST   | Test a section for locks by other processes. |

23572

23573

23574

23575

23576 F\_TEST detects if a lock by another process is present on the specified section.

23577 F\_LOCK and F\_TLOCK both lock a section of a file if the section is available.

23578 F\_ULOCK removes locks from a section of the file.

23579 The *size* argument is the number of contiguous bytes to be locked or unlocked. The section to be  
 23580 locked or unlocked starts at the current offset in the file and extends forward for a positive size  
 23581 or backward for a negative size (the preceding bytes up to but not including the current offset).  
 23582 If *size* is 0, the section from the current offset through the largest possible file offset is locked  
 23583 (that is, from the current offset through the present or any future end-of-file). An area need not  
 23584 be allocated to the file to be locked because locks may exist past the end-of-file.

23585 The sections locked with F\_LOCK or F\_TLOCK may, in whole or in part, contain or be contained  
 23586 by a previously locked section for the same process. When this occurs, or if adjacent locked  
 23587 sections would occur, the sections are combined into a single locked section. If the request  
 23588 would cause the number of locks to exceed a system-imposed limit, the request shall fail.

23589 F\_LOCK and F\_TLOCK requests differ only by the action taken if the section is not available.  
 23590 F\_LOCK blocks the calling thread until the section is available. F\_TLOCK makes the function  
 23591 fail if the section is already locked by another process.

23592 File locks are released on first close by the locking process of any file descriptor for the file.

23593 F\_ULOCK requests may release (wholly or in part) one or more locked sections controlled by the  
 23594 process. Locked sections shall be unlocked starting at the current file offset through *size* bytes or  
 23595 to the end-of-file if *size* is (off\_t)0. When all of a locked section is not released (that is, when the  
 23596 beginning or end of the area to be unlocked falls within a locked section), the remaining portions  
 23597 of that section are still locked by the process. Releasing the center portion of a locked section

23598 shall cause the remaining locked beginning and end portions to become two separate locked  
 23599 sections. If the request would cause the number of locks in the system to exceed a system-  
 23600 imposed limit, the request shall fail.

23601 A potential for deadlock occurs if the threads of a process controlling a locked section are  
 23602 blocked by accessing another process' locked section. If the system detects that deadlock would  
 23603 occur, *lockf()* shall fail with an [EDEADLK] error.

23604 The interaction between *fcntl()* and *lockf()* locks is unspecified.

23605 Blocking on a section is interrupted by any signal.

23606 An F\_ULOCK request in which *size* is non-zero and the offset of the last byte of the requested  
 23607 section is the maximum value for an object of type *off\_t*, when the process has an existing lock  
 23608 in which *size* is 0 and which includes the last byte of the requested section, shall be treated as a  
 23609 request to unlock from the start of the requested section with a size equal to 0. Otherwise, an  
 23610 F\_ULOCK request shall attempt to unlock only the requested section.

23611 Attempting to lock a section of a file that is associated with a buffered stream produces  
 23612 unspecified results.

#### 23613 RETURN VALUE

23614 Upon successful completion, *lockf()* shall return 0. Otherwise, it shall return -1, set *errno* to  
 23615 indicate an error, and existing locks shall not be changed.

#### 23616 ERRORS

23617 The *lockf()* function shall fail if:

23618 [EBADF] The *fildev* argument is not a valid open file descriptor; or *function* is F\_LOCK  
 23619 or F\_TLOCK and *fildev* is not a valid file descriptor open for writing.

23620 [EACCES] or [EAGAIN]

23621 The *function* argument is F\_TLOCK or F\_TEST and the section is already  
 23622 locked by another process.

23623 [EDEADLK] The *function* argument is F\_LOCK and a deadlock is detected.

23624 [EINTR] A signal was caught during execution of the function.

23625 [EINVAL] The *function* argument is not one of F\_LOCK, F\_TLOCK, F\_TEST, or  
 23626 F\_ULOCK; or *size* plus the current file offset is less than 0.

23627 [EOVERFLOW] The offset of the first, or if *size* is not 0 then the last, byte in the requested  
 23628 section cannot be represented correctly in an object of type *off\_t*.

23629 The *lockf()* function may fail if:

23630 [EAGAIN] The *function* argument is F\_LOCK or F\_TLOCK and the file is mapped with  
 23631 *mmap()*.

23632 [EDEADLK] or [ENOLCK]

23633 The *function* argument is F\_LOCK, F\_TLOCK, or F\_ULOCK, and the request  
 23634 would cause the number of locks to exceed a system-imposed limit.

23635 [EOPNOTSUPP] or [EINVAL]

23636 The implementation does not support the locking of files of the type indicated  
 23637 by the *fildev* argument.

23638 **EXAMPLES**23639 **Locking a Portion of a File**

23640 In the following example, a file named `/home/cnd/mod1` is being modified. Other processes that  
23641 use locking are prevented from changing it during this process. Only the first 10,000 bytes are  
23642 locked, and the lock call fails if another process has any part of this area locked already.

```
23643 #include <fcntl.h>
23644 #include <unistd.h>
23645 int fildes;
23646 int status;
23647 ...
23648 fildes = open("/home/cnd/mod1", O_RDWR);
23649 status = lockf(fildes, F_TLOCK, (off_t)10000);
```

23650 **APPLICATION USAGE**

23651 Record-locking should not be used in combination with the *fopen()*, *fread()*, *fwrite()*, and other  
23652 *stdio* functions. Instead, the more primitive, non-buffered functions (such as *open()*) should be  
23653 used. Unexpected results may occur in processes that do buffering in the user address space. The  
23654 process may later read/write data which is/was locked. The *stdio* functions are the most  
23655 common source of unexpected buffering.

23656 The *alarm()* function may be used to provide a timeout facility in applications requiring it.

23657 **RATIONALE**

23658 None.

23659 **FUTURE DIRECTIONS**

23660 None.

23661 **SEE ALSO**

23662 *alarm()*, *chmod()*, *close()*, *creat()*, *fcntl()*, *fopen()*, *mmap()*, *open()*, *read()*, *write()*, the Base  
23663 Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>

23664 **CHANGE HISTORY**

23665 First released in Issue 4, Version 2.

23666 **Issue 5**

23667 Moved from X/OPEN UNIX extension to BASE.

23668 Large File Summit extensions are added. In particular, the description of [EINVAL] is clarified  
23669 and moved from optional to mandatory status.

23670 A note is added to the DESCRIPTION indicating the effects of attempting to lock a section of a  
23671 file that is associated with a buffered stream.

23672 **Issue 6**

23673 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.



23674 **NAME**

23675 log, logf, logl — natural logarithm function

23676 **SYNOPSIS**

23677 #include &lt;math.h&gt;

23678 double log(double x);

23679 float logf(float x);

23680 long double logl(long double x);

23681 **DESCRIPTION**

23682 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23683 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23684 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

23685 These functions shall compute the natural logarithm of  $x$ ,  $\log_e(x)$ . The application shall ensure  
 23686 that the value of  $x$  is positive.

23687 An application wishing to check for error situations should set *errno* to 0 before calling *log()*. If  
 23688 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

23689 **RETURN VALUE**23690 Upon successful completion, these functions shall return the natural logarithm of  $x$ .23691 XSI If  $x$  is NaN, NaN shall be returned and *errno* may be set to [EDOM].23692 XSI If  $x$  is less than 0, -HUGE\_VAL or NaN shall be returned, and *errno* shall be set to [EDOM].23693 If  $x$  is 0, -HUGE\_VAL shall be returned and *errno* may be set to [ERANGE].23694 **ERRORS**

23695 These functions shall fail if:

23696 [EDOM] The value of  $x$  is negative.

23697 These functions may fail if:

23698 XSI [EDOM] The value of  $x$  is NaN.23699 [ERANGE] The value of  $x$  is 0.

23700 XSI No other errors shall occur.

23701 **EXAMPLES**

23702 None.

23703 **APPLICATION USAGE**

23704 None.

23705 **RATIONALE**

23706 None.

23707 **FUTURE DIRECTIONS**

23708 None.

23709 **SEE ALSO**23710 *exp()*, *isnan()*, *log10()*, *log1p()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>23711 **CHANGE HISTORY**

23712 First released in Issue 1. Derived from Issue 1 of the SVID.

23713 **Issue 4**

23714 References to *matherr()* are removed.

23715 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the  
23716 ISO C standard and to rationalize error handling in the mathematics functions.

23717 The return value specified for [EDOM] is marked as an extension.

23718 **Issue 5**

23719 The DESCRIPTION is updated to indicate how an application should check for an error. This  
23720 text was previously published in the APPLICATION USAGE section.

23721 **Issue 6**

23722 The DESCRIPTION is updated to avoid use of the term “must” for application requirements. |

23723 The *logf()* and *logl()* functions are added for alignment with the ISO/IEC 9899:1999 standard. |

23724 **NAME**

23725 log10, log10f, log10l — base 10 logarithm function

23726 **SYNOPSIS**

23727 #include &lt;math.h&gt;

23728 double log10(double x);

23729 float log10f(float x);

23730 long double log10l(long double x);

23731 **DESCRIPTION**

23732 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23733 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23734 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

23735 These functions shall compute the base 10 logarithm of  $x$ ,  $\log_{10}(x)$ . The application shall ensure  
 23736 that the value of  $x$  is positive.

23737 An application wishing to check for error situations should set *errno* to 0 before calling *log10()*.  
 23738 If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

23739 **RETURN VALUE**23740 Upon successful completion, these functions shall return the base 10 logarithm of  $x$ .23741 **XSI** If  $x$  is NaN, NaN shall be returned and *errno* may be set to [EDOM].23742 **XSI** If  $x$  is less than 0, -HUGE\_VAL or NaN shall be returned, and *errno* shall be set to [EDOM].23743 If  $x$  is 0, -HUGE\_VAL shall be returned and *errno* may be set to [ERANGE].23744 **ERRORS**

23745 These functions shall fail if:

23746 [EDOM] The value of  $x$  is negative.

23747 These functions may fail if:

23748 **XSI** [EDOM] The value of  $x$  is NaN.23749 [ERANGE] The value of  $x$  is 0.23750 **XSI** No other errors shall occur.23751 **EXAMPLES**

23752 None.

23753 **APPLICATION USAGE**

23754 None.

23755 **RATIONALE**

23756 None.

23757 **FUTURE DIRECTIONS**

23758 None.

23759 **SEE ALSO**23760 *isnan()*, *log()*, *pow()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>23761 **CHANGE HISTORY**

23762 First released in Issue 1. Derived from Issue 1 of the SVID.

23763 **Issue 4**

23764       References to *matherr()* are removed.

23765       The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the  
23766       ISO C standard and to rationalize error handling in the mathematics functions.

23767       The return value specified for [EDOM] is marked as an extension.

23768 **Issue 5**

23769       The DESCRIPTION is updated to indicate how an application should check for an error. This  
23770       text was previously published in the APPLICATION USAGE section.

23771 **Issue 6**

23772       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23773       The *log10f()* and *log10l()* functions are added for alignment with the ISO/IEC 9899:1999  
23774       standard.

23775 **NAME**

23776 log1p, log1pf, log1pl — compute a natural logarithm

23777 **SYNOPSIS**

23778 #include &lt;math.h&gt;

23779 double log1p(double x);

23780 float log1pf(float x);

23781 long double log1pl(long double x);

23782 **DESCRIPTION**23783 These functions shall compute  $\log_e(1.0 + x)$ . The application shall ensure that the value of  $x$  is  
23784 greater than  $-1.0$ .23785 **RETURN VALUE**23786 Upon successful completion, these functions shall return the natural logarithm of  $1.0 + x$ .23787 If  $x$  is NaN, *log1p()* shall return NaN and may set *errno* to [EDOM].23788 If  $x$  is less than  $-1.0$ , *log1p()* shall return  $-\text{HUGE\_VAL}$  or NaN and set *errno* to [EDOM].23789 If  $x$  is  $-1.0$ , *log1p()* shall return  $-\text{HUGE\_VAL}$  and may set *errno* to [ERANGE].23790 **ERRORS**

23791 These functions shall fail if:

23792 [EDOM] The value of  $x$  is less than  $-1.0$ .23793 These functions may fail and set *errno* to:23794 [EDOM] The value of  $x$  is NaN.23795 [ERANGE] The value of  $x$  is  $-1.0$ .

23796 No other errors shall occur.

23797 **EXAMPLES**

23798 None.

23799 **APPLICATION USAGE**

23800 None.

23801 **RATIONALE**

23802 None.

23803 **FUTURE DIRECTIONS**

23804 None.

23805 **SEE ALSO**23806 *log()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>23807 **CHANGE HISTORY**

23808 First released in Issue 4, Version 2.

23809 **Issue 5**

23810 Moved from X/OPEN UNIX extension to BASE.

23811 **Issue 6**

23812 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23813 The *log1pf()* and *log1pl()* functions are added for alignment with the ISO/IEC 9899:1999  
23814 standard.

23815 **NAME**

23816 log2, log2f, log2l — compute base 2 logarithm functions

23817 **SYNOPSIS**

23818 #include <math.h>

23819 double log2(double x);

23820 float log2f(float x);

23821 long double log2l(long double x);

23822 **DESCRIPTION**

23823 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
23824 conflict between the requirements described here and the ISO C standard is unintentional. This  
23825 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

23826 These functions shall compute the base 2 logarithm of  $x$ ,  $\log_2(x)$ . The application shall ensure  
23827 that the value of  $x$  is positive.

23828 An application wishing to check for error situations should set *errno* to 0 before calling these  
23829 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

23830 **RETURN VALUE**

23831 Upon successful completion, these functions shall return the base 2 logarithm of  $x$ .

23832 If  $x$  is NaN, these functions shall return NaN and may set *errno* to [EDOM].

23833 If  $x$  is less than 0, these functions shall return  $-\text{HUGE\_VAL}$  or NaN and set *errno* to [EDOM].

23834 If  $x$  is 0, these functions shall return  $-\text{HUGE\_VAL}$  and may set *errno* to [ERANGE].

23835 **ERRORS**

23836 These functions shall fail if:

23837 [EDOM] The value of  $x$  is less than 0.

23838 These functions may fail if:

23839 [EDOM] The value of  $x$  is NaN.

23840 [ERANGE] The value of  $x$  is 0.

23841 No other errors shall occur.

23842 **EXAMPLES**

23843 None.

23844 **APPLICATION USAGE**

23845 None.

23846 **RATIONALE**

23847 None.

23848 **FUTURE DIRECTIONS**

23849 None.

23850 **SEE ALSO**

23851 *log()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

23852 **CHANGE HISTORY**

23853 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23854 **NAME**

23855 logb, logbf, logbl — radix-independent exponent

23856 **SYNOPSIS**

23857 XSI #include &lt;math.h&gt;

23858 double logb(double x);

23859 float logbf(float x);

23860 long double logbl(long double x);

23861

23862 **DESCRIPTION**

23863 These functions shall compute the exponent of  $x$ , which is the integral part of  $\log_r |x|$ , as a  
 23864 signed floating point value, for non-zero  $x$ , where  $r$  is the radix of the machine's floating-point  
 23865 arithmetic, which is the value of FLT\_RADIX defined in the <float.h> header.

23866 **RETURN VALUE**23867 Upon successful completion, these functions shall return the exponent of  $x$ .23868 If  $x$  is 0.0, *logb()* shall return `-HUGE_VAL` and set *errno* to [EDOM].23869 If  $x$  is  $\pm\text{Inf}$ , *logb()* shall return `+Inf`.23870 If  $x$  is NaN, *logb()* shall return NaN and may set *errno* to [EDOM].23871 **ERRORS**

23872 These functions shall fail if:

23873 [EDOM] The  $x$  argument is 0.0.

23874 These functions may fail if:

23875 [EDOM] The  $x$  argument is NaN.23876 **EXAMPLES**

23877 None.

23878 **APPLICATION USAGE**

23879 None.

23880 **RATIONALE**

23881 None.

23882 **FUTURE DIRECTIONS**

23883 None.

23884 **SEE ALSO**23885 *ilogb()*, *scalb()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <float.h> <math.h>23886 **CHANGE HISTORY**

23887 First released in Issue 4, Version 2.

23888 **Issue 5**

23889 Moved from X/OPEN UNIX extension to BASE.

23890 **Issue 6**23891 The *logbf()* and *logbl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

## 23892 NAME

23893 longjmp — non-local goto

## 23894 SYNOPSIS

23895 #include &lt;setjmp.h&gt;

23896 void longjmp(jmp\_buf env, int val);

## 23897 DESCRIPTION

23898 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 23899 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23900 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

23901 The *longjmp()* function shall restore the environment saved by the most recent invocation of  
 23902 *setjmp()* in the same thread, with the corresponding **jmp\_buf** argument. If there is no such  
 23903 invocation, or if the function containing the invocation of the *setjmp()* macro has terminated  
 23904 execution in the interim, or if the invocation of *setjmp()* was within the scope of an identifier  
 23905 with variably modified type and execution has left that scope in the interim, the behavior is  
 23906 cx undefined. It is unspecified whether *longjmp()* restores the signal mask, leaves the signal mask  
 23907 unchanged, or restores it to its value at the time *setjmp()* was called.

23908 All accessible objects have values as of the time *longjmp()* was called, and all other components  
 23909 of the abstract machine have state (for example, floating-point status flags and open files),  
 23910 except that the values of objects of automatic storage duration are indeterminate if they meet all  
 23911 the following conditions:

- 23912 • They are local to the function containing the corresponding *setjmp()* invocation.
- 23913 • They do not have volatile-qualified type.
- 23914 • They are changed between the *setjmp()* invocation and *longjmp()* call.

23915 As it bypasses the usual function call and return mechanisms, *longjmp()* shall execute correctly  
 23916 in contexts of interrupts, signals, and any of their associated functions. However, if *longjmp()* is  
 23917 invoked from a nested signal handler (that is, from a function invoked as a result of a signal  
 23918 raised during the handling of another signal), the behavior is undefined.

23919 cx The effect of a call to *longjmp()* where initialization of the **jmp\_buf** structure was not performed  
 23920 in the calling thread is undefined.

## 23921 RETURN VALUE

23922 After *longjmp()* is completed, program execution continues as if the corresponding invocation of  
 23923 *setjmp()* had just returned the value specified by *val*. The *longjmp()* function shall not cause  
 23924 *setjmp()* to return 0; if *val* is 0, *setjmp()* shall return 1.

## 23925 ERRORS

23926 No errors are defined.

## 23927 EXAMPLES

23928 None.

## 23929 APPLICATION USAGE

23930 Applications whose behavior depends on the value of the signal mask should not use *longjmp()*  
 23931 and *setjmp()*, since their effect on the signal mask is unspecified, but should instead use the  
 23932 *siglongjmp()* and *sigsetjmp()* functions (which can save and restore the signal mask under  
 23933 application control).



23934 **RATIONALE**

23935 None.

23936 **FUTURE DIRECTIONS**

23937 None.

23938 **SEE ALSO**

23939 *setjmp()*, *sigaction()*, *siglongjmp()*, *sigsetjmp()*, the Base Definitions volume of  
23940 IEEE Std. 1003.1-200x, <**setjmp.h**>

23941 **CHANGE HISTORY**

23942 First released in Issue 1. Derived from Issue 1 of the SVID.

23943 **Issue 4**

23944 The APPLICATION USAGE section is deleted.

23945 The following change is incorporated for alignment with the ISO C standard:

23946

- Mention of volatile-qualified types is added to the DESCRIPTION.

23947 **Issue 4, Version 2**

23948 The DESCRIPTION is updated for X/OPEN UNIX conformance and discusses valid possibilities  
23949 for the resulting state of the signal mask.

23950 **Issue 5**

23951 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

23952 **Issue 6**

23953 Extensions beyond the ISO C standard are now marked.

23954 The following new requirements on POSIX implementations derive from alignment with the  
23955 Single UNIX Specification:

23956

- The DESCRIPTION now explicitly makes *longjmp()*'s effect on the signal mask unspecified.

23957 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

23958 **NAME**

23959       lrand48 — generate uniformly distributed pseudo-random non-negative long integers

23960 **SYNOPSIS**

23961 xSI     #include <stdlib.h>

23962       long lrand48(void);

23963

23964 **DESCRIPTION**

23965       Refer to *drand48()*.

23966 **NAME**

23967 lsearch, lfind — linear search and update

23968 **SYNOPSIS**

```
23969 xSI #include <search.h>
23970 void *lsearch(const void *key, void *base, size_t *nel, size_t width,
23971 int (*compar)(const void *, const void *));
23972 void *lfind(const void *key, const void *base, size_t *nel,
23973 size_t width, int (*compar)(const void *, const void *));
23974
```

23975 **DESCRIPTION**

23976 The *lsearch()* function is a linear search routine. It returns a pointer into the table for the  
 23977 matching entry. If the entry does not occur, it is added at the end of the table. The *key* argument  
 23978 points to the entry to be sought in the table. The *base* argument points to the first element in the  
 23979 table. The *width* argument is the size of an element in bytes. The *nel* argument points to an  
 23980 integer containing the current number of elements in the table. The integer to which *nel* points  
 23981 is incremented if the entry is added to the table. The *compar* argument points to a comparison  
 23982 function which the application shall supply (for example, *strcmp()*). It is called with two  
 23983 arguments that point to the elements being compared. The application shall ensure that the  
 23984 function returns 0 if the elements are equal, and non-zero otherwise.

23985 The *lfind()* function is the same as *lsearch()* except that if the entry is not found, it is not added to  
 23986 the table. Instead, a null pointer is returned.

23987 **RETURN VALUE**

23988 If the searched for entry is found, both *lsearch()* and *lfind()* shall return a pointer to it. Otherwise,  
 23989 *lfind()* shall return a null pointer and *lsearch()* shall return a pointer to the newly added element.

23990 Both functions shall return a null pointer in case of error.

23991 **ERRORS**

23992 No errors are defined.

23993 **EXAMPLES**23994 **Storing Strings in a Table**

23995 This fragment reads in less than or equal to TABSIZE strings of length less than or equal to  
 23996 ELSIZE and stores them in a table, eliminating duplicates.

```
23997 #include <stdio.h>
23998 #include <string.h>
23999 #include <search.h>
24000 #define TABSIZE 50
24001 #define ELSIZE 120
24002 ...
24003 char line[ELSIZE], tab[TABSIZE][ELSIZE];
24004 size_t nel = 0;
24005 ...
24006 while (fgets(line, ELSIZE, stdin) != NULL && nel < TABSIZE)
24007 (void) lsearch(line, tab, &nel,
24008 ELSIZE, (int (*)(const void *, const void *)) strcmp);
24009 ...
```

24010 **Finding a Matching Entry**

24011 The following example finds any line that reads "This is a test.".

```
24012 #include <search.h>
24013 #include <string.h>
24014 ...
24015 char line[ELSIZE], tab[TABSIZE][ELSIZE];
24016 size_t nel = 0;
24017 char *findline;
24018 void *entry;

24019 findline = "This is a test.\n";

24020 entry = lfind(findline, tab, &nel, ELSIZE, (
24021 int (*)(const void *, const void *)) strcmp);
```

24022 **APPLICATION USAGE**

24023 The comparison function need not compare every byte, so arbitrary data may be contained in  
24024 the elements in addition to the values being compared.

24025 Undefined results can occur if there is not enough room in the table to add a new item.

24026 **RATIONALE**

24027 None.

24028 **FUTURE DIRECTIONS**

24029 None.

24030 **SEE ALSO**

24031 *hcreate()*, *tsearch()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**search.h**>

24032 **CHANGE HISTORY**

24033 First released in Issue 1. Derived from Issue 1 of the SVID.

24034 **Issue 4**

24035 In the SYNOPSIS section, the type of argument *key* in the declaration of *lsearch()* is changed from  
24036 **void\*** to **const void\***, the type arguments *key* and *base* have been changed from **void\*** to **const**  
24037 **void\*** in the declaration of *lfind()*, and the arguments to *compar* are defined for both functions.

24038 In the EXAMPLES section, the sample code is updated to use ISO C standard syntax.

24039 Warnings about the casting of various arguments are removed from the APPLICATION USAGE  
24040 section, as casting requirements are now clear from the function definitions.

24041 **Issue 6**

24042 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

24043 **NAME**24044 `lseek` — move the read/write file offset24045 **SYNOPSIS**24046 `#include <unistd.h>`24047 `off_t lseek(int fildev, off_t offset, int whence);`24048 **DESCRIPTION**24049 The `lseek()` function shall set the file offset for the open file description associated with the file descriptor *fildev*, as follows:

- 24051 • If *whence* is {SEEK\_SET}, the file offset is set to *offset* bytes.
- 24052 • If *whence* is {SEEK\_CUR}, the file offset is set to its current location plus *offset*.
- 24053 • If *whence* is {SEEK\_END}, the file offset is set to the size of the file plus *offset*.

24054 The symbolic constants {SEEK\_SET}, {SEEK\_CUR}, and {SEEK\_END} are defined in `<unistd.h>`.24055 The behavior of `lseek()` on devices which are incapable of seeking is implementation-defined.  
24056 The value of the file offset associated with such a device is undefined.24057 The `lseek()` function shall allow the file offset to be set beyond the end of the existing data in the file. If data is later written at this point, subsequent reads of data in the gap shall return bytes with the value 0 until data is actually written into the gap.24060 The `lseek()` function shall not, by itself, extend the size of a file.24061 SHM If *fildev* refers to a shared memory object, the result of the `lseek()` function is unspecified.24062 TYM If *fildev* refers to a typed memory object, the result of the `lseek()` function is unspecified.24063 **RETURN VALUE**24064 Upon successful completion, the resulting offset, as measured in bytes from the beginning of the file, shall be returned. Otherwise, `(off_t)-1` shall be returned, `errno` shall be set to indicate the error, and the file offset shall remain unchanged.24067 **ERRORS**24068 The `lseek()` function shall fail if:

- |       |             |                                                                                                                                                        |
|-------|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| 24069 | [EBADF]     | The <i>fildev</i> argument is not an open file descriptor.                                                                                             |
| 24070 | [EINVAL]    | The <i>whence</i> argument is not a proper value, or the resulting file offset would be negative for a regular file, block special file, or directory. |
| 24072 | [EOVERFLOW] | The resulting file offset would be a value which cannot be represented correctly in an object of type <code>off_t</code> .                             |
| 24074 | [ESPIPE]    | The <i>fildev</i> argument is associated with a pipe, FIFO, or socket.                                                                                 |

24075 **EXAMPLES**

24076 None.

24077 **APPLICATION USAGE**

24078 None.

24079 **RATIONALE**24080 The ISO C standard includes the functions `fgetpos()` and `fsetpos()`, which work on very large files by use of a special positioning type.24082 Although `lseek()` may position the file offset beyond the end of the file, this function does not itself extend the size of the file. While the only function in this volume of IEEE Std. 1003.1-200x

- 24084 that may directly extend the size of the file is `write()`, several functions originally derived from  
24085 the ISO C standard, such as `fwrite()`, `fprintf()`, and so on, may do so (by causing calls on `write()`).
- 24086 An invalid file offset that would cause [EINVAL] to be returned may be both implementation-  
24087 defined and device-dependent (for example, memory may have few invalid values). A negative  
24088 file offset may be valid for some devices in some implementations.
- 24089 The POSIX.1-1990 standard did not specifically prohibit `lseek()` from returning a negative offset.  
24090 Therefore, an application was required to clear `errno` prior to the call and check `errno` upon return  
24091 to determine whether a return value of `(off_t)-1` is a negative offset or an indication of an error  
24092 condition. The standard developers did not wish to require this action on the part of a portable  
24093 application, and chose to require that `errno` be set to [EINVAL] when the resulting file offset  
24094 would be negative for a regular file, block special file, or directory.
- 24095 **FUTURE DIRECTIONS**
- 24096 None.
- 24097 **SEE ALSO**
- 24098 `open()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<sys/types.h>`, `<unistd.h>`
- 24099 **CHANGE HISTORY**
- 24100 First released in Issue 1. Derived from Issue 1 of the SVID.
- 24101 **Issue 4**
- 24102 The `<sys/types.h>` header is now marked as optional (OH); this header need not be included on  
24103 XSI-conformant systems.
- 24104 The APPLICATION USAGE section is removed, as the ISO POSIX-1 standard now requires that  
24105 `off_t` be signed.
- 24106 **Issue 5**
- 24107 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.
- 24108 Large File Summit extensions are added.
- 24109 **Issue 6**
- 24110 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.
- 24111 The following new requirements on POSIX implementations derive from alignment with the  
24112 Single UNIX Specification:
- 24113 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
24114 required for conforming implementations of previous POSIX specifications, it was not  
24115 required for UNIX applications.
  - 24116 • The [EOVERFLOW] error condition is added. This change is to support large files.
- 24117 An additional [ESPIPE] error condition is added for sockets.
- 24118 The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by specifying that  
24119 `lseek()` results are unspecified for typed memory objects.

24120 **NAME**

24121 lstat — get symbolic link status

24122 **SYNOPSIS**

24123 #include &lt;sys/stat.h&gt;

24124 int lstat(const char \*restrict *path*, struct stat \*restrict *buf*);24125 **DESCRIPTION**

24126 The *lstat()* function shall have the same effect as *stat()*, except when *path* refers to a symbolic  
 24127 link. In that case *lstat()* shall return information about the link, while *stat()* shall return  
 24128 information about the file the link references.

24129 For symbolic links, the *st\_mode* member shall contain meaningful information when used with  
 24130 the file type macros, and the *st\_size* member shall contain the length of the path name contained  
 24131 in the symbolic link. File mode bits and the contents of the remaining members of the **stat**  
 24132 structure are unspecified. The value returned in the *st\_size* member is the length of the contents  
 24133 of the symbolic link, and does not count any trailing null.

24134 **RETURN VALUE**

24135 Upon successful completion, *lstat()* shall return 0. Otherwise, it shall return  $-1$  and set *errno* to  
 24136 indicate the error.

24137 **ERRORS**24138 The *lstat()* function shall fail if:

24139 [EACCES] A component of the path prefix denies search permission.

24140 [EIO] An error occurred while reading from the file system.

24141 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 24142 argument.

24143 [ENAMETOOLONG]

24144 The length of a path name exceeds {PATH\_MAX} or a path name component  
 24145 is longer than {NAME\_MAX}.

24146 [ENOTDIR] A component of the path prefix is not a directory.

24147 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

24148 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file  
 24149 serial number cannot be represented correctly in the structure pointed to by  
 24150 *buf*.

24151 The *lstat()* function may fail if:

24152 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 24153 resolution of the *path* argument.

24154 [ENAMETOOLONG]

24155 As a result of encountering a symbolic link in resolution of the *path* argument,  
 24156 the length of the substituted path name string exceeded {PATH\_MAX}.

24157 [EOVERFLOW] One of the members is too large to store into the structure pointed to by the  
 24158 *buf* argument.

24159 **EXAMPLES**24160 **Obtaining Symbolic Link Status Information**

24161 The following example shows how to obtain status information for a symbolic link named  
24162 **/modules/pass1**. The structure variable *buffer* is defined for the **stat** structure. If the *path*  
24163 argument specified the file name for the file pointed to by the symbolic link (**/home/cnd/mod1**),  
24164 the results of calling the function would be the same as those returned by a call to the *stat()*  
24165 function.

```
24166 #include <sys/stat.h>
24167 struct stat buffer;
24168 int status;
24169 ...
24170 status = lstat("/modules/pass1", &buffer);
```

24171 **APPLICATION USAGE**

24172 None.

24173 **RATIONALE**

24174 The *lstat()* function is not required to update the time-related fields if the named file is not a  
24175 symbolic link. While the *st\_uid*, *st\_gid*, *st\_atime*, *st\_mtime*, and *st\_ctime* members of the **stat**  
24176 structure may apply to a symbolic link, they are not required to do so. No functions in  
24177 IEEE Std. 1003.1-200x are required to maintain any of these time fields.

24178 **FUTURE DIRECTIONS**

24179 None.

24180 **SEE ALSO**

24181 *lstat()*, *readlink()*, *stat()*, *symlink()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
24182 **<sys/stat.h>**

24183 **CHANGE HISTORY**

24184 First released in Issue 4, Version 2.

24185 **Issue 5**

24186 Moved from X/OPEN UNIX extension to BASE.

24187 Large File Summit extensions are added.

24188 **Issue 6**

24189 The following changes were made to align with the IEEE P1003.1a draft standard:

- 24190 • This function is now mandatory.
- 24191 • The [ELOOP] optional error condition is added.

24192 The **restrict** keyword is added to the *lstat()* prototype for alignment with the ISO/IEC 9899:1999  
24193 standard.



24194 **NAME**

24195           makecontext, swapcontext — manipulate user contexts

24196 **SYNOPSIS**

24197 XSI       #include &lt;ucontext.h&gt;

24198       void makecontext(ucontext\_t \*ucp, void (\*func)(void),  
24199                       int argc, ...);

24200       int swapcontext(ucontext\_t \*restrict oucp,

24201                       const ucontext\_t \*restrict ucp);

24202

24203 **DESCRIPTION**

24204       The *makecontext()* function shall modify the context specified by *ucp*, which has been initialized  
24205       using *getcontext()*. When this context is resumed using *swapcontext()* or *setcontext()*, program  
24206       execution shall continue by calling *func*, passing it the arguments that follow *argc* in the  
24207       *makecontext()* call.

24208       Before a call is made to *makecontext()*, the application shall ensure that the context being  
24209       modified has a stack allocated for it. The application shall ensure that the value of *argc* matches  
24210       the number of integer arguments passed to *func*; otherwise, the behavior is undefined.

24211       The *uc\_link* member is used to determine the context that shall be resumed when the context  
24212       being modified by *makecontext()* returns. The application shall ensure that the *uc\_link* member is  
24213       initialized prior to the call to *makecontext()*.

24214       The *swapcontext()* function shall save the current context in the context structure pointed to by  
24215       *oucp* and shall set the context to the context structure pointed to by *ucp*.

24216 **RETURN VALUE**

24217       Upon successful completion, *swapcontext()* shall return 0. Otherwise,  $-1$  shall be returned and  
24218       *errno* set to indicate the error.

24219 **ERRORS**24220       The *swapcontext()* function shall fail if:24221       [ENOMEM]       The *ucp* argument does not have enough stack left to complete the operation.24222 **EXAMPLES**

24223       None.

24224 **APPLICATION USAGE**

24225       None.

24226 **RATIONALE**

24227       None.

24228 **FUTURE DIRECTIONS**

24229       None.

24230 **SEE ALSO**

24231       *exit()*, *getcontext()*, *sigaction()*, *sigprocmask()*, the Base Definitions volume of  
24232       IEEE Std. 1003.1-200x, <**ucontext.h**>

24233 **CHANGE HISTORY**

24234       First released in Issue 4, Version 2.

24235 **Issue 5**

24236 Moved from X/OPEN UNIX extension to BASE.

24237 In the ERRORS section, the description of [ENOMEM] is changed to apply to *swapcontext()* only.

24238 **Issue 6**

24239 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

24240 The **restrict** keyword is added to the *swapcontext()* prototype for alignment with the

24241 ISO/IEC 9899:1999 standard.

24242 **NAME**24243            **malloc** — a memory allocator24244 **SYNOPSIS**

24245            #include &lt;stdlib.h&gt;

24246            void \*malloc(size\_t *size*);24247 **DESCRIPTION**

24248 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
 24249 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24250 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

24251            The *malloc()* function shall allocate unused space for an object whose size in bytes is specified by  
 24252 *size* and whose value is indeterminate.

24253            The order and contiguity of storage allocated by successive calls to *malloc()* is unspecified. The  
 24254 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to  
 24255 a pointer to any type of object and then used to access such an object in the space allocated (until  
 24256 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object  
 24257 disjoint from any other object. The pointer returned points to the start (lowest byte address) of  
 24258 the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of  
 24259 the space requested is 0, the behavior is implementation-defined; the value returned shall be  
 24260 either a null pointer or a unique pointer.

24261 **RETURN VALUE**

24262            Upon successful completion with *size* not equal to 0, *malloc()* shall return a pointer to the  
 24263 allocated space. If *size* is 0, either a null pointer or a unique pointer that can be successfully  
 24264 **CX** passed to *free()* shall be returned. Otherwise, it shall return a null pointer and set *errno* to  
 24265 indicate the error.

24266 **ERRORS**24267            The *malloc()* function shall fail if:24268 **CX**        [ENOMEM]        Insufficient storage space is available.24269 **EXAMPLES**

24270            None.

24271 **APPLICATION USAGE**

24272            None.

24273 **RATIONALE**

24274            None.

24275 **FUTURE DIRECTIONS**

24276            None.

24277 **SEE ALSO**24278            *calloc()*, *free()*, *realloc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>24279 **CHANGE HISTORY**

24280            First released in Issue 1. Derived from Issue 1 of the SVID.

24281 **Issue 4**24282            The setting of *errno* and the [ENOMEM] error are marked as extensions.

24283            The APPLICATION USAGE section is changed to record that <malloc.h> need no longer be  
 24284 supported on XSI-conformant systems.

- 24285 The following change is incorporated for alignment with the ISO C standard:
- 24286
  - The RETURN VALUE section is updated to indicate what is returned if *size* is 0.
- 24287 **Issue 6**
- 24288 Extensions beyond the ISO C standard are now marked.
- 24289 The following new requirements on POSIX implementations derive from alignment with the  
24290 Single UNIX Specification:
- 24291
  - In the RETURN VALUE section, the requirement to set *errno* to indicate an error is added.
- 24292
  - The [ENOMEM] error condition is added.

24293 **NAME**

24294           mblen — get number of bytes in a character

24295 **SYNOPSIS**

24296           #include &lt;stdlib.h&gt;

24297           int mblen(const char \*s, size\_t n);

24298 **DESCRIPTION**

24299 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 24300 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24301 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

24302       If *s* is not a null pointer, *mblen()* determines the number of bytes constituting the character  
 24303 pointed to by *s*. Except that the shift state of *mbtowc()* is not affected, it is equivalent to:

24304       mbtowc((wchar\_t \*)0, s, n);

24305       The implementation shall behave as if no function defined in this volume of  
 24306 IEEE Std. 1003.1-200x calls *mblen()*.

24307       The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
 24308 state-dependent encoding, this function is placed into its initial state by a call for which its  
 24309 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null  
 24310 pointer cause the internal state of the function to be altered as necessary. A call with *s* as a null  
 24311 pointer causes this function to return a non-zero value if encodings have state dependency, and  
 24312 0 otherwise. If the implementation employs special bytes to change the shift state, these bytes do  
 24313 not produce separate wide-character codes, but are grouped with an adjacent character.  
 24314 Changing the *LC\_CTYPE* category causes the shift state of this function to be indeterminate.

24315 **RETURN VALUE**

24316       If *s* is a null pointer, *mblen()* shall return a non-zero or 0 value, if character encodings,  
 24317 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mblen()* shall  
 24318 either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the  
 24319 character (if the next *n* or fewer bytes form a valid character), or return -1 (if they do not form a  
 24320 **CX** valid character) and may set *errno* to indicate the error. In no case shall the value returned be  
 24321 greater than *n* or the value of the {MB\_CUR\_MAX} macro.

24322 **ERRORS**24323       The *mblen()* function may fail if:24324 **XSI**       [EILSEQ]       Invalid character sequence is detected.24325 **EXAMPLES**

24326       None.

24327 **APPLICATION USAGE**

24328       None.

24329 **RATIONALE**

24330       None.

24331 **FUTURE DIRECTIONS**

24332       None.

24333 **SEE ALSO**

24334       *mbtowc()*, *mbstowcs()*, *wctomb()*, *wcstombs()*, the Base Definitions volume of  
 24335 IEEE Std. 1003.1-200x, <stdlib.h>

24336 **CHANGE HISTORY**

24337 First released in Issue 4. Aligned with the ISO C standard.

24338 **NAME**24339 `mbrlen` — get number of bytes in a character (restartable)24340 **SYNOPSIS**24341 `#include <wchar.h>`24342 `size_t mbrlen(const char *restrict s, size_t n,`  
24343 `mbstate_t *restrict ps);`24344 **DESCRIPTION**24345 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
24346 conflict between the requirements described here and the ISO C standard is unintentional. This  
24347 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.24348 If *s* is not a null pointer, `mbrlen()` shall determine the number of bytes constituting the character  
24349 pointed to by *s*. It is equivalent to:24350 `mbstate_t internal;`24351 `mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);`24352 If *ps* is a null pointer, the `mbrlen()` function uses its own internal `mbstate_t` object, which is  
24353 initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t` object  
24354 pointed to by *ps* is used to completely describe the current conversion state of the associated  
24355 character sequence. The implementation shall behave as if no function defined in this volume of  
24356 IEEE Std. 1003.1-200x calls `mbrlen()`.24357 **XSI** The behavior of this function is affected by the `LC_CTYPE` category of the current locale.24358 **RETURN VALUE**24359 The `mbrlen()` function shall return the first of the following that applies:24360 **0** If the next *n* or fewer bytes complete the character that corresponds to the null  
24361 wide character.24362 **positive** If the next *n* or fewer bytes complete a valid character; the value returned shall  
24363 be the number of bytes that complete the character.24364 **(size\_t)-2** If the next *n* bytes contribute to an incomplete but potentially valid character,  
24365 and all *n* bytes have been processed. When *n* has at least the value of the  
24366 `{MB_CUR_MAX}` macro, this case can only occur if *s* points at a sequence of  
24367 redundant shift sequences (for implementations with state-dependent  
24368 encodings).24369 **(size\_t)-1** If an encoding error occurs, in which case the next *n* or fewer bytes do not  
24370 contribute to a complete and valid character. In this case, `[EILSEQ]` shall be  
24371 stored in `errno` and the conversion state is undefined.24372 **ERRORS**24373 The `mbrlen()` function may fail if:24374 `[EINVAL]` *ps* points to an object that contains an invalid conversion state.24375 `[EILSEQ]` Invalid character sequence is detected.

24376 **EXAMPLES**

24377 None.

24378 **APPLICATION USAGE**

24379 None.

24380 **RATIONALE**

24381 None.

24382 **FUTURE DIRECTIONS**

24383 None.

24384 **SEE ALSO**24385 *mbsinit()*, *mbrtowc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**wchar.h**>24386 **CHANGE HISTORY**

24387 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995

24388 (E).

24389 **Issue 6**24390 The *mbrlen()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.



24391 **NAME**

24392 mbrtowc — convert a character to a wide-character code (restartable)

24393 **SYNOPSIS**

24394 #include &lt;wchar.h&gt;

24395 size\_t mbrtowc(wchar\_t \*restrict *pwc*, const char \*restrict *s*,  
24396 size\_t *n*, mbstate\_t \*restrict *ps*);24397 **DESCRIPTION**24398 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
24399 conflict between the requirements described here and the ISO C standard is unintentional. This  
24400 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.24401 If *s* is a null pointer, the *mbrtowc()* function shall be equivalent to the call:24402 mbrtowc(NULL, "", 1, *ps*)24403 In this case, the values of the arguments *pwc* and *n* are ignored.24404 If *s* is not a null pointer, the *mbrtowc()* function inspects at most *n* bytes beginning at the byte  
24405 pointed to by *s* to determine the number of bytes needed to complete the next character  
24406 (including any shift sequences). If the function determines that the next character is completed, it  
24407 determines the value of the corresponding wide character and then, if *pwc* is not a null pointer,  
24408 stores that value in the object pointed to by *pwc*. If the corresponding wide character is the null  
24409 wide character, the resulting state described is the initial conversion state.24410 If *ps* is a null pointer, the *mbrtowc()* function uses its own internal **mbstate\_t** object, which is  
24411 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
24412 pointed to by *ps* is used to completely describe the current conversion state of the associated  
24413 character sequence. The implementation shall behave as if no function defined in this volume of  
24414 IEEE Std. 1003.1-200x calls *mbrtowc()*.24415 **XSI** The behavior of this function is affected by the *LC\_CTYPE* category of the current locale.24416 **RETURN VALUE**24417 The *mbrtowc()* function shall return the first of the following that applies:24418 **0** If the next *n* or fewer bytes complete the character that corresponds to the null  
24419 wide character (which is the value stored).24420 *positive* If the next *n* or fewer bytes complete a valid character (which is the value  
24421 stored); the value returned shall be the number of bytes that complete the  
24422 character.24423 **(size\_t)−2** If the next *n* bytes contribute to an incomplete but potentially valid character,  
24424 and all *n* bytes have been processed (no value is stored). When *n* has at least  
24425 the value of the {*MB\_CUR\_MAX*} macro, this case can only occur if *s* points at  
24426 a sequence of redundant shift sequences (for implementations with state-  
24427 dependent encodings).24428 **(size\_t)−1** If an encoding error occurs, in which case the next *n* or fewer bytes do not  
24429 contribute to a complete and valid character (no value is stored). In this case,  
24430 [*EILSEQ*] shall be stored in *errno* and the conversion state is undefined.24431 **ERRORS**24432 The *mbrtowc()* function may fail if:24433 **CX** [*EINVAL*] *ps* points to an object that contains an invalid conversion state.

- 24434 [EILSEQ] Invalid character sequence is detected.
- 24435 **EXAMPLES**
- 24436 None.
- 24437 **APPLICATION USAGE**
- 24438 None.
- 24439 **RATIONALE**
- 24440 None.
- 24441 **FUTURE DIRECTIONS**
- 24442 None.
- 24443 **SEE ALSO**
- 24444 *mbstinit()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**wchar.h**>
- 24445 **CHANGE HISTORY**
- 24446 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
- 24447 (E).
- 24448 **Issue 6**
- 24449 The *mbrtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24450 **NAME**

24451 mbsinit — determine conversion object status

24452 **SYNOPSIS**

24453 #include <wchar.h>

24454 int mbsinit(const mbstate\_t \*ps);

24455 **DESCRIPTION**

24456 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
24457 conflict between the requirements described here and the ISO C standard is unintentional. This  
24458 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

24459 If *ps* is not a null pointer, the *mbsinit()* function shall determine whether the object pointed to by  
24460 *ps* describes an initial conversion state.

24461 **RETURN VALUE**

24462 The *mbsinit()* function shall return non-zero if *ps* is a null pointer, or if the pointed-to object  
24463 describes an initial conversion state; otherwise, it shall return zero.

24464 If an **mbstate\_t** object is altered by any of the functions described as “restartable”, and is then  
24465 used with a different character sequence, or in the other conversion direction, or with a different  
24466 *LC\_CTYPE* category setting than on earlier function calls, the behavior is undefined.

24467 **ERRORS**

24468 No errors are defined.

24469 **EXAMPLES**

24470 None.

24471 **APPLICATION USAGE**

24472 The **mbstate\_t** object is used to describe the current conversion state from a particular character  
24473 sequence to a wide-character sequence (or *vice versa*) under the rules of a particular setting of the  
24474 *LC\_CTYPE* category of the current locale.

24475 The initial conversion state corresponds, for a conversion in either direction, to the beginning of  
24476 a new character sequence in the initial shift state. A zero valued **mbstate\_t** object is at least one  
24477 way to describe an initial conversion state. A zero valued **mbstate\_t** object can be used to initiate  
24478 conversion involving any character sequence, in any *LC\_CTYPE* category setting.

24479 **RATIONALE**

24480 None.

24481 **FUTURE DIRECTIONS**

24482 None.

24483 **SEE ALSO**

24484 *mbrlen()*, *mbrtowc()*, *wcrtomb()*, *mbsrtowcs()*, *wcsrtombs()*, the Base Definitions volume of  
24485 IEEE Std. 1003.1-200x, <wchar.h>

24486 **CHANGE HISTORY**

24487 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
24488 (E).

24489 **NAME**

24490 mbsrtowcs — convert a character string to a wide-character string (restartable)

24491 **SYNOPSIS**

24492 #include &lt;wchar.h&gt;

24493 size\_t mbsrtowcs(wchar\_t \*restrict *dst*, const char \*\*restrict *src*,  
24494 size\_t *len*, mbstate\_t \*restrict *ps*);24495 **DESCRIPTION**24496 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
24497 conflict between the requirements described here and the ISO C standard is unintentional. This  
24498 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.24499 The *mbsrtowcs()* function shall convert a sequence of characters, beginning in the conversion  
24500 state described by the object pointed to by *ps*, from the array indirectly pointed to by *src* into a  
24501 sequence of corresponding wide characters. If *dst* is not a null pointer, the converted characters  
24502 are stored into the array pointed to by *dst*. Conversion continues up to and including a  
24503 terminating null character, which is also stored. Conversion stops early in either of the following  
24504 cases:

- 24505
- A sequence of bytes is encountered that does not form a valid character.
  - *len* codes have been stored into the array pointed to by *dst* (and *dst* is not a null pointer).
- 24506

24507 Each conversion takes place as if by a call to the *mbrtowc()* function.24508 If *dst* is not a null pointer, the pointer object pointed to by *src* is assigned either a null pointer (if  
24509 conversion stopped due to reaching a terminating null character) or the address just past the last  
24510 character converted (if any). If conversion stopped due to reaching a terminating null character,  
24511 and if *dst* is not a null pointer, the resulting state described is the initial conversion state.24512 If *ps* is a null pointer, the *mbsrtowcs()* function uses its own internal **mbstate\_t** object, which is  
24513 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
24514 pointed to by *ps* is used to completely describe the current conversion state of the associated  
24515 character sequence. The implementation behaves as if no function defined in this volume of  
24516 IEEE Std. 1003.1-200x calls *mbsrtowcs()*.24517 **XSI** The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.24518 **RETURN VALUE**24519 If the input conversion encounters a sequence of bytes that do not form a valid character, an  
24520 encoding error occurs. In this case, the *mbsrtowcs()* function stores the value of the macro  
24521 [EILSEQ] in *errno* and shall return (**size\_t**)-1; the conversion state is undefined. Otherwise, it  
24522 shall return the number of characters successfully converted, not including the terminating null  
24523 (if any).24524 **ERRORS**24525 The *mbsrtowcs()* function may fail if:

- 24526 [EINVAL]
- ps*
- points to an object that contains an invalid conversion state.
- 
- 24527 [EILSEQ] Invalid character sequence is detected.

24528 **EXAMPLES**

24529 None.

24530 **APPLICATION USAGE**

24531 None.

24532 **RATIONALE**

24533 None.

24534 **FUTURE DIRECTIONS**

24535 None.

24536 **SEE ALSO**24537 *mbsinit()*, *mbrtowc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <wchar.h>24538 **CHANGE HISTORY**24539 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
24540 (E).24541 **Issue 6**24542 The *mbsrtowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24543 **NAME**

24544 mbstowcs — convert a character string to a wide-character string

24545 **SYNOPSIS**

24546 #include &lt;stdlib.h&gt;

24547 size\_t mbstowcs(wchar\_t \*restrict pwcs, const char \*restrict s,  
24548 size\_t n);24549 **DESCRIPTION**24550 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
24551 conflict between the requirements described here and the ISO C standard is unintentional. This  
24552 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.24553 The *mbstowcs()* function shall convert a sequence of characters that begins in the initial shift  
24554 state from the array pointed to by *s* into a sequence of corresponding wide-character codes and  
24555 stores not more than *n* wide-character codes into the array pointed to by *pwcs*. No characters  
24556 that follow a null byte (which is converted into a wide-character code with value 0) shall be  
24557 examined or converted. Each character is converted as if by a call to *mbtowc()*, except that the  
24558 shift state of *mbtowc()* is not affected.24559 No more than *n* elements shall be modified in the array pointed to by *pwcs*. If copying takes  
24560 place between objects that overlap, the behavior is undefined.24561 **XSI** The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale. If  
24562 *pwcs* is a null pointer, *mbstowcs()* shall return the length required to convert the entire array  
24563 regardless of the value of *n*, but no values are stored.24564 **RETURN VALUE**24565 **CX** If an invalid character is encountered, *mbstowcs()* shall return **(size\_t)-1** and may set *errno* to  
24566 indicate the error. Otherwise, *mbstowcs()* shall return the number of the array elements modified  
24567 (or required if *pwcs* is null), not including a terminating 0 code, if any. The array shall not be  
24568 zero-terminated if the value returned is *n*.24569 **ERRORS**24570 The *mbstowcs()* function may fail if:24571 **XSI** [EILSEQ] Invalid byte sequence is detected.24572 **EXAMPLES**

24573 None.

24574 **APPLICATION USAGE**

24575 None.

24576 **RATIONALE**

24577 None.

24578 **FUTURE DIRECTIONS**

24579 None.

24580 **SEE ALSO**24581 *mblen()*, *mbtowc()*, *wctomb()*, *wcstombs()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
24582 <stdlib.h>24583 **CHANGE HISTORY**

24584 First released in Issue 4. Aligned with the ISO C standard.

24585 **Issue 6**

24586

The *mbstowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24587 **NAME**

24588           mbtowc — convert a character to a wide-character code

24589 **SYNOPSIS**

24590           #include <stdlib.h>

24591           int mbtowc(wchar\_t \*restrict *pwc*, const char \*restrict *s*, size\_t *n*);

24592 **DESCRIPTION**

24593 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
24594 conflict between the requirements described here and the ISO C standard is unintentional. This  
24595 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

24596       If *s* is not a null pointer, *mbtowc()* shall determine the number of the bytes that constitute the  
24597 character pointed to by *s*. It then determines the wide-character code for the value of type  
24598 **wchar\_t** that corresponds to that character. (The value of the wide-character code corresponding  
24599 to the null byte is 0.) If the character is valid and *pwc* is not a null pointer, *mbtowc()* stores the  
24600 wide-character code in the object pointed to by *pwc*.

24601       The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
24602 state-dependent encoding, this function is placed into its initial state by a call for which its  
24603 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null  
24604 pointer cause the internal state of the function to be altered as necessary. A call with *s* as a null  
24605 pointer causes this function to return a non-zero value if encodings have state dependency, and  
24606 0 otherwise. If the implementation employs special bytes to change the shift state, these bytes do  
24607 not produce separate wide-character codes, but are grouped with an adjacent character.  
24608 Changing the *LC\_CTYPE* category causes the shift state of this function to be indeterminate. At  
24609 most *n* bytes of the array pointed to by *s* shall be examined.

24610       The implementation shall behave as if no function defined in this volume of  
24611 IEEE Std. 1003.1-200x calls *mbtowc()*.

24612 **RETURN VALUE**

24613       If *s* is a null pointer, *mbtowc()* shall return a non-zero or 0 value, if character encodings,  
24614 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mbtowc()*  
24615 shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the  
24616 **CX** converted character (if the next *n* or fewer bytes form a valid character), or return -1 and may  
24617 set *errno* to indicate the error (if they do not form a valid character).

24618       In no case shall the value returned be greater than *n* or the value of the {MB\_CUR\_MAX} macro.

24619 **ERRORS**

24620       The *mbtowc()* function may fail if:

24621 **XSI**       [EILSEQ]       Invalid character sequence is detected.

24622 **EXAMPLES**

24623       None.

24624 **APPLICATION USAGE**

24625       None.

24626 **RATIONALE**

24627       None.

24628 **FUTURE DIRECTIONS**

24629       None.



24630 **SEE ALSO**

24631 *mblen()*, *mbstowcs()*, *wctomb()*, *wcstombs()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
24632 **<stdlib.h>**

24633 **CHANGE HISTORY**

24634 First released in Issue 4. Aligned with the ISO C standard.

24635 **Issue 6**

24636 The *mbtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24637 **NAME**

24638 memccpy — copy bytes in memory

24639 **SYNOPSIS**

24640 XSI #include &lt;string.h&gt;

24641 void \*memccpy(void \*restrict *s1*, const void \*restrict *s2*,  
24642 int *c*, size\_t *n*);

24643

24644 **DESCRIPTION**

24645 The *memccpy()* function shall copy bytes from memory area *s2* into *s1*, stopping after the first  
24646 occurrence of byte *c* (converted to an **unsigned char**) is copied, or after *n* bytes are copied,  
24647 whichever comes first. If copying takes place between objects that overlap, the behavior is  
24648 undefined.

24649 **RETURN VALUE**

24650 The *memccpy()* function shall return a pointer to the byte after the copy of *c* in *s1*, or a null  
24651 pointer if *c* was not found in the first *n* bytes of *s2*.

24652 **ERRORS**

24653 No errors are defined.

24654 **EXAMPLES**

24655 None.

24656 **APPLICATION USAGE**24657 The *memccpy()* function does not check for the overflow of the receiving memory area.24658 **RATIONALE**

24659 None.

24660 **FUTURE DIRECTIONS**

24661 None.

24662 **SEE ALSO**

24663 The Base Definitions volume of IEEE Std. 1003.1-200x, &lt;string.h&gt;

24664 **CHANGE HISTORY**

24665 First released in Issue 1. Derived from Issue 1 of the SVID.

24666 **Issue 4**24667 The type of argument *s2* is changed from **void\*** to **const void\***.

24668 Reference to use of the <**memory.h**> header is removed from the APPLICATION USAGE  
24669 section.

24670 The FUTURE DIRECTIONS section is removed.

24671 **Issue 6**

24672 The **restrict** keyword is added to the *memccpy()* prototype for alignment with the  
24673 ISO/IEC 9899:1999 standard.

24674 **NAME**

24675 memchr — find byte in memory

24676 **SYNOPSIS**

24677 #include <string.h>

24678 void \*memchr(const void \*s, int c, size\_t n);

24679 **DESCRIPTION**

24680 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
24681 conflict between the requirements described here and the ISO C standard is unintentional. This  
24682 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

24683 The *memchr()* function shall locate the first occurrence of *c* (converted to an **unsigned char**) in  
24684 the initial *n* bytes (each interpreted as **unsigned char**) of the object pointed to by *s*.

24685 **RETURN VALUE**

24686 The *memchr()* function shall return a pointer to the located byte, or a null pointer if the byte does  
24687 not occur in the object.

24688 **ERRORS**

24689 No errors are defined.

24690 **EXAMPLES**

24691 None.

24692 **APPLICATION USAGE**

24693 None.

24694 **RATIONALE**

24695 None.

24696 **FUTURE DIRECTIONS**

24697 None.

24698 **SEE ALSO**

24699 The Base Definitions volume of IEEE Std. 1003.1-200x, <string.h>

24700 **CHANGE HISTORY**

24701 First released in Issue 1. Derived from Issue 1 of the SVID.

24702 **Issue 4**

24703 The APPLICATION USAGE section is removed.

24704 The following changes are incorporated for alignment with the ISO C standard:

- 24705 • The function is no longer marked as an extension.
- 24706 • The type of argument *s* is changed from **void\*** to **const void\***.

24707 **NAME**

24708        memcmp — compare bytes in memory

24709 **SYNOPSIS**

24710        #include &lt;string.h&gt;

24711        int memcmp(const void \*s1, const void \*s2, size\_t n);

24712 **DESCRIPTION**

24713 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
24714        conflict between the requirements described here and the ISO C standard is unintentional. This  
24715        volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

24716        The *memcmp()* function shall compare the first *n* bytes (each interpreted as **unsigned char**) of the  
24717        object pointed to by *s1* to the first *n* bytes of the object pointed to by *s2*.

24718        The sign of a non-zero return value shall be determined by the sign of the difference between the  
24719        values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the objects  
24720        being compared.

24721 **RETURN VALUE**

24722        The *memcmp()* function shall return an integer greater than, equal to, or less than 0, if the object  
24723        pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*, respectively.

24724 **ERRORS**

24725        No errors are defined.

24726 **EXAMPLES**

24727        None.

24728 **APPLICATION USAGE**

24729        None.

24730 **RATIONALE**

24731        None.

24732 **FUTURE DIRECTIONS**

24733        None.

24734 **SEE ALSO**

24735        The Base Definitions volume of IEEE Std. 1003.1-200x, &lt;string.h&gt;

24736 **CHANGE HISTORY**

24737        First released in Issue 1. Derived from Issue 1 of the SVID.

24738 **Issue 4**

24739        The RETURN VALUE section is clarified.

24740        The APPLICATION USAGE section is removed.

24741        The following changes are incorporated for alignment with the ISO C standard:

- 24742        • The function is no longer marked as an extension.
- 24743        • The type of arguments *s1* and *s2* are changed from **void\*** to **const void\***.

24744 **NAME**

24745           memcpy — copy bytes in memory

24746 **SYNOPSIS**

24747           #include <string.h>

24748           void \*memcpy(void \*restrict *s1*, const void \*restrict *s2*, size\_t *n*);

24749 **DESCRIPTION**

24750 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
24751 conflict between the requirements described here and the ISO C standard is unintentional. This  
24752 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

24753       The *memcpy()* function shall copy *n* bytes from the object pointed to by *s2* into the object pointed  
24754 to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

24755 **RETURN VALUE**

24756       The *memcpy()* function shall return *s1*; no return value is reserved to indicate an error.

24757 **ERRORS**

24758       No errors are defined.

24759 **EXAMPLES**

24760       None.

24761 **APPLICATION USAGE**

24762       The *memcpy()* function does not check for the overflowing of the receiving memory area.

24763 **RATIONALE**

24764       None.

24765 **FUTURE DIRECTIONS**

24766       None.

24767 **SEE ALSO**

24768       The Base Definitions volume of IEEE Std. 1003.1-200x, <**string.h**>

24769 **CHANGE HISTORY**

24770       First released in Issue 1. Derived from Issue 1 of the SVID.

24771 **Issue 4**

24772       Reference to use of the <**memory.h**> header is removed from the APPLICATION USAGE  
24773 section, and a note about overflow checking has been added.

24774       The FUTURE DIRECTIONS section is removed.

24775       The following changes are incorporated for alignment with the ISO C standard:

- 24776           • The function is no longer marked as an extension.
- 24777           • The type of argument *s2* is changed from **void\*** to **const void\***.

24778 **Issue 6**

24779       The *memcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24780 **NAME**

24781 memmove — copy bytes in memory with overlapping areas

24782 **SYNOPSIS**

24783 #include &lt;string.h&gt;

24784 void \*memmove(void \*s1, const void \*s2, size\_t n);

24785 **DESCRIPTION**

24786 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
24787 conflict between the requirements described here and the ISO C standard is unintentional. This  
24788 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

24789 The *memmove()* function shall copy *n* bytes from the object pointed to by *s2* into the object  
24790 pointed to by *s1*. Copying takes place as if the *n* bytes from the object pointed to by *s2* are first  
24791 copied into a temporary array of *n* bytes that does not overlap the objects pointed to by *s1* and  
24792 *s2*, and then the *n* bytes from the temporary array are copied into the object pointed to by *s1*.

24793 **RETURN VALUE**24794 The *memmove()* function shall return *s1*; no return value is reserved to indicate an error.24795 **ERRORS**

24796 No errors are defined.

24797 **EXAMPLES**

24798 None.

24799 **APPLICATION USAGE**

24800 None.

24801 **RATIONALE**

24802 None.

24803 **FUTURE DIRECTIONS**

24804 None.

24805 **SEE ALSO**24806 The Base Definitions volume of IEEE Std. 1003.1-200x, <**string.h**>24807 **CHANGE HISTORY**

24808 First released in Issue 4. Derived from the ANSI C standard.

24809 **NAME**

24810           memset — set bytes in memory

24811 **SYNOPSIS**

24812           #include <string.h>

24813           void \*memset(void \*s, int c, size\_t n);

24814 **DESCRIPTION**

24815 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
24816       conflict between the requirements described here and the ISO C standard is unintentional. This  
24817       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

24818       The *memset()* function shall copy *c* (converted to an **unsigned char**) into each of the first *n* bytes  
24819       of the object pointed to by *s*.

24820 **RETURN VALUE**

24821       The *memset()* function shall return *s*; no return value is reserved to indicate an error.

24822 **ERRORS**

24823       No errors are defined.

24824 **EXAMPLES**

24825       None.

24826 **APPLICATION USAGE**

24827       None.

24828 **RATIONALE**

24829       None.

24830 **FUTURE DIRECTIONS**

24831       None.

24832 **SEE ALSO**

24833       The Base Definitions volume of IEEE Std. 1003.1-200x, <**string.h**>

24834 **CHANGE HISTORY**

24835       First released in Issue 1. Derived from Issue 1 of the SVID.

24836 **Issue 4**

24837       The APPLICATION USAGE section is removed.

24838       The following change is incorporated for alignment with the ISO C standard:

- 24839
  - The function is no longer marked as an extension.

## 24840 NAME

24841 mkdir — make a directory

## 24842 SYNOPSIS

24843 #include &lt;sys/stat.h&gt;

24844 int mkdir(const char \*path, mode\_t mode);

## 24845 DESCRIPTION

24846 The *mkdir()* function creates a new directory with name *path*. The file permission bits of the new  
 24847 directory are initialized from *mode*. These file permission bits of the *mode* argument are modified  
 24848 by the process' file creation mask.

24849 When bits in *mode* other than the file permission bits are set, the meaning of these additional bits  
 24850 is implementation-defined.

24851 The directory's user ID is set to the process' effective user ID. The directory's group ID shall be  
 24852 set to the group ID of the parent directory or to the effective group ID of the process.

24853 The newly created directory shall be an empty directory.

24854 If *path* names a symbolic link, *mkdir()* shall fail and set *errno* to [EEXIST].

24855 Upon successful completion, *mkdir()* shall mark for update the *st\_atime*, *st\_ctime*, and *st\_mtime*  
 24856 fields of the directory. Also, the *st\_ctime* and *st\_mtime* fields of the directory that contains the  
 24857 new entry shall be marked for update.

## 24858 RETURN VALUE

24859 Upon successful completion, *mkdir()* shall return 0. Otherwise, -1 shall be returned, no directory  
 24860 shall be created, and *errno* shall be set to indicate the error.

## 24861 ERRORS

24862 The *mkdir()* function shall fail if:

24863 [EACCES] Search permission is denied on a component of the path prefix, or write  
 24864 permission is denied on the parent directory of the directory to be created.

24865 [EEXIST] The named file exists.

24866 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 24867 argument.

24868 [EMLINK] The link count of the parent directory would exceed {LINK\_MAX}.

24869 [ENAMETOOLONG]

24870 The length of the *path* argument exceeds {PATH\_MAX} or a path name  
 24871 component is longer than {NAME\_MAX}.

24872 [ENOENT] A component of the path prefix specified by *path* does not name an existing  
 24873 directory or *path* is an empty string.

24874 [ENOSPC] The file system does not contain enough space to hold the contents of the new  
 24875 directory or to extend the parent directory of the new directory.

24876 [ENOTDIR] A component of the path prefix is not a directory.

24877 [EROFS] The parent directory resides on a read-only file system.

24878 The *mkdir()* function may fail if:

24879 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 24880 resolution of the *path* argument.



24881 [ENAMETOOLONG]  
 24882 As a result of encountering a symbolic link in resolution of the *path* argument,  
 24883 the length of the substituted path name string exceeded {PATH\_MAX}.

#### 24884 EXAMPLES

##### 24885 **Creating a Directory**

24886 The following example shows how to create a directory named `/home/cnd/mod1`, with  
 24887 read/write/search permissions for owner and group, and with read/search permissions for  
 24888 others.

```
24889 #include <sys/types.h>
24890 #include <sys/stat.h>
24891
24892 int status;
24893 ...
24894 status = mkdir("/home/cnd/mod1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
```

#### 24894 APPLICATION USAGE

24895 None.

#### 24896 RATIONALE

24897 The `mkdir()` function originated in 4.2 BSD and was added to System V in Release 3.0.

24898 4.3 BSD detects [ENAMETOOLONG].

24899 See `getgroups()` about the group of a newly created directory.

#### 24900 FUTURE DIRECTIONS

24901 None.

#### 24902 SEE ALSO

24903 `umask()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<sys/stat.h>`, `<sys/types.h>`

#### 24904 CHANGE HISTORY

24905 First released in Issue 3.

24906 Entry included for alignment with the POSIX.1-1988 standard.

#### 24907 Issue 4

24908 The `<sys/types.h>` header is now marked as optional (OH); this header need not be included on  
 24909 XSI-conformant systems.

24910 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 24911 • The type of argument *path* is changed from `char*` to `const char*`.

24912 The following changes are incorporated for alignment with the FIPS requirements:

- 24913 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path  
 24914 name component is larger than {NAME\_MAX} is now defined as mandatory and marked as  
 24915 an extension.

#### 24916 Issue 4, Version 2

24917 The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- 24918 • It states that [ELOOP] is returned if too many symbolic links are encountered during path  
 24919 name resolution.
- 24920 • A second [ENAMETOOLONG] condition is defined that may report excessive length of an  
 24921 intermediate result of path name resolution of a symbolic link.

24922 **Issue 6**

24923 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

24924 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 24925 • The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.  
24926 This is since behavior may vary from one file system to another.

24927 The following new requirements on POSIX implementations derive from alignment with the  
24928 Single UNIX Specification:

- 24929 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
24930 required for conforming implementations of previous POSIX specifications, it was not  
24931 required for UNIX applications.
- 24932 • The [ELOOP] mandatory error condition is added.
- 24933 • A second [ENAMETOOLONG] is added as an optional error condition.

24934 The following changes were made to align with the IEEE P1003.1a draft standard:

- 24935 • The [ELOOP] optional error condition is added.

24936 **NAME**

24937 mkfifo — make a FIFO special file

24938 **SYNOPSIS**

24939 #include &lt;sys/stat.h&gt;

24940 int mkfifo(const char \*path, mode\_t mode);

24941 **DESCRIPTION**

24942 The *mkfifo()* function shall create a new FIFO special file named by the path name pointed to by  
 24943 *path*. The file permission bits of the new FIFO are initialized from *mode*. The file permission bits  
 24944 of the *mode* argument are modified by the process' file creation mask.

24945 When bits in *mode* other than the file permission bits are set, the effect is implementation-  
 24946 defined.

24947 If *path* names a symbolic link, *mkfifo()* shall fail and set *errno* to [EEXIST].

24948 The FIFO's user ID shall be set to the process' effective user ID. The FIFO's group ID shall be set  
 24949 to the group ID of the parent directory or to the effective group ID of the process.

24950 Upon successful completion, *mkfifo()* shall mark for update the *st\_atime*, *st\_ctime*, and *st\_mtime*  
 24951 fields of the file. Also, the *st\_ctime* and *st\_mtime* fields of the directory that contains the new  
 24952 entry shall be marked for update.

24953 **RETURN VALUE**

24954 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, no FIFO shall  
 24955 be created, and *errno* shall be set to indicate the error.

24956 **ERRORS**24957 The *mkfifo()* function shall fail if:

24958 [EACCES] A component of the path prefix denies search permission, or write permission  
 24959 is denied on the parent directory of the FIFO to be created.

24960 [EEXIST] The named file already exists.

24961 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 24962 argument.

24963 [ENAMETOOLONG]

24964 The length of the *path* argument exceeds {PATH\_MAX} or a path name  
 24965 component is longer than {NAME\_MAX}.

24966 [ENOENT] A component of the path prefix specified by *path* does not name an existing  
 24967 directory or *path* is an empty string.

24968 [ENOSPC] The directory that would contain the new file cannot be extended or the file  
 24969 system is out of file-allocation resources.

24970 [ENOTDIR] A component of the path prefix is not a directory.

24971 [EROFS] The named file resides on a read-only file system.

24972 The *mkfifo()* function may fail if:

24973 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 24974 resolution of the *path* argument.

24975 [ENAMETOOLONG]

24976 As a result of encountering a symbolic link in resolution of the *path* argument,  
 24977 the length of the substituted path name string exceeded {PATH\_MAX}.

24978 **EXAMPLES**24979 **Creating a FIFO File**

24980 The following example shows how to create a FIFO file named `/home/cnd/mod_done`, with  
 24981 read/write permissions for owner, and with read permissions for group and others.

```
24982 #include <sys/types.h>
24983 #include <sys/stat.h>
24984 int status;
24985 ...
24986 status = mkfifo("/home/cnd/mod_done", S_IWUSR | S_IRUSR |
24987 S_IRGRP | S_IROTH);
```

24988 **APPLICATION USAGE**

24989 None.

24990 **RATIONALE**

24991 The syntax of this function is intended to maintain compatibility with historical  
 24992 implementations of `mknod()`. The latter function was included in the 1984 `/usr/group` standard  
 24993 but only for use in creating FIFO special files. The `mknod()` function was originally excluded  
 24994 from the POSIX.1-1988 standard as implementation-defined and replaced by `mkdir()` and  
 24995 `mkfifo()`. The `mknod()` function is now included for alignment with the Single UNIX  
 24996 Specification.

24997 See `getgroups()` about the group of a newly created FIFO.

24998 **FUTURE DIRECTIONS**

24999 None.

25000 **SEE ALSO**

25001 `umask()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<sys/stat.h>`, `<sys/types.h>`

25002 **CHANGE HISTORY**

25003 First released in Issue 3.

25004 Entry included for alignment with the POSIX.1-1988 standard.

25005 **Issue 4**

25006 The `<sys/types.h>` header is now marked as optional (OH); this header need not be included on  
 25007 XSI-conformant systems.

25008 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 25009 • The type of argument `path` is changed from `char*` to `const char*`.
- 25010 • The description of [EACCES] is updated to indicate that this error is also returned if write  
 25011 permission is denied to the parent directory.

25012 The following changes are incorporated for alignment with the FIPS requirements:

- 25013 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path  
 25014 name component is larger than {NAME\_MAX} is now defined as mandatory and marked as  
 25015 an extension.

25016 **Issue 4, Version 2**

25017 The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- 25018 • It states that [ELOOP] is returned if too many symbolic links are encountered during path  
 25019 name resolution.

- 25020           • A second [ENAMETOOLONG] condition is defined that may report excessive length of an  
25021           intermediate result of path name resolution of a symbolic link.

25022 **Issue 6**

25023           In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

25024           The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 25025           • The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.  
25026           This is since behavior may vary from one file system to another.

25027           The following new requirements on POSIX implementations derive from alignment with the  
25028           Single UNIX Specification:

- 25029           • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
25030           required for conforming implementations of previous POSIX specifications, it was not  
25031           required for UNIX applications.

- 25032           • The [ELOOP] mandatory error condition is added.

- 25033           • A second [ENAMETOOLONG] is added as an optional error condition.

25034           The following changes were made to align with the IEEE P1003.1a draft standard:

- 25035           • The [ELOOP] optional error condition is added.

25036 **NAME**

25037 `mknod` — make a directory, a special or regular file

25038 **SYNOPSIS**

25039 XSI `#include <sys/stat.h>`

25040 `int mknod(const char *path, mode_t mode, dev_t dev);`

25041

25042 **DESCRIPTION**

25043 The `mknod()` function shall create a new file named by the path name to which the argument  
25044 `path` points.

25045 The file type for `path` is OR'ed into the `mode` argument, and the application shall select one of the  
25046 following symbolic constants:

25047

25048

25049

25050

25051

25052

25053

| Name    | Description                      |
|---------|----------------------------------|
| S_IFIFO | FIFO-special                     |
| S_IFCHR | Character-special (non-portable) |
| S_IFDIR | Directory (non-portable)         |
| S_IFBLK | Block-special (non-portable)     |
| S_IFREG | Regular (non-portable)           |

25054 The only portable use of `mknod()` is to create a FIFO-special file. If `mode` is not S\_IFIFO or `dev` is  
25055 not 0, the behavior of `mknod()` is unspecified.

25056 The permissions for the new file are OR'ed into the `mode` argument, and may be selected from  
25057 any combination of the following symbolic constants:

25058

25059

25060

25061

25062

25063

25064

25065

25066

25067

25068

25069

25070

25071

25072

25073

25074

| Name    | Description                                 |
|---------|---------------------------------------------|
| S_ISUID | Set user ID on execution.                   |
| S_ISGID | Set group ID on execution.                  |
| S_IRWXU | Read, write, or execute (search) by owner.  |
| S_IRUSR | Read by owner.                              |
| S_IWUSR | Write by owner.                             |
| S_IXUSR | Execute (search) by owner.                  |
| S_IRWXG | Read, write, or execute (search) by group.  |
| S_IRGRP | Read by group.                              |
| S_IWGRP | Write by group.                             |
| S_IXGRP | Execute (search) by group.                  |
| S_IRWXO | Read, write, or execute (search) by others. |
| S_IROTH | Read by others.                             |
| S_IWOTH | Write by others.                            |
| S_IXOTH | Execute (search) by others.                 |
| S_ISVTX | On directories, restricted deletion flag.   |

25075 The user ID of the file is initialized to the effective user ID of the process. The group ID of the file  
25076 is initialized to either the effective group ID of the process or the group ID of the parent  
25077 directory.

25078 The owner, group, and other permission bits of `mode` are modified by the file mode creation  
25079 mask of the process. The `mknod()` function clears each bit whose corresponding bit in the file  
25080 mode creation mask of the process is set.

25081 If *path* names a symbolic link, *mknod()* shall fail and set *errno* to [EEXIST].

25082 Upon successful completion, *mknod()* shall mark for update the *st\_atime*, *st\_ctime*, and *st\_mtime*  
 25083 fields of the file. Also, the *st\_ctime* and *st\_mtime* fields of the directory that contains the new  
 25084 entry shall be marked for update.

25085 Only a process with appropriate privileges may invoke *mknod()* for file types other than FIFO-  
 25086 special.

25087 **RETURN VALUE**

25088 Upon successful completion, *mknod()* shall return 0. Otherwise, it shall return -1, the new file  
 25089 shall not be created, and *errno* shall be set to indicate the error.

25090 **ERRORS**

25091 The *mknod()* function shall fail if:

|       |                |                                                                                                                                         |
|-------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| 25092 | [EACCES]       | A component of the path prefix denies search permission, or write permission<br>25093 is denied on the parent directory.                |
| 25094 | [EEXIST]       | The named file exists.                                                                                                                  |
| 25095 | [EINVAL]       | An invalid argument exists.                                                                                                             |
| 25096 | [EIO]          | An I/O error occurred while accessing the file system.                                                                                  |
| 25097 | [ELOOP]        | A loop exists in symbolic links encountered during resolution of the <i>path</i><br>25098 argument.                                     |
| 25099 | [ENAMETOOLONG] |                                                                                                                                         |
| 25100 |                | The length of a path name exceeds {PATH_MAX} or a path name component<br>25101 is longer than {NAME_MAX}.                               |
| 25102 | [ENOENT]       | A component of the path prefix specified by <i>path</i> does not name an existing<br>25103 directory or <i>path</i> is an empty string. |
| 25104 | [ENOSPC]       | The directory that would contain the new file cannot be extended or the file<br>25105 system is out of file allocation resources.       |
| 25106 | [ENOTDIR]      | A component of the path prefix is not a directory.                                                                                      |
| 25107 | [EPERM]        | The invoking process does not have appropriate privileges and the file type is<br>25108 not FIFO-special.                               |
| 25109 | [EROFS]        | The directory in which the file is to be created is located on a read-only file<br>25110 system.                                        |

25111 The *mknod()* function may fail if:

|       |                |                                                                                                                   |
|-------|----------------|-------------------------------------------------------------------------------------------------------------------|
| 25112 | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during<br>25113 resolution of the <i>path</i> argument.   |
| 25114 | [ENAMETOOLONG] |                                                                                                                   |
| 25115 |                | Path name resolution of a symbolic link produced an intermediate result<br>25116 whose length exceeds {PATH_MAX}. |

25117 **EXAMPLES**25118 **Creating a FIFO Special File**

25119 The following example shows how to create a FIFO special file named `/home/cnd/mod_done`,  
25120 with read/write permissions for owner, and with read permissions for group and others.

```
25121 #include <sys/types.h>
25122 #include <sys/stat.h>
25123 dev_t dev;
25124 int status;
25125 ...
25126 status = mknod("/home/cnd/mod_done", S_IFIFO | S_IWUSR |
25127 S_IRUSR | S_IRGRP | S_IROTH, dev);
```

25128 **APPLICATION USAGE**

25129 *mkfifo()* is preferred over this function for making FIFO special files.

25130 **RATIONALE**

25131 None.

25132 **FUTURE DIRECTIONS**

25133 None.

25134 **SEE ALSO**

25135 *chmod()*, *creat()*, *exec*, *mkdir()*, *mkfifo()*, *open()*, *stat()*, *umask()*, the Base Definitions volume of  
25136 IEEE Std. 1003.1-200x, `<sys/stat.h>`

25137 **CHANGE HISTORY**

25138 First released in Issue 4, Version 2.

25139 **Issue 5**

25140 Moved from X/OPEN UNIX extension to BASE.

25141 **Issue 6**

25142 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25143 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
25144 [ELOOP] error condition is added.



25145 **NAME**

25146 mkstemp — make a unique file name

25147 **SYNOPSIS**25148 XSI `#include <stdlib.h>`25149 `int mkstemp(char *template);`

25150

25151 **DESCRIPTION**

25152 The *mkstemp()* function shall replace the contents of the string pointed to by *template* by a unique  
25153 file name, and return a file descriptor for the file open for reading and writing. The function thus  
25154 prevents any possible race condition between testing whether the file exists and opening it for  
25155 use. The string in *template* should look like a file name with six trailing *Xs*; *mkstemp()* replaces  
25156 each *X* with a character from the portable file name character set. The characters are chosen such  
25157 that the resulting name does not duplicate the name of an existing file at the time of a call to  
25158 *mkstemp()*.

25159 **RETURN VALUE**

25160 Upon successful completion, *mkstemp()* shall return an open file descriptor. Otherwise,  $-1$  shall  
25161 be returned if no suitable file could be created.

25162 **ERRORS**

25163 No errors are defined.

25164 **EXAMPLES**25165 **Generating a File Name**

25166 The following example creates a file with a 10-character name beginning with the characters  
25167 "file" and opens the file for reading and writing. The value returned as the value of *fd* is a file  
25168 descriptor that identifies the file.

25169 `#include <stdlib.h>`25170 `...`25171 `char *template = "/tmp/fileXXXXXX";`25172 `int fd;`25173 `fd = mkstemp(template);`25174 **APPLICATION USAGE**

25175 It is possible to run out of letters.

25176 The *mkstemp()* function need not check to determine whether the file name part of *template*  
25177 exceeds the maximum allowable file name length.

25178 **RATIONALE**

25179 None.

25180 **FUTURE DIRECTIONS**

25181 None.

25182 **SEE ALSO**

25183 *getpid()*, *open()*, *tmpfile()*, *tmpnam()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
25184 `<stdlib.h>`

25185 **CHANGE HISTORY**

25186           First released in Issue 4, Version 2.

25187 **Issue 5**

25188           Moved from X/OPEN UNIX extension to BASE.

25189 **NAME**25190 mktemp — make a unique file name (**LEGACY**)25191 **SYNOPSIS**25192 XSI `#include <stdlib.h>`25193 `char *mktemp(char *template);`

25194

25195 **DESCRIPTION**

25196 The *mktemp()* function shall replace the contents of the string pointed to by *template* by a unique  
 25197 file name and return *template*. The application shall initialize *template* to be a file name with six  
 25198 trailing *Xs*; *mktemp()* shall replace each *X* with a single byte character from the portable file  
 25199 name character set.

25200 **RETURN VALUE**

25201 The *mktemp()* function shall return the pointer *template*. If a unique name cannot be created,  
 25202 *template* shall point to a null string.

25203 **ERRORS**

25204 No errors are defined.

25205 **EXAMPLES**25206 **Generating a File Name**

25207 The following example replaces the contents of the "template" string with a 10-character file  
 25208 name beginning with the characters "file" and returns a pointer to the "template" string  
 25209 that contains the new file name.

25210 `#include <stdlib.h>`25211 `...`25212 `char *template = "/tmp/fileXXXXXX";`25213 `char *ptr;`25214 `ptr = mktemp(template);`25215 **APPLICATION USAGE**

25216 Between the time a path name is created and the file opened, it is possible for some other process  
 25217 to create a file with the same name. The *mkstemp()* function avoids this problem and is preferred  
 25218 over this function.

25219 **RATIONALE**

25220 None.

25221 **FUTURE DIRECTIONS**

25222 This function may be withdrawn in a future version.

25223 **SEE ALSO**25224 *mkstemp()*, *tmpfile()*, *tmpnam()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<stdlib.h>`25225 **CHANGE HISTORY**

25226 First released in Issue 4, Version 2.

25227 **Issue 5**

25228 Moved from X/OPEN UNIX extension to BASE.

25229 **Issue 6**

25230 This function is marked LEGACY.

25231 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25232 **NAME**

25233 mktime — convert broken-down time into time since the Epoch

25234 **SYNOPSIS**

25235 #include &lt;time.h&gt;

25236 time\_t mktime(struct tm \*timeptr);

25237 **DESCRIPTION**

25238 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 25239 conflict between the requirements described here and the ISO C standard is unintentional. This  
 25240 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

25241 The *mktime()* function shall convert the broken-down time, expressed as local time, in the  
 25242 structure pointed to by *timeptr*, into a time since the Epoch value with the same encoding as that  
 25243 of the values returned by *time()*. The original values of the *tm\_wday* and *tm\_yday* components of  
 25244 the structure are ignored, and the original values of the other components are not restricted to  
 25245 the ranges described in <time.h>.

25246 **CX** A positive or 0 value for *tm\_isdst* shall cause *mktime()* to presume initially that Daylight Savings  
 25247 Time, respectively, is or is not in effect for the specified time. A negative value for *tm\_isdst* shall  
 25248 cause *mktime()* to attempt to determine whether Daylight Saving Time is in effect for the  
 25249 specified time.

25250 Local timezone information shall be set as though *mktime()* called *tzset()*.

25251 Upon successful completion, the values of the *tm\_wday* and *tm\_yday* components of the structure  
 25252 shall be set appropriately, and the other components are set to represent the specified time since  
 25253 the Epoch, but with their values forced to the ranges indicated in the <time.h> entry; the final  
 25254 value of *tm\_mday* shall not be set until *tm\_mon* and *tm\_year* are determined.

25255 **RETURN VALUE**

25256 The *mktime()* function shall return the specified time since the Epoch encoded as a value of type  
 25257 **time\_t**. If the time since the Epoch cannot be represented, the function shall return the value  
 25258 **(time\_t)-1**.

25259 **ERRORS**

25260 No errors are defined.

25261 **EXAMPLES**

25262 What day of the week is July 4, 2001?

25263 #include &lt;stdio.h&gt;

25264 #include &lt;time.h&gt;

25265 struct tm time\_str;

25266 char daybuf[20];

25267 int main(void)

25268 {

25269 time\_str.tm\_year = 2001 - 1900;

25270 time\_str.tm\_mon = 7 - 1;

25271 time\_str.tm\_mday = 4;

25272 time\_str.tm\_hour = 0;

25273 time\_str.tm\_min = 0;

25274 time\_str.tm\_sec = 1;

25275 time\_str.tm\_isdst = -1;

25276 if (mktime(&amp;time\_str) == -1)

```
25277 (void)puts("-unknown-");
25278 else {
25279 (void)strftime(daybuf, sizeof(daybuf), "%A", &time_str);
25280 (void)puts(daybuf);
25281 }
25282 return 0;
25283 }
```

**25284 APPLICATION USAGE**

25285 None.

**25286 RATIONALE**

25287 None.

**25288 FUTURE DIRECTIONS**

25289 None.

**25290 SEE ALSO**

25291 *asctime()*, *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *strftime()*, *strptime()*, *time()*, *utime()*,  
25292 the Base Definitions volume of IEEE Std. 1003.1-200x, <**time.h**>

**25293 CHANGE HISTORY**

25294 First released in Issue 3.

25295 Entry included for alignment with the POSIX.1-1988 standard and the ANSI C standard.

**25296 Issue 4**

25297 In the DESCRIPTION, a paragraph is added indicating the possible settings of *tm\_isdst*, and  
25298 reference to setting of *tm\_sec* for leap seconds or double leap seconds is removed (although this  
25299 functionality is still supported).

25300 In the EXAMPLES section, the sample code is updated to use ISO C standard syntax.

**25301 Issue 6**

25302 Extensions beyond the ISO C standard are now marked.

25303 **NAME**25304 mlock, munlock — lock or unlock a range of process address space (**REALTIME**)25305 **SYNOPSIS**

25306 MLR #include &lt;sys/mman.h&gt;

25307 int mlock(const void \* *addr*, size\_t *len*);25308 int munlock(const void \* *addr*, size\_t *len*);

25309

25310 **DESCRIPTION**

25311 The *mlock()* function shall cause those whole pages containing any part of the address space of  
 25312 the process starting at address *addr* and continuing for *len* bytes to be memory-resident until  
 25313 unlocked or until the process exits or *execs* another process image. The implementation may  
 25314 require that *addr* be a multiple of {PAGESIZE}.

25315 The *munlock()* function shall unlock those whole pages containing any part of the address space  
 25316 of the process starting at address *addr* and continuing for *len* bytes, regardless of how many  
 25317 times *mlock()* has been called by the process for any of the pages in the specified range. The  
 25318 implementation may require that *addr* be a multiple of the {PAGESIZE}.

25319 If any of the pages in the range specified to a call to *munlock()* are also mapped into the address  
 25320 spaces of other processes, any locks established on those pages by another process are  
 25321 unaffected by the call of this process to *munlock()*. If any of the pages in the range specified by a  
 25322 call to *munlock()* are also mapped into other portions of the address space of the calling process  
 25323 outside the range specified, any locks established on those pages via the other mappings are also  
 25324 unaffected by this call.

25325 Upon successful return from *mlock()*, pages in the specified range shall be locked and memory-  
 25326 resident. Upon successful return from *munlock()*, pages in the specified range shall be unlocked  
 25327 with respect to the address space of the process. Memory residency of unlocked pages is  
 25328 unspecified.

25329 The appropriate privilege is required to lock process memory with *mlock()*.

25330 **RETURN VALUE**

25331 Upon successful completion, the *mlock()* and *munlock()* functions shall return a value of zero.  
 25332 Otherwise, no change is made to any locks in the address space of the process, and the function  
 25333 shall return a value of  $-1$  and set *errno* to indicate the error.

25334 **ERRORS**

25335 The *mlock()* and *munlock()* functions shall fail if:

25336 [ENOMEM] Some or all of the address range specified by the *addr* and *len* arguments does  
 25337 not correspond to valid mapped pages in the address space of the process.

25338 The *mlock()* function shall fail if:

25339 [EAGAIN] Some or all of the memory identified by the operation could not be locked  
 25340 when the call was made.

25341 The *mlock()* and *munlock()* functions may fail if:

25342 [EINVAL] The *addr* argument is not a multiple of {PAGESIZE}.

25343 The *mlock()* function may fail if:

25344 [ENOMEM] Locking the pages mapped by the specified range would exceed an  
 25345 implementation-defined limit on the amount of memory that the process may  
 25346 lock.

25347 [EPERM] The calling process does not have the appropriate privilege to perform the  
25348 requested operation.

25349 **EXAMPLES**

25350 None.

25351 **APPLICATION USAGE**

25352 None.

25353 **RATIONALE**

25354 None.

25355 **FUTURE DIRECTIONS**

25356 None.

25357 **SEE ALSO**

25358 *exec*, *exit()*, *fork()*, *mlockall()*, *munmap()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
25359 `<sys/mman.h>`

25360 **CHANGE HISTORY**

25361 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25362 **Issue 6**

25363 The *mlock()* and *munlock()* functions are marked as part of the Range Memory Locking option.

25364 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25365 implementation does not support the Range Memory Locking option.



25366 **NAME**25367 mlockall, munlockall — lock/unlock the address space of a process (**REALTIME**)25368 **SYNOPSIS**25369 ML 

```
#include <sys/mman.h>
```

25370 

```
int mlockall(int flags);
```

25371 

```
int munlockall(void);
```

25372

25373 **DESCRIPTION**

25374 The *mlockall()* function shall cause all of the pages mapped by the address space of a process to  
 25375 be memory-resident until unlocked or until the process exits or *execs* another process image. The  
 25376 *flags* argument determines whether the pages to be locked are those currently mapped by the  
 25377 address space of the process, those that are mapped in the future, or both. The *flags* argument is  
 25378 constructed from the bitwise-inclusive OR of one or more of the following symbolic constants,  
 25379 defined in *<sys/mman.h>*:

25380 MCL\_CURRENT Lock all of the pages currently mapped into the address space of the process.

25381 MCL\_FUTURE Lock all of the pages that become mapped into the address space of the  
25382 process in the future, when those mappings are established.

25383 If MCL\_FUTURE is specified, and the automatic locking of future mappings eventually causes  
 25384 the amount of locked memory to exceed the amount of available physical memory or any other  
 25385 implementation-defined limit, the behavior is implementation-defined. The manner in which the  
 25386 implementation informs the application of these situations is also implementation-defined.

25387 The *munlockall()* function unlocks all currently mapped pages of the address space of the  
 25388 process. Any pages that become mapped into the address space of the process after a call to  
 25389 *munlockall()* shall not be locked, unless there is an intervening call to *mlockall()* specifying  
 25390 MCL\_FUTURE or a subsequent call to *mlockall()* specifying MCL\_CURRENT. If pages mapped  
 25391 into the address space of the process are also mapped into the address spaces of other processes  
 25392 and are locked by those processes, the locks established by the other processes are unaffected by  
 25393 a call by this process to *munlockall()*.

25394 Upon successful return from the *mlockall()* function that specifies MCL\_CURRENT, all currently  
 25395 mapped pages of the process' address space shall be memory-resident and locked. Upon return  
 25396 from the *munlockall()* function, all currently mapped pages of the process' address space shall be  
 25397 unlocked with respect to the process' address space. The memory residency of unlocked pages is  
 25398 unspecified.

25399 The appropriate privilege is required to lock process memory with *mlockall()*.25400 **RETURN VALUE**

25401 Upon successful completion, the *mlockall()* function shall return a value of zero. Otherwise, no  
 25402 additional memory shall be locked, and the function shall return a value of  $-1$  and set *errno* to  
 25403 indicate the error. The effect of failure of *mlockall()* on previously existing locks in the address  
 25404 space is unspecified.

25405 If it is supported by the implementation, the *munlockall()* function shall always return a value of  
 25406 zero. Otherwise, the function shall return a value of  $-1$  and set *errno* to indicate the error.

25407 **ERRORS**25408 The *mlockall()* function shall fail if:

25409 [EAGAIN] Some or all of the memory identified by the operation could not be locked  
 25410 when the call was made.

|       |          |                                                                                                                                        |
|-------|----------|----------------------------------------------------------------------------------------------------------------------------------------|
| 25411 | [EINVAL] | The <i>flags</i> argument is zero, or includes unimplemented flags.                                                                    |
| 25412 |          | The <i>mlockall()</i> function may fail if:                                                                                            |
| 25413 | [ENOMEM] | Locking all of the pages currently mapped into the address space of the                                                                |
| 25414 |          | process would exceed an implementation-defined limit on the amount of                                                                  |
| 25415 |          | memory that the process may lock.                                                                                                      |
| 25416 | [EPERM]  | The calling process does not have the appropriate privilege to perform the                                                             |
| 25417 |          | requested operation.                                                                                                                   |
| 25418 |          | <b>EXAMPLES</b>                                                                                                                        |
| 25419 |          | None.                                                                                                                                  |
| 25420 |          | <b>APPLICATION USAGE</b>                                                                                                               |
| 25421 |          | None.                                                                                                                                  |
| 25422 |          | <b>RATIONALE</b>                                                                                                                       |
| 25423 |          | None.                                                                                                                                  |
| 25424 |          | <b>FUTURE DIRECTIONS</b>                                                                                                               |
| 25425 |          | None.                                                                                                                                  |
| 25426 |          | <b>SEE ALSO</b>                                                                                                                        |
| 25427 |          | <i>exec</i> , <i>exit()</i> , <i>fork()</i> , <i>mlock()</i> , <i>munmap()</i> , the Base Definitions volume of IEEE Std. 1003.1-200x, |
| 25428 |          | < <i>sys/mman.h</i> >                                                                                                                  |
| 25429 |          | <b>CHANGE HISTORY</b>                                                                                                                  |
| 25430 |          | First released in Issue 5. Included for alignment with the POSIX Realtime Extension.                                                   |
| 25431 |          | <b>Issue 6</b>                                                                                                                         |
| 25432 |          | The <i>mlockall()</i> and <i>munlockall()</i> functions are marked as part of the Process Memory Locking                               |
| 25433 |          | option.                                                                                                                                |
| 25434 |          | The [ENOSYS] error condition has been removed as stubs need not be provided if an                                                      |
| 25435 |          | implementation does not support the Process Memory Locking option.                                                                     |

## 25436 NAME

25437 mmap — map pages of memory

## 25438 SYNOPSIS

25439 MF|SHM #include &lt;sys/mman.h&gt;

```
25440 void *mmap(void *addr, size_t len, int prot, int flags,
25441 int fildes, off_t off);
25442
```

## 25443 DESCRIPTION

25444 The *mmap()* function shall establish a mapping between a process' address space and a file, shared memory object, or typed memory object. The format of the call is as follows:

```
25446 pa=mmap(addr, len, prot, flags, fildes, off);
```

25447 The *mmap()* function establishes a mapping between the address space of the process at an address *pa* for *len* bytes to the memory object represented by the file descriptor *fildes* at offset *off* for *len* bytes. The value of *pa* is an implementation-defined function of the parameter *addr* and the values of *flags*, further described below. A successful *mmap()* call shall return *pa* as its result. The address range starting at *pa* and continuing for *len* bytes shall be legitimate for the possible (not necessarily current) address space of the process. The range of bytes starting at *off* and continuing for *len* bytes shall be legitimate for the possible (not necessarily current) offsets in the file, shared memory object, or typed memory object represented by *fildes*.

25455 TYM If *fildes* represents a typed memory object opened with either the `POSIX_TYPED_MEM_ALLOCATE` flag or the `POSIX_TYPED_MEM_ALLOCATE_CONTIG` flag, the memory object to be mapped shall be that portion of the typed memory object allocated by the implementation as specified below. In this case, if *off* is non-zero, the behavior of *mmap()* is undefined. If *fildes* refers to a valid typed memory object that is not accessible from the calling process, *mmap()* shall fail.

25461 The mapping established by *mmap()* replaces any previous mappings for those whole pages containing any part of the address space of the process starting at *pa* and continuing for *len* bytes.

25464 If the size of the mapped file changes after the call to *mmap()* as a result of some other operation on the mapped file, the effect of references to portions of the mapped region that correspond to added or removed portions of the file is unspecified.

25467 TYM The *mmap()* function is supported for regular files, shared memory objects, and typed memory objects. Support for any other type of file is unspecified.

25469 The parameter *prot* determines whether read, write, execute, or some combination of accesses are permitted to the data being mapped. The *prot* should be either `PROT_NONE` or the bitwise-inclusive OR of one or more of the other flags in the following table, defined in the header `<sys/mman.h>`.

25473

25474

25475

25476

25477

25478

| Symbolic Constant       | Description              |
|-------------------------|--------------------------|
| <code>PROT_READ</code>  | Data can be read.        |
| <code>PROT_WRITE</code> | Data can be written.     |
| <code>PROT_EXEC</code>  | Data can be executed.    |
| <code>PROT_NONE</code>  | Data cannot be accessed. |

25479 If an implementation cannot support the combination of access types specified by *prot*, the call to *mmap()* fails. An implementation may permit accesses other than those specified by *prot*; however, if the Memory Protection option is supported, the implementation shall not permit a

25480 MPR

25481

25482 write to succeed where PROT\_WRITE has not been set or shall not permit any access where  
 25483 PROT\_NONE alone has been set. The implementation shall support at least the following values  
 25484 of *prot*: PROT\_NONE, PROT\_READ, PROT\_WRITE, and the bitwise-inclusive OR of  
 25485 PROT\_READ and PROT\_WRITE. If the Memory Protection option is not supported, the result of  
 25486 any access that conflicts with the specified protection is undefined. The file descriptor *fd* shall  
 25487 have been opened with read permission, regardless of the protection options specified. If  
 25488 PROT\_WRITE is specified, the application shall ensure that it has opened the file descriptor  
 25489 *fd* with write permission unless MAP\_PRIVATE is specified in the *flags* parameter as  
 25490 described below.

25491 The parameter *flags* provides other information about the handling of the mapped data. The  
 25492 value of *flags* is the bitwise-inclusive OR of these options, defined in `<sys/mman.h>`:

25493

25494

25495

25496

25497

| Symbolic Constant | Description                    |
|-------------------|--------------------------------|
| MAP_SHARED        | Changes are shared.            |
| MAP_PRIVATE       | Changes are private.           |
| MAP_FIXED         | Interpret <i>addr</i> exactly. |

25498 Implementations that do not support the Memory Mapped Files option are not required to  
 25499 XSI support MAP\_PRIVATE. It is implementation-defined whether MAP\_FIXED shall be supported.  
 25500 MAP\_FIXED shall be supported on XSI-conformant systems.

25501 MAP\_SHARED and MAP\_PRIVATE describe the disposition of write references to the memory  
 25502 object. If MAP\_SHARED is specified, write references change the underlying object. If  
 25503 MAP\_PRIVATE is specified, modifications to the mapped data by the calling process shall be  
 25504 visible only to the calling process and shall not change the underlying object. It is unspecified  
 25505 whether modifications to the underlying object done after the MAP\_PRIVATE mapping is  
 25506 established are visible through the MAP\_PRIVATE mapping. Either MAP\_SHARED or  
 25507 MAP\_PRIVATE can be specified, but not both. The mapping type is retained across *fork*().

25508 TYM When *fd* represents a typed memory object opened with either the  
 25509 POSIX\_TYPED\_MEM\_ALLOCATE flag or the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG  
 25510 flag, *mmap*(*fd*) shall, if there are enough resources available, map *len* bytes allocated from the  
 25511 corresponding typed memory object which were not previously allocated to any process in any  
 25512 processor that may access that typed memory object. If there are not enough resources available,  
 25513 the function shall fail. If *fd* represents a typed memory object opened with the  
 25514 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG flag, these allocated bytes shall be contiguous  
 25515 within the typed memory object. If *fd* represents a typed memory object opened with the  
 25516 POSIX\_TYPED\_MEM\_ALLOCATE flag, these allocated bytes may be composed of non-  
 25517 contiguous fragments within the typed memory object. If *fd* represents a typed memory  
 25518 object opened with neither the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG flag nor the  
 25519 POSIX\_TYPED\_MEM\_ALLOCATE flag, *len* bytes starting at offset *off* within the typed memory  
 25520 object are mapped, exactly as when mapping a file or shared memory object. In this case, if two  
 25521 processes map an area of typed memory using the same *off* and *len* values and using file  
 25522 descriptors that refer to the same memory pool (either from the same port or from a different  
 25523 port), both processes shall map the same region of storage.

25524 When MAP\_FIXED is set in the *flags* argument, the implementation is informed that the value of  
 25525 *pa* shall be *addr*, exactly. If MAP\_FIXED is set, *mmap*(*fd*) may return MAP\_FAILED and set *errno* to  
 25526 [EINVAL]. If a MAP\_FIXED request is successful, the mapping established by *mmap*(*fd*) replaces  
 25527 any previous mappings for the process' pages in the range [*pa*,*pa+len*).

25528 When MAP\_FIXED is not set, the implementation uses *addr* in an implementation-defined  
 25529 manner to arrive at *pa*. The *pa* so chosen shall be an area of the address space that the

25530 implementation deems suitable for a mapping of *len* bytes to the file. All implementations  
 25531 interpret an *addr* value of 0 as granting the implementation complete freedom in selecting *pa*,  
 25532 subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a  
 25533 process address near which the mapping should be placed. When the implementation selects a  
 25534 value for *pa*, it never places a mapping at address 0, nor does it replace any extant mapping.

25535 The *off* argument is constrained to be aligned and sized according to the value returned by  
 25536 *sysconf()* when passed *\_SC\_PAGESIZE* or *\_SC\_PAGE\_SIZE*. When *MAP\_FIXED* is specified, the  
 25537 application shall ensure that the argument *addr* also meets these constraints. The  
 25538 implementation performs mapping operations over whole pages. Thus, while the argument *len*  
 25539 need not meet a size or alignment constraint, the implementation shall include, in any mapping  
 25540 operation, any partial page specified by the range [*pa*,*pa+len*).

25541 The system shall always zero-fill any partial page at the end of an object. Further, the system  
 25542 shall never write out any modified portions of the last page of an object which are beyond its  
 25543 end. References within the address range starting at *pa* and continuing for *len* bytes to whole  
 25544 MPR pages following the end of an object shall result in delivery of a SIGBUS signal.

25545 An implementation may generate SIGBUS signals when a reference would cause an error in the  
 25546 mapped object, such as out-of-space condition.

25547 The *mmap()* function adds an extra reference to the file associated with the file descriptor *fd*  
 25548 which is not removed by a subsequent *close()* on that file descriptor. This reference is removed  
 25549 when there are no more mappings to the file.

25550 The *st\_atime* field of the mapped file may be marked for update at any time between the *mmap()*  
 25551 call and the corresponding *munmap()* call. The initial read or write reference to a mapped region  
 25552 shall cause the file's *st\_atime* field to be marked for update if it has not already been marked for  
 25553 update.

25554 The *st\_ctime* and *st\_mtime* fields of a file that is mapped with *MAP\_SHARED* and *PROT\_WRITE*  
 25555 shall be marked for update at some point in the interval between a write reference to the  
 25556 mapped region and the next call to *msync()* with *MS\_ASYNC* or *MS\_SYNC* for that portion of  
 25557 the file by any process. If there is no such call and if the underlying file is modified as a result of  
 25558 a write reference, then these fields shall be marked for update at some time after the write  
 25559 reference.

25560 There may be implementation-defined limits on the number of memory regions that can be  
 25561 mapped (per process or per system).

25562 XSI If such a limit is imposed, whether the number of memory regions that can be mapped by a  
 25563 process is decreased by the use of *shmat()* is implementation-defined.

25564 If *mmap()* fails for reasons other than [EBADF], [EINVAL], or [ENOTSUP], some of the  
 25565 mappings in the address range starting at *addr* and continuing for *len* bytes may have been  
 25566 unmapped.

#### 25567 RETURN VALUE

25568 Upon successful completion, the *mmap()* function shall return the address at which the mapping  
 25569 was placed (*pa*); otherwise, it shall return a value of *MAP\_FAILED* and set *errno* to indicate the  
 25570 error. The symbol *MAP\_FAILED* is defined in the header *<sys/mman.h>*. No successful return  
 25571 from *mmap()* shall return the value *MAP\_FAILED*.

#### 25572 ERRORS

25573 The *mmap()* function shall fail if:

25574 [EACCES] The *fd* argument is not open for read, regardless of the protection specified,  
 25575 or *fd* is not open for write and *PROT\_WRITE* was specified for a

|                         |             |                                                                                                                                                                                                                                        |
|-------------------------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 25576                   |             | MAP_SHARED type mapping.                                                                                                                                                                                                               |
| 25577 ML<br>25578       | [EAGAIN]    | The mapping could not be locked in memory, if required by <i>mlockall()</i> , due to a lack of resources.                                                                                                                              |
| 25579                   | [EBADF]     | The <i>fildev</i> argument is not a valid open file descriptor.                                                                                                                                                                        |
| 25580<br>25581<br>25582 | [EINVAL]    | The <i>addr</i> argument (if MAP_FIXED was specified) or <i>off</i> is not a multiple of the page size as returned by <i>sysconf()</i> , or are considered invalid by the implementation.                                              |
| 25583<br>25584          | [EINVAL]    | The value of <i>flags</i> is invalid (neither MAP_PRIVATE nor MAP_SHARED is set).                                                                                                                                                      |
| 25585<br>25586          | [EMFILE]    | The number of mapped regions would exceed an implementation-defined limit (per process or per system).                                                                                                                                 |
| 25587                   | [ENODEV]    | The <i>fildev</i> argument refers to a file whose type is not supported by <i>mmap()</i> .                                                                                                                                             |
| 25588<br>25589<br>25590 | [ENOMEM]    | MAP_FIXED was specified, and the range [ <i>addr,addr+len</i> ) exceeds that allowed for the address space of a process; or, if MAP_FIXED was not specified and there is insufficient room in the address space to effect the mapping. |
| 25591 ML<br>25592       | [ENOMEM]    | The mapping could not be locked in memory, if required by <i>mlockall()</i> , because it would require more space than the system is able to supply.                                                                                   |
| 25593<br>25594          |             | MAP_FIXED or MAP_PRIVATE was specified in the <i>flags</i> argument and the implementation does not support this functionality.                                                                                                        |
| 25595 TYM<br>25596      | [ENOMEM]    | Not enough unallocated memory resources remain in the typed memory object designated by <i>fildev</i> to allocate <i>len</i> bytes.                                                                                                    |
| 25597<br>25598          | [ENOTSUP]   | The implementation does not support the combination of accesses requested in the <i>prot</i> argument.                                                                                                                                 |
| 25599                   | [ENXIO]     | Addresses in the range [ <i>off,off+len</i> ) are invalid for the object specified by <i>fildev</i> .                                                                                                                                  |
| 25600<br>25601          | [ENXIO]     | MAP_FIXED was specified in <i>flags</i> and the combination of <i>addr</i> , <i>len</i> , and <i>off</i> is invalid for the object specified by <i>fildev</i> .                                                                        |
| 25602 TYM<br>25603      | [ENXIO]     | The <i>fildev</i> argument refers to a typed memory object that is not accessible from the calling process.                                                                                                                            |
| 25604<br>25605          | [EOVERFLOW] | The file is a regular file and the value of <i>off</i> plus <i>len</i> exceeds the offset maximum established in the open file description associated with <i>fildev</i> .                                                             |

#### 25606 EXAMPLES

25607 None.

#### 25608 APPLICATION USAGE

25609 Use of *mmap()* may reduce the amount of memory available to other memory allocation  
25610 functions.

25611 Use of MAP\_FIXED may result in unspecified behavior in further use of *malloc()* and *shmat()*.  
25612 The use of MAP\_FIXED is discouraged, as it may prevent an implementation from making the  
25613 most effective use of resources.

25614 The application must ensure correct synchronization when using *mmap()* in conjunction with  
25615 any other file access method, such as *read()* and *write()*, standard input/output, and *shmat()*.

25616 The *mmap()* function allows access to resources via address space manipulations, instead of  
25617 *read()/write()*. Once a file is mapped, all a process has to do to access it is use the data at the

25618 address to which the file was mapped. So, using pseudo-code to illustrate the way in which an  
25619 existing program might be changed to use *mmap()*, the following:

```
25620 fildes = open(...)
25621 lseek(fildes, some_offset)
25622 read(fildes, buf, len)
25623 /* Use data in buf. */
```

25624 becomes:

```
25625 fildes = open(...)
25626 address = mmap(0, len, PROT_READ, MAP_PRIVATE, fildes, some_offset)
25627 /* Use data at address. */
```

25628 The [EINVAL] error above is marked EX because it is defined as an optional error in the POSIX  
25629 Realtime Extension.

### 25630 RATIONALE

25631 After considering several other alternatives, it was decided to adopt the *mmap()* definition found  
25632 in SVR4 for mapping memory objects into process address spaces. The SVR4 definition is  
25633 minimal, in that it describes only what has been built, and what appears to be necessary for a  
25634 general and portable mapping facility.

25635 Note that while *mmap()* was first designed for mapping files, it is actually a general-purpose  
25636 mapping facility. It can be used to map any appropriate object, such as memory, files, devices,  
25637 and so on, into the address space of a process.

25638 When a mapping is established, it is possible that the implementation may need to map more  
25639 than is requested into the address space of the process because of hardware requirements. An  
25640 application, however, cannot count on this behavior. Implementations that do not use a paged  
25641 architecture may simply allocate a common memory region and return the address of it; such  
25642 implementations probably do not allocate any more than is necessary. References past the end of  
25643 the requested area are unspecified.

25644 If an application requests a mapping that would overlay existing mappings in the process, it  
25645 might be desirable that an implementation detect this and inform the application. However, the  
25646 default, portable (not MAP\_FIXED) operation does not overlay existing mappings. On the other  
25647 hand, if the program specifies a fixed address mapping (which requires some implementation  
25648 knowledge to determine a suitable address, if the function is supported at all), then the program  
25649 is presumed to be successfully managing its own address space and should be trusted when it  
25650 asks to map over existing data structures. Furthermore, it is also desirable to make as few system  
25651 calls as possible, and it might be considered onerous to require an *munmap()* before an *mmap()*  
25652 to the same address range. This volume of IEEE Std. 1003.1-200x specifies that the new  
25653 mappings replace any existing mappings, following existing practice in this regard.

25654 It is not expected, when the Memory Protection option is supported, that all hardware  
25655 implementations are able to support all combinations of permissions at all addresses. When this  
25656 option is supported, implementations are required to disallow write access to mappings without  
25657 write permission and to disallow access to mappings without any access permission. Other than  
25658 these restrictions, implementations may allow access types other than those requested by the  
25659 application. For example, if the application requests only PROT\_WRITE, the implementation  
25660 may also allow read access. A call to *mmap()* fails if the implementation cannot support allowing  
25661 all the access requested by the application. For example, some implementations cannot support  
25662 a request for both write access and execute access simultaneously. All implementations  
25663 supporting the Memory Protection option must support requests for no access, read access,  
25664 write access, and both read and write access. Strictly conforming code must only rely on the  
25665 required checks. These restrictions allow for portability across a wide range of hardware.

25666 The MAP\_FIXED address treatment is likely to fail for non-page-aligned values and for certain  
25667 architecture-dependent address ranges. Conforming implementations cannot count on being  
25668 able to choose address values for MAP\_FIXED without utilizing non-portable, implementation-  
25669 defined knowledge. Nonetheless, MAP\_FIXED is provided as a standard interface conforming to  
25670 existing practice for utilizing such knowledge when it is available.

25671 Similarly, in order to allow implementations that do not support virtual addresses, support for  
25672 directly specifying any mapping addresses via MAP\_FIXED is not required and thus a  
25673 conforming application may not count on it.

25674 The MAP\_PRIVATE function can be implemented efficiently when memory protection hardware  
25675 is available. When such hardware is not available, implementations can implement such  
25676 “mappings” by simply making a real copy of the relevant data into process private memory,  
25677 though this tends to behave similarly to *read()*.

25678 The function has been defined to allow for many different models of using shared memory.  
25679 However, all uses are not equally portable across all machine architectures. In particular, the  
25680 *mmap()* function allows the system as well as the application to specify the address at which to  
25681 map a specific region of a memory object. The most portable way to use the function is always to  
25682 let the system choose the address, specifying NULL as the value for the argument *addr* and not  
25683 to specify MAP\_FIXED.

25684 If it is intended that a particular region of a memory object be mapped at the same address in a  
25685 group of processes (on machines where this is even possible), then MAP\_FIXED can be used to  
25686 pass in the desired mapping address. The system can still be used to choose the desired address  
25687 if the first such mapping is made without specifying MAP\_FIXED, and then the resulting  
25688 mapping address can be passed to subsequent processes for them to pass in via MAP\_FIXED.  
25689 The availability of a specific address range cannot be guaranteed, in general.

25690 The *mmap()* function can be used to map a region of memory that is larger than the current size  
25691 of the object. Memory access within the mapping but beyond the current end of the underlying  
25692 objects may result in SIGBUS signals being sent to the process. The reason for this is that the size  
25693 of the object can be manipulated by other processes and can change at any moment. The  
25694 implementation should tell the application that a memory reference is outside the object where  
25695 this can be detected; otherwise, written data may be lost and read data may not reflect actual  
25696 data in the object.

25697 Note that references beyond the end of the object do not extend the object as the new end cannot  
25698 be determined precisely by most virtual memory hardware. Instead, the size can be directly  
25699 manipulated by *ftruncate()*.

25700 Process memory locking does apply to shared memory regions, and the MEMLOCK\_FUTURE  
25701 argument to *memlockall()* can be relied upon to cause new shared memory regions to be  
25702 automatically locked.

25703 Existing implementations of *mmap()* return the value `-1` when unsuccessful. Since the casting of  
25704 this value to type `void*` cannot be guaranteed by the ISO C standard to be distinct from a  
25705 successful value, this volume of IEEE Std. 1003.1-200x defines the symbol MAP\_FAILED, which  
25706 a conforming implementation does not return as the result of a successful call.

#### 25707 FUTURE DIRECTIONS

25708 None.

#### 25709 SEE ALSO

25710 *exec*, *fcntl()*, *fork()*, *lockf()*, *msync()*, *munmap()*, *mprotect()*, *posix\_typed\_mem\_open()*, *shmat()*,  
25711 *sysconf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/mman.h>



25712 **CHANGE HISTORY**

25713 First released in Issue 4, Version 2.

25714 **Issue 5**

25715 Moved from X/OPEN UNIX extension to BASE.

25716 Aligned with *mmap()* in the POSIX Realtime Extension as follows:

- 25717 • The DESCRIPTION is extensively reworded.
- 25718 • The [EAGAIN] and [ENOTSUP] mandatory error conditions are added.
- 25719 • New cases of [ENOMEM] and [ENXIO] are added as mandatory error conditions.
- 25720 • The value returned on failure is the value of the constant MAP\_FAILED; this was previously
- 25721 defined as -1.

25722 Large File Summit extensions are added.

25723 **Issue 6**25724 The *mmap()* function is marked as part of the Memory Mapped Files option.25725 The Open Group corrigenda item U028/6 has been applied, changing (void \*)-1 to  
25726 MAP\_FAILED.25727 The following new requirements on POSIX implementations derive from alignment with the  
25728 Single UNIX Specification:

- 25729 • The DESCRIPTION is updated to described the use of MAP\_FIXED.
- 25730 • The DESCRIPTION is updated to describe the addition of an extra reference to the file
- 25731 associated with the file descriptor passed to *mmap()*.
- 25732 • The DESCRIPTION is updated to state that there may be implementation-defined limits on
- 25733 the number of memory regions that can be mapped.
- 25734 • The DESCRIPTION is updated to describe constraints on the alignment and size of the *off*
- 25735 argument.
- 25736 • The [EINVAL] and [EMFILE] error conditions are added.
- 25737 • The [EOVERFLOW] error condition is added. This change is to support large files.

25738 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 25739 • The DESCRIPTION is updated to describe the cases when MAP\_PRIVATE and MAP\_FIXED
- 25740 need not be supported.

25741 The following changes are made for alignment with IEEE Std. 1003.1j-2000:

- 25742 • Semantics for typed memory objects are added to the DESCRIPTION.
- 25743 • New [ENOMEM] and [ENXIO] errors are added to the ERRORS section.
- 25744 • The *posix\_typed\_mem\_open()* function is added to the SEE ALSO section.

25745 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25746 **NAME**

25747 modf, modff, modfl — decompose a floating-point number

25748 **SYNOPSIS**

25749 #include <math.h>

25750 double modf(double *x*, double \**iptr*);

25751 float modff(float *value*, float \**iptr*);

25752 long double modfl(long double *value*, long double \**iptr*);

25753 **DESCRIPTION**

25754 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 25755 conflict between the requirements described here and the ISO C standard is unintentional. This  
 25756 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

25757 These functions shall break the argument *x* into integral and fractional parts, each of which has  
 25758 the same sign as the argument. It stores the integral part as a double in the object pointed to by  
 25759 *iptr*.

25760 An application wishing to check for error situations should set *errno* to 0 before calling *modf()*. If  
 25761 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

25762 **RETURN VALUE**

25763 Upon successful completion, these functions shall return the signed fractional part of *x*.

25764 XSI If *x* is NaN, NaN shall be returned, *errno* may be set to [EDOM], and \**iptr* shall be set to NaN.

25765 If the correct value would cause underflow, 0 shall be returned and *errno* may be set to  
 25766 [ERANGE].

25767 **ERRORS**

25768 These functions may fail if:

25769 XSI [EDOM] The value of *x* is NaN.

25770 [ERANGE] The result underflows.

25771 XSI No other errors shall occur.

25772 **EXAMPLES**

25773 None.

25774 **APPLICATION USAGE**

25775 The *modf()* function computes the function result and \**iptr* such that:

25776 a = modf(x, &iptr)

25777 x == a+iptr

25778 allowing for the usual floating point inaccuracies.

25779 **RATIONALE**

25780 None.

25781 **FUTURE DIRECTIONS**

25782 None.

25783 **SEE ALSO**

25784 *frexp()*, *isnan()*, *ldexp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

25785 **CHANGE HISTORY**

25786 First released in Issue 1. Derived from Issue 1 of the SVID.

25787 **Issue 4**

25788 References to *matherr()* are removed.

25789 The name of the first argument is changed from *value* to *x*.

25790 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalize error handling in the mathematics functions.

25792 The return value specified for [EDOM] is marked as an extension.

25793 **Issue 5**

25794 The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

25796 **Issue 6**

25797 The *modff()* and *modfl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

25799 **NAME**

25800 mprotect — set protection of memory mapping

25801 **SYNOPSIS**

25802 MPR #include &lt;sys/mman.h&gt;

25803 int mprotect(void \*addr, size\_t len, int prot);

25804

25805 **DESCRIPTION**

25806 The *mprotect()* function shall change the access protections to be that specified by *prot* for those  
 25807 whole pages containing any part of the address space of the process starting at address *addr* and  
 25808 continuing for *len* bytes. The parameter *prot* determines whether read, write, execute, or some  
 25809 combination of accesses are permitted to the data being mapped. The *prot* argument should be  
 25810 either PROT\_NONE or the bitwise-inclusive OR of one or more of PROT\_READ, PROT\_WRITE,  
 25811 and PROT\_EXEC.

25812 If an implementation cannot support the combination of access types specified by *prot*, the call  
 25813 to *mprotect()* shall fail.

25814 An implementation may permit accesses other than those specified by *prot*; however, no  
 25815 implementation shall permit a write to succeed where PROT\_WRITE has not been set or shall  
 25816 permit any access where PROT\_NONE alone has been set. Implementations shall support at  
 25817 least the following values of *prot*: PROT\_NONE, PROT\_READ, PROT\_WRITE, and the bitwise-  
 25818 inclusive OR of PROT\_READ and PROT\_WRITE. If PROT\_WRITE is specified, the application  
 25819 shall ensure that it has opened the mapped objects in the specified address range with write  
 25820 permission, unless MAP\_PRIVATE was specified in the original mapping, regardless of whether  
 25821 the file descriptors used to map the objects have since been closed.

25822 The implementation shall require that *addr* be a multiple of the page size as returned by  
 25823 *sysconf()*.

25824 The behavior of this function is unspecified if the mapping was not established by a call to  
 25825 *mmap()*.

25826 When *mprotect()* fails for reasons other than [EINVAL], the protections on some of the pages in  
 25827 the range [*addr,addr+len*) may have been changed.

25828 **RETURN VALUE**

25829 Upon successful completion, *mprotect()* shall return 0; otherwise, it shall return -1 and set *errno*  
 25830 to indicate the error.

25831 **ERRORS**25832 The *mprotect()* function shall fail if:

25833 [EACCES] The *prot* argument specifies a protection that violates the access permission  
 25834 the process has to the underlying memory object.

25835 [EAGAIN] The *prot* argument specifies PROT\_WRITE over a MAP\_PRIVATE mapping  
 25836 and there are insufficient memory resources to reserve for locking the private  
 25837 page.

25838 [EINVAL] The *addr* argument is not a multiple of the page size as returned by *sysconf()*.

25839 [ENOMEM] Addresses in the range [*addr,addr+len*) are invalid for the address space of a  
 25840 process, or specify one or more pages which are not mapped.

25841 [ENOMEM] The *prot* argument specifies PROT\_WRITE on a MAP\_PRIVATE mapping, and  
 25842 it would require more space than the system is able to supply for locking the  
 25843 private pages, if required.

25844 [ENOTSUP] The implementation does not support the combination of accesses requested  
25845 in the *prot* argument.

25846 **EXAMPLES**

25847 None.

25848 **APPLICATION USAGE**

25849 The [EINVAL] error above is marked EX because it is defined as an optional error in the POSIX  
25850 Realtime Extension.

25851 **RATIONALE**

25852 None.

25853 **FUTURE DIRECTIONS**

25854 None.

25855 **SEE ALSO**

25856 *mmap()*, *sysconf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/mman.h>

25857 **CHANGE HISTORY**

25858 First released in Issue 4, Version 2.

25859 **Issue 5**

25860 Moved from X/OPEN UNIX extension to BASE.

25861 Aligned with *mprotect()* in the POSIX Realtime Extension as follows:

- 25862 • The DESCRIPTION is largely reworded.
- 25863 • [ENOTSUP] and a second form of [ENOMEM] are added as mandatory error conditions.
- 25864 • [EAGAIN] is moved from the optional to the mandatory error conditions.

25865 **Issue 6**

25866 The *mprotect()* function is marked as part of the Memory Protection option.

25867 The following new requirements on POSIX implementations derive from alignment with the  
25868 Single UNIX Specification:

- 25869 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of  
25870 the page size as returned by *sysconf()*.
- 25871 • The [EINVAL] error condition is added.

25872 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25873 **NAME**25874 mq\_close — close a message queue (**REALTIME**)25875 **SYNOPSIS**

25876 MSG #include &lt;mqueue.h&gt;

25877 int mq\_close(mqd\_t mqdes);

25878

25879 **DESCRIPTION**

25880 The *mq\_close()* function shall remove the association between the message queue descriptor,  
25881 *mqdes*, and its message queue. The results of using this message queue descriptor after  
25882 successful return from this *mq\_close()*, and until the return of this message queue descriptor  
25883 from a subsequent *mq\_open()*, are undefined.

25884 If the process has successfully attached a notification request to the message queue via this  
25885 *mqdes*, this attachment shall be removed, and the message queue is available for another process  
25886 to attach for notification.

25887 **RETURN VALUE**

25888 Upon successful completion, the *mq\_close()* function shall return a value of zero; otherwise, the  
25889 function shall return a value of -1 and set *errno* to indicate the error.

25890 **ERRORS**25891 The *mq\_close()* function shall fail if:25892 [EBADF] The *mqdes* argument is not a valid message queue descriptor.25893 **EXAMPLES**

25894 None.

25895 **APPLICATION USAGE**

25896 None.

25897 **RATIONALE**

25898 None.

25899 **FUTURE DIRECTIONS**

25900 None.

25901 **SEE ALSO**

25902 *mq\_open()*, *mq\_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of  
25903 IEEE Std. 1003.1-200x, <mqueue.h>

25904 **CHANGE HISTORY**

25905 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25906 **Issue 6**25907 The *mq\_close()* function is marked as part of the Message Passing option.

25908 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25909 implementation does not support the Message Passing option.

25910 **NAME**25911 mq\_getattr — get message queue attributes (**REALTIME**)25912 **SYNOPSIS**

25913 MSG #include &lt;mqqueue.h&gt;

25914 int mq\_getattr(mqd\_t mqdes, struct mq\_attr \*mqstat);

25915

25916 **DESCRIPTION**25917 The *mqdes* argument specifies a message queue descriptor.25918 The *mq\_getattr()* function is used to get status information and attributes of the message queue and the open message queue description associated with the message queue descriptor.25920 The results are returned in the **mq\_attr** structure referenced by the *mqstat* argument.25921 Upon return, the following members have the values associated with the open message queue description as set when the message queue was opened and as modified by subsequent *mq\_setattr()* calls: *mq\_flags*.25922 The following attributes of the message queue shall be returned as set at message queue creation: *mq\_maxmsg*, *mq\_msgsize*.25926 Upon return, the following members within the **mq\_attr** structure referenced by the *mqstat* argument are set to the current state of the message queue:25927 *mq\_curmsgs* The number of messages currently on the queue.25928 **RETURN VALUE**25929 Upon successful completion, the *mq\_getattr()* function shall return zero. Otherwise, the function shall return -1 and set *errno* to indicate the error.25930 **ERRORS**25931 The *mq\_getattr()* function shall fail if:25932 [EBADF] The *mqdes* argument is not a valid message queue descriptor.25933 **EXAMPLES**

25934 None.

25935 **APPLICATION USAGE**

25936 None.

25937 **RATIONALE**

25938 None.

25939 **FUTURE DIRECTIONS**

25940 None.

25941 **SEE ALSO**25942 *mq\_open()*, *mq\_send()*, *mq\_setattr()*, *mq\_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**mqqueue.h**>25943 **CHANGE HISTORY**

25944 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25945 **Issue 6**25946 The *mq\_getattr()* function is marked as part of the Message Passing option.

25947 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

25952           The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with  
25953           IEEE Std. 1003.1d-1999.



25954 **NAME**25955 mq\_notify — notify process that a message is available (**REALTIME**)25956 **SYNOPSIS**

25957 MSG #include &lt;mqueue.h&gt;

25958 int mq\_notify(mqd\_t mqdes, const struct sigevent \*notification);

25959

25960 **DESCRIPTION**

25961 If the argument *notification* is not NULL, this function registers the calling process to be notified  
 25962 of message arrival at an empty message queue associated with the specified message queue  
 25963 descriptor, *mqdes*. The notification specified by the *notification* argument shall be sent to the  
 25964 process when the message queue transitions from empty to non-empty. At any time, only one  
 25965 process may be registered for notification by a message queue. If the calling process or any other  
 25966 process has already registered for notification of message arrival at the specified message queue,  
 25967 subsequent attempts to register for that message queue fail.

25968 If *notification* is NULL and the process is currently registered for notification by the specified  
 25969 message queue, the existing registration is removed.

25970 When the notification is sent to the registered process, its registration shall be removed. The  
 25971 message queue shall then be available for registration.

25972 If a process has registered for notification of message arrival at a message queue and some  
 25973 thread is blocked in *mq\_receive()* waiting to receive a message when a message arrives at the  
 25974 queue, the arriving message satisfies the appropriate *mq\_receive()*. The resulting behavior is as if  
 25975 the message queue remains empty, and no notification is sent.

25976 **RETURN VALUE**

25977 Upon successful completion, the *mq\_notify()* function shall return a value of zero; otherwise, the  
 25978 function shall return a value of -1 and set *errno* to indicate the error.

25979 **ERRORS**25980 The *mq\_notify()* function shall fail if:25981 [EBADF] The *mqdes* argument is not a valid message queue descriptor. |

25982 [EBUSY] A process is already registered for notification by the message queue. |

25983 **EXAMPLES**

25984 None.

25985 **APPLICATION USAGE**

25986 None.

25987 **RATIONALE**

25988 None.

25989 **FUTURE DIRECTIONS**

25990 None.

25991 **SEE ALSO**

25992 *mq\_open()*, *mq\_send()*, *mq\_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base |  
 25993 Definitions volume of IEEE Std. 1003.1-200x, <mqueue.h> |

25994 **CHANGE HISTORY**

25995 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25996 **Issue 6**

25997 The *mq\_notify()* function is marked as part of the Message Passing option.

25998 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25999 implementation does not support the Message Passing option.

26000 The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with  
26001 IEEE Std. 1003.1d-1999.

26002 **NAME**26003 mq\_open — open a message queue (**REALTIME**)26004 **SYNOPSIS**

26005 MSG #include &lt;mqueue.h&gt;

26006 mqd\_t mq\_open(const char \*name, int oflag, ...);

26007

26008 **DESCRIPTION**

26009 The *mq\_open()* function shall establish the connection between a process and a message queue  
 26010 with a message queue descriptor. It creates an open message queue description that refers to the  
 26011 message queue, and a message queue descriptor that refers to that open message queue  
 26012 description. The message queue descriptor is used by other functions to refer to that message  
 26013 queue. The *name* argument points to a string naming a message queue. It is unspecified whether  
 26014 the name appears in the file system and is visible to other functions that take path names as  
 26015 arguments. The *name* argument conforms to the construction rules for a path name. If *name*  
 26016 begins with the slash character, then processes calling *mq\_open()* with the same value of *name*  
 26017 refer to the same message queue object, as long as that name has not been removed. If *name* does  
 26018 not begin with the slash character, the effect is implementation-defined. The interpretation of  
 26019 slash characters other than the leading slash character in *name* is implementation-defined. If the  
 26020 *name* argument is not the name of an existing message queue and creation is not requested,  
 26021 *mq\_open()* shall fail and return an error.

26022 A message queue descriptor may be implemented using a file descriptor, in which case  
 26023 applications can open up to at least {OPEN\_MAX} file and message queues.

26024 The *oflag* argument requests the desired receive and/or send access to the message queue. The  
 26025 requested access permission to receive messages or send messages is granted if the calling  
 26026 process would be granted read or write access, respectively, to an equivalently protected file.

26027 The value of *oflag* is the bitwise-inclusive OR of values from the following list. Applications  
 26028 specify exactly one of the first three values (access modes) below in the value of *oflag*:

26029 **O\_RDONLY** Open the message queue for receiving messages. The process can use the  
 26030 returned message queue descriptor with *mq\_receive()*, but not *mq\_send()*. A  
 26031 message queue may be open multiple times in the same or different processes  
 26032 for receiving messages.

26033 **O\_WRONLY** Open the queue for sending messages. The process can use the returned  
 26034 message queue descriptor with *mq\_send()* but not *mq\_receive()*. A message  
 26035 queue may be open multiple times in the same or different processes for  
 26036 sending messages.

26037 **O\_RDWR** Open the queue for both receiving and sending messages. The process can use  
 26038 any of the functions allowed for **O\_RDONLY** and **O\_WRONLY**. A message  
 26039 queue may be open multiple times in the same or different processes for  
 26040 sending messages.

26041 Any combination of the remaining flags may be specified in the value of *oflag*:

26042 **O\_CREAT** This option is used to create a message queue, and it requires two additional  
 26043 arguments: *mode*, which is of type **mode\_t**, and *attr*, which is a pointer to a  
 26044 **mq\_attr** structure. If the path name *name* has already been used to create a  
 26045 message queue that still exists, then this flag has no effect, except as noted  
 26046 under **O\_EXCL**. Otherwise, a message queue is created without any messages  
 26047 in it. The user ID of the message queue is set to the effective user ID of the  
 26048 process, and the group ID of the message queue is set to the effective group ID

|       |                     |                                                                                                          |
|-------|---------------------|----------------------------------------------------------------------------------------------------------|
| 26049 |                     | of the process. The file permission bits are set to the value of <i>mode</i> . When bits                 |
| 26050 |                     | in <i>mode</i> other than file permission bits are set, the effect is implementation-                    |
| 26051 |                     | defined. If <i>attr</i> is NULL, the message queue is created with implementation-                       |
| 26052 |                     | defined default message queue attributes. If <i>attr</i> is non-NULL and the calling                     |
| 26053 |                     | process has the appropriate privilege on <i>name</i> , the message queue <i>mq_maxmsg</i>                |
| 26054 |                     | and <i>mq_msgsize</i> attributes are set to the values of the corresponding members                      |
| 26055 |                     | in the <b>mq_attr</b> structure referred to by <i>attr</i> . If <i>attr</i> is non-NULL, but the calling |
| 26056 |                     | process does not have the appropriate privilege on <i>name</i> , the <i>mq_open()</i>                    |
| 26057 |                     | function shall fail and return an error without creating the message queue.                              |
| 26058 | O_EXCL              | If O_EXCL and O_CREAT are set, <i>mq_open()</i> fails if the message queue <i>name</i>                   |
| 26059 |                     | exists. The check for the existence of the message queue and the creation of                             |
| 26060 |                     | the message queue if it does not exist shall be atomic with respect to other                             |
| 26061 |                     | threads executing <i>mq_open()</i> naming the same <i>name</i> with O_EXCL and                           |
| 26062 |                     | O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is                                      |
| 26063 |                     | undefined.                                                                                               |
| 26064 | O_NONBLOCK          | The setting of this flag is associated with the open message queue description                           |
| 26065 |                     | and determines whether a <i>mq_send()</i> or <i>mq_receive()</i> waits for resources or                  |
| 26066 |                     | messages that are not currently available, or fails with <i>errno</i> set to [EAGAIN];                   |
| 26067 |                     | see <i>mq_send()</i> and <i>mq_receive()</i> for details.                                                |
| 26068 |                     | The <i>mq_open()</i> function does not add or remove messages from the queue.                            |
| 26069 | <b>RETURN VALUE</b> |                                                                                                          |
| 26070 |                     | Upon successful completion, the function shall return a message queue descriptor; otherwise,             |
| 26071 |                     | the function shall return ( <b>mqd_t</b> )−1 and set <i>errno</i> to indicate the error.                 |
| 26072 | <b>ERRORS</b>       |                                                                                                          |
| 26073 |                     | The <i>mq_open()</i> function shall fail if:                                                             |
| 26074 | [EACCES]            | The message queue exists and the permissions specified by <i>oflag</i> are denied, or                    |
| 26075 |                     | the message queue does not exist and permission to create the message queue                              |
| 26076 |                     | is denied.                                                                                               |
| 26077 | [EEXIST]            | O_CREAT and O_EXCL are set and the named message queue already exists.                                   |
| 26078 | [EINTR]             | The <i>mq_open()</i> function was interrupted by a signal.                                               |
| 26079 | [EINVAL]            | The <i>mq_open()</i> function is not supported for the given name.                                       |
| 26080 | [EINVAL]            | O_CREAT was specified in <i>oflag</i> , the value of <i>attr</i> is not NULL, and either                 |
| 26081 |                     | <i>mq_maxmsg</i> or <i>mq_msgsize</i> was less than or equal to zero.                                    |
| 26082 | [EMFILE]            | Too many message queue descriptors or file descriptors are currently in use by                           |
| 26083 |                     | this process.                                                                                            |
| 26084 | [ENAMETOOLONG]      |                                                                                                          |
| 26085 |                     | The length of the <i>name</i> argument exceeds {PATH_MAX} or a path name                                 |
| 26086 |                     | component is longer than {NAME_MAX}.                                                                     |
| 26087 | [ENFILE]            | Too many message queues are currently open in the system.                                                |
| 26088 | [ENOENT]            | O_CREAT is not set and the named message queue does not exist.                                           |
| 26089 | [ENOSPC]            | There is insufficient space for the creation of the new message queue.                                   |

26090 **EXAMPLES**

26091 None.

26092 **APPLICATION USAGE**

26093 None.

26094 **RATIONALE**

26095 None.

26096 **FUTURE DIRECTIONS**

26097 None.

26098 **SEE ALSO**

26099 *mq\_close()*, *mq\_getattr()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*, *mq\_timedreceive()*, *mq\_timedsend()*,  
26100 *mq\_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of  
26101 IEEE Std. 1003.1-200x, <mqqueue.h>

26102 **CHANGE HISTORY**

26103 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26104 **Issue 6**26105 The *mq\_open()* function is marked as part of the Message Passing option.

26106 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
26107 implementation does not support the Message Passing option.

26108 The *mq\_timedreceive()* and *mq\_timedsend()* functions are added to the SEE ALSO section for  
26109 alignment with IEEE Std. 1003.1d-1999.

26110 The DESCRIPTION of O\_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

## 26111 NAME

26112 mq\_receive, mq\_timedreceive — receive a message from a message queue (**REALTIME**)

## 26113 SYNOPSIS

26114 MSG #include <mqueue.h>

```
26115 ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
26116 unsigned *msg_prio);
```

26117

26118 MSG TMO #include <mqueue.h>

26119 #include <time.h>

```
26120 ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
26121 size_t msg_len, unsigned *restrict msg_prio,
26122 const struct timespec *restrict abs_timeout);
```

26123

## 26124 DESCRIPTION

26125 The *mq\_receive()* function is used to receive the oldest of the highest priority message(s) from the  
 26126 message queue specified by *mqdes*. If the size of the buffer in bytes, specified by the *msg\_len*  
 26127 argument, is less than the *mq\_msgsize* attribute of the message queue, the function shall fail and  
 26128 return an error. Otherwise, the selected message is removed from the queue and copied to the  
 26129 buffer pointed to by the *msg\_ptr* argument.

26130 If the value of *msg\_len* is greater than {SSIZE\_MAX}, the result is implementation-defined.

26131 If the argument *msg\_prio* is not NULL, the priority of the selected message is stored in the  
 26132 location referenced by *msg\_prio*.

26133 If the specified message queue is empty and O\_NONBLOCK is not set in the message queue  
 26134 description associated with *mqdes*, *mq\_receive()* blocks until a message is enqueued on the  
 26135 message queue or until *mq\_receive()* is interrupted by a signal. If more than one thread is waiting  
 26136 to receive a message when a message arrives at an empty queue and the Priority Scheduling  
 26137 option is supported, then the thread of highest priority that has been waiting the longest shall be  
 26138 selected to receive the message. Otherwise, it is unspecified which waiting thread receives the  
 26139 message. If the specified message queue is empty and O\_NONBLOCK is set in the message  
 26140 queue description associated with *mqdes*, no message is removed from the queue, and  
 26141 *mq\_receive()* shall return an error.

26142 TMO The *mq\_timedreceive()* function is used to receive the oldest of the highest priority messages from  
 26143 the message queue specified by *mqdes* as in the *mq\_receive()* function. However, if  
 26144 O\_NONBLOCK was not specified when the message queue was opened via the *mq\_open()*  
 26145 function, and no message exists on the queue to satisfy the receive, the wait for such a message  
 26146 will be terminated when the specified timeout expires. If O\_NONBLOCK is set, this function  
 26147 shall behave identically to *mq\_receive()*.

26148 The timeout expires when the absolute time specified by *abs\_timeout* passes, as measured by the  
 26149 clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 26150 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already been passed at the time  
 26151 of the call. If the Timers option is supported, the timeout is based on the CLOCK\_REALTIME  
 26152 clock; if the Timers option is not supported, the timeout is based on the system clock as returned  
 26153 by the *time()* function. The resolution of the timeout is the resolution of the clock on which it is  
 26154 based. The *timespec* argument is defined as a structure in the <time.h> header.

26155 Under no circumstance shall the operation fail with a timeout if a message can be removed from  
 26156 the message queue immediately. The validity of the *abs\_timeout* parameter need not be checked  
 26157 if a message can be removed from the message queue immediately.

26158 **RETURN VALUE**

26159 TMO Upon successful completion, the *mq\_receive()* and *mq\_timedreceive()* functions shall return the  
 26160 length of the selected message in bytes and the message shall be removed from the queue.  
 26161 Otherwise, no message shall be removed from the queue, the functions shall return a value of -1,  
 26162 and set *errno* to indicate the error.

26163 **ERRORS**

26164 TMO The *mq\_receive()* and *mq\_timedreceive()* functions shall fail if:

26165 [EAGAIN] O\_NONBLOCK was set in the message description associated with *mqdes*,  
 26166 and the specified message queue is empty.

26167 [EBADF] The *mqdes* argument is not a valid message queue descriptor open for reading.

26168 [EMSGSIZE] The specified message buffer size, *msg\_len*, is less than the message size  
 26169 attribute of the message queue.

26170 TMO [EINTR] The *mq\_receive()* or *mq\_timedreceive()* operation was interrupted by a signal.

26171 TMO [EINVAL] The process or thread would have blocked, and the *abs\_timeout* parameter  
 26172 specified a nanoseconds field value less than zero or greater than or equal to  
 26173 1 000 million.

26174 TMO [ETIMEDOUT] The O\_NONBLOCK flag was not set when the message queue was opened,  
 26175 but no message arrived on the queue before the specified timeout expired.

26176 TMO The *mq\_receive()* and *mq\_timedreceive()* functions may fail if:

26177 [EBADMSG] The implementation has detected a data corruption problem with the  
 26178 message.

26179 **EXAMPLES**

26180 None.

26181 **APPLICATION USAGE**

26182 None.

26183 **RATIONALE**

26184 None.

26185 **FUTURE DIRECTIONS**

26186 None.

26187 **SEE ALSO**

26188 *mq\_open()*, *mq\_send()*, *mq\_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, *time()*, the Base  
 26189 Definitions volume of IEEE Std. 1003.1-200x, <**mqqueue.h**>, <**time.h**>

26190 **CHANGE HISTORY**

26191 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26192 **Issue 6**

26193 The *mq\_receive()* function is marked as part of the Message Passing option.

26194 The Open Group corrigenda item U021/4 has been applied. The DESCRIPTION is changed to  
 26195 refer to *msg\_len* rather than *maxsize*.

26196 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 26197 implementation does not support the Message Passing option.

26198 The following new requirements on POSIX implementations derive from alignment with the  
 26199 Single UNIX Specification:

- 26200           • In this function it is possible for the return value to exceed the range of the type **ssize\_t** (since
- 26201           **size\_t** has a larger range of positive values than **ssize\_t**). A sentence restricting the size of
- 26202           the **size\_t** object is added to the description to resolve this conflict.
  
- 26203           The *mq\_timedreceive()* function is added for alignment with IEEE Std. 1003.1d-1999.
  
- 26204           The **restrict** keyword is added to the *mq\_timedreceive()* prototype for alignment with the
- 26205           ISO/IEC 9899:1999 standard.
  
- 26206           IEEE PASC Interpretation 1003.1 #109 is applied, correcting the return type for *mq\_timedreceive()*
- 26207           from **int** to **ssize\_t**.



## 26208 NAME

26209 mq\_send, mq\_timedsend — send a message to a message queue (REALTIME)

## 26210 SYNOPSIS

26211 MSG #include &lt;mqueue.h&gt;

26212 int mq\_send(mqd\_t mqdes, const char \*msg\_ptr, size\_t msg\_len,  
26213 unsigned msg\_prio);

26214

26215 MSG TMO #include &lt;mqueue.h&gt;

26216 #include &lt;time.h&gt;

26217 int mq\_timedsend(mqd\_t mqdes, const char \*msg\_ptr, size\_t msg\_len,  
26218 unsigned msg\_prio, const struct timespec \*abs\_timeout);

26219

## 26220 DESCRIPTION

26221 The *mq\_send()* function shall add the message pointed to by the argument *msg\_ptr* to the  
26222 message queue specified by *mqdes*. The *msg\_len* argument specifies the length of the message in  
26223 bytes pointed to by *msg\_ptr*. The value of *msg\_len* is less than or equal to the *mq\_msgsize*  
26224 attribute of the message queue, or *mq\_send()* shall fail.

26225 If the specified message queue is not full, *mq\_send()* behaves as if the message shall be inserted  
26226 into the message queue at the position indicated by the *msg\_prio* argument. A message with a  
26227 larger numeric value of *msg\_prio* shall be inserted before messages with lower values of  
26228 *msg\_prio*. A message shall be inserted after other messages in the queue, if any, with equal  
26229 *msg\_prio*. The value of *msg\_prio* shall be less than {MQ\_PRIO\_MAX}.

26230 If the specified message queue is full and O\_NONBLOCK is not set in the message queue  
26231 description associated with *mqdes*, *mq\_send()* shall block until space becomes available to  
26232 enqueue the message, or until *mq\_send()* is interrupted by a signal. If more than one thread is  
26233 waiting to send when space becomes available in the message queue and the Priority Scheduling  
26234 option is supported, then the thread of the highest priority that has been waiting the longest  
26235 shall be unblocked to send its message. Otherwise, it is unspecified which waiting thread is  
26236 unblocked. If the specified message queue is full and O\_NONBLOCK is set in the message  
26237 queue description associated with *mqdes*, the message shall not be queued and *mq\_send()* shall  
26238 return an error.

26239 TMO The *mq\_timedsend()* function adds a message to the message queue specified by *mqdes* in the  
26240 manner defined for the *mq\_send()* function. However, if the specified message queue is full and  
26241 O\_NONBLOCK is not set in the message queue description associated with *mqdes*, the wait for  
26242 sufficient room in the queue shall be terminated when the specified timeout expires. If  
26243 O\_NONBLOCK is set in the message queue description, this function shall behave identically to  
26244 *mq\_send()*.

26245 The timeout expires when the absolute time specified by *abs\_timeout* passes, as measured by the  
26246 clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
26247 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already been passed at the time  
26248 of the call. If the Timers option is supported, the timeout is based on the CLOCK\_REALTIME  
26249 clock; if the Timers option is not supported, the timeout is based on the system clock as returned  
26250 by the *time()* function. The resolution of the timeout is the resolution of the clock on which it is  
26251 based. The *timespec* argument is defined as a structure in the <time.h> header.

26252 Under no circumstance shall the operation fail with a timeout if there is sufficient room in the  
26253 queue to add the message immediately. The validity of the *abs\_timeout* parameter need not be  
26254 checked when there is sufficient room in the queue.

26255 **RETURN VALUE**

26256 TMO Upon successful completion, the `mq_send()` and `mq_timedsend()` functions shall return a value of  
 26257 zero. Otherwise, no message shall be enqueued, the functions shall return `-1`, and `errno` shall be  
 26258 set to indicate the error.

26259 **ERRORS**

26260 TMO The `mq_send()` and `mq_timedsend()` functions shall fail if:

26261 [EAGAIN] The `O_NONBLOCK` flag is set in the message queue description associated  
 26262 with `mqdes`, and the specified message queue is full.

26263 [EBADF] The `mqdes` argument is not a valid message queue descriptor open for writing.

26264 TMO [EINTR] A signal interrupted the call to `mq_send()` or `mq_timedsend()`.

26265 [EINVAL] The value of `msg_prio` was outside the valid range.

26266 TMO [EINVAL] The process or thread would have blocked, and the `abs_timeout` parameter  
 26267 specified a nanoseconds field value less than zero or greater than or equal to  
 26268 1 000 million.

26269 [EMSGSIZE] The specified message length, `msg_len`, exceeds the message size attribute of  
 26270 the message queue.

26271 TMO [ETIMEDOUT] The `O_NONBLOCK` flag was not set when the message queue was opened,  
 26272 but the timeout expired before the message could be added to the queue.

26273 **EXAMPLES**

26274 None.

26275 **APPLICATION USAGE**

26276 The value of the symbol `{MQ_PRIO_MAX}` limits the number of priority levels supported by the  
 26277 application. Message priorities range from 0 to `{MQ_PRIO_MAX}-1`.

26278 **RATIONALE**

26279 None.

26280 **FUTURE DIRECTIONS**

26281 None.

26282 **SEE ALSO**

26283 `mq_open()`, `mq_receive()`, `mq_setattr()`, `mq_timedreceive()`, `time()`, the Base Definitions volume of  
 26284 IEEE Std. 1003.1-200x, `<mqqueue.h>`, `<time.h>`

26285 **CHANGE HISTORY**

26286 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26287 **Issue 6**

26288 The `mq_send()` function is marked as part of the Message Passing option.

26289 The `[ENOSYS]` error condition has been removed as stubs need not be provided if an  
 26290 implementation does not support the Message Passing option.

26291 The `mq_timedsend()` function is added for alignment with IEEE Std. 1003.1d-1999.

26292 **NAME**26293 mq\_setattr — set message queue attributes (**REALTIME**)26294 **SYNOPSIS**

26295 MSG #include &lt;mqqueue.h&gt;

```
26296 int mq_setattr(mqd_t mqdes, const struct mq_attr *restrict mqstat,
26297 struct mq_attr *restrict omqstat);
26298
```

26299 **DESCRIPTION**26300 The *mq\_setattr()* function is used to set attributes associated with the open message queue description referenced by the message queue descriptor specified by *mqdes*.26302 The message queue attributes corresponding to the following members defined in the **mq\_attr** structure are set to the specified values upon successful completion of *mq\_setattr()*:26304 *mq\_flags* The value of this member is the bitwise-logical OR of zero or more of  
26305 O\_NONBLOCK and any implementation-defined flags.26306 The values of the *mq\_maxmsg*, *mq\_msgsize*, and *mq\_curmsgs* members of the **mq\_attr** structure are ignored by *mq\_setattr()*.26308 If *omqstat* is non-NULL, the *mq\_setattr()* function stores, in the location referenced by *omqstat*, the previous message queue attributes and the current queue status. These values are the same as would be returned by a call to *mq\_getattr()* at that point.26311 **RETURN VALUE**

26312 Upon successful completion, the function shall return a value of zero and the attributes of the message queue shall have been changed as specified.

26314 Otherwise, the message queue attributes shall be unchanged, and the function shall return a value of -1 and set *errno* to indicate the error.26316 **ERRORS**26317 The *mq\_setattr()* function shall fail if:26318 [EBADF] The *mqdes* argument is not a valid message queue descriptor.26319 **EXAMPLES**

26320 None.

26321 **APPLICATION USAGE**

26322 None.

26323 **RATIONALE**

26324 None.

26325 **FUTURE DIRECTIONS**

26326 None.

26327 **SEE ALSO**26328 *mq\_open()*, *mq\_send()*, *mq\_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base  
26329 Definitions volume of IEEE Std. 1003.1-200x, <**mqqueue.h**>26330 **CHANGE HISTORY**

26331 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26332 **Issue 6**

26333 The *mq\_setattr()* function is marked as part of the Message Passing option.

26334 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
26335 implementation does not support the Message Passing option.

26336 The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with  
26337 IEEE Std. 1003.1d-1999.

26338 The **restrict** keyword is added to the *mq\_setattr()* prototype for alignment with the  
26339 ISO/IEC 9899:1999 standard.

26340 **NAME**26341 mq\_timedreceive — receive a message from a message queue (**REALTIME**)26342 **SYNOPSIS**

26343 MSG TMO #include &lt;mqueue.h&gt;

26344 #include &lt;time.h&gt;

26345 int mq\_timedreceive(mqd\_t mqdes, char \*restrict msg\_ptr,

26346 size\_t msg\_len, unsigned \*restrict msg\_prio,

26347 const struct timespec \*restrict abs\_timeout);

26348

26349 **DESCRIPTION**26350 Refer to *mq\_receive()*.

26351 **NAME**

26352 mq\_timedsend — send a message to a message queue (**REALTIME**)

26353 **SYNOPSIS**

26354 MSG TMO #include <mqueue.h>

26355 #include <time.h>

26356 int mq\_timedsend(mqd\_t mqdes, const char \*msg\_ptr, size\_t msg\_len,

26357 unsigned msg\_prio, const struct timespec \*abs\_timeout);

26358

26359 **DESCRIPTION**

26360 Refer to *mq\_send()*.

26361 **NAME**26362 mq\_unlink — remove a message queue (**REALTIME**)26363 **SYNOPSIS**

26364 MSG #include &lt;mqueue.h&gt;

26365 int mq\_unlink(const char \*name);

26366

26367 **DESCRIPTION**

26368 The *mq\_unlink()* function shall remove the message queue named by the path name *name*. After  
 26369 a successful call to *mq\_unlink()* with *name*, a call to *mq\_open()* with *name* fails if the flag  
 26370 O\_CREAT is not set in *flags*. If one or more processes have the message queue open when  
 26371 *mq\_unlink()* is called, destruction of the message queue is postponed until all references to the  
 26372 message queue have been closed.

26373 Calls to *mq\_open()* to recreate the message queue may fail until the message queue is actually  
 26374 removed. However, the *mq\_unlink()* call need not block until all references have been closed; it  
 26375 may return immediately.

26376 **RETURN VALUE**

26377 Upon successful completion, the function shall return a value of zero. Otherwise, the named  
 26378 message queue shall be unchanged by this function call, and the function shall return a value of  
 26379 -1 and set *errno* to indicate the error.

26380 **ERRORS**26381 The *mq\_unlink()* function shall fail if:

26382 [EACCES] Permission is denied to unlink the named message queue.

26383 [ENAMETOOLONG]

26384 The length of the *name* argument exceeds {PATH\_MAX} or a path name  
 26385 component is longer than {NAME\_MAX}.

26386 [ENOENT] The named message queue does not exist.

26387 **EXAMPLES**

26388 None.

26389 **APPLICATION USAGE**

26390 None.

26391 **RATIONALE**

26392 None.

26393 **FUTURE DIRECTIONS**

26394 None.

26395 **SEE ALSO**

26396 *mq\_close()*, *mq\_open()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of  
 26397 IEEE Std. 1003.1-200x, <mqueue.h>

26398 **CHANGE HISTORY**

26399 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26400 **Issue 6**26401 The *mq\_unlink()* function is marked as part of the Message Passing option.

26402 The Open Group corrigenda item U021/5 has been applied, clarifying that upon unsuccessful  
 26403 completion, the named message queue is unchanged by this function.

26404  
26405

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.



26406 **NAME**

26407       mrand48 — generate uniformly distributed pseudo-random signed long integers

26408 **SYNOPSIS**

26409 xSI     #include <stdlib.h>

26410       long mrand48(void);

26411

26412 **DESCRIPTION**

26413       Refer to *drand48()*.

## 26414 NAME

26415 msgctl — XSI message control operations

## 26416 SYNOPSIS

26417 XSI #include &lt;sys/msg.h&gt;

26418 int msgctl(int *msqid*, int *cmd*, struct *msqid\_ds* \**buf*);

26419

## 26420 DESCRIPTION

26421 The *msgctl()* function operates on XSI message queues (see the Base Definitions volume of  
 26422 IEEE Std. 1003.1-200x, Section 3.226, Message Queue). It is unspecified whether this function  
 26423 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 26424 page 543).

26425 The *msgctl()* function shall provide message control operations as specified by *cmd*. The  
 26426 following values for *cmd*, and the message control operations they specify, are:

26427 IPC\_STAT Place the current value of each member of the **msqid\_ds** data structure  
 26428 associated with *msqid* into the structure pointed to by *buf*. The contents of this  
 26429 structure are defined in <sys/msg.h>.

26430 IPC\_SET Set the value of the following members of the **msqid\_ds** data structure  
 26431 associated with *msqid* to the corresponding value found in the structure  
 26432 pointed to by *buf*:

26433 msg\_perm.uid  
 26434 msg\_perm.gid  
 26435 msg\_perm.mode  
 26436 msg\_qbytes

26437 IPC\_SET can only be executed by a process with appropriate privileges or that  
 26438 has an effective user ID equal to the value of **msg\_perm.cuid** or  
 26439 **msg\_perm.uid** in the **msqid\_ds** data structure associated with *msqid*. Only a  
 26440 process with appropriate privileges can raise the value of *msg\_qbytes*.

26441 IPC\_RMID Remove the message queue identifier specified by *msqid* from the system and  
 26442 destroy the message queue and **msqid\_ds** data structure associated with it.  
 26443 IPC\_RMID can only be executed by a process with appropriate privileges or  
 26444 one that has an effective user ID equal to the value of **msg\_perm.cuid** or  
 26445 **msg\_perm.uid** in the **msqid\_ds** data structure associated with *msqid*.

## 26446 RETURN VALUE

26447 Upon successful completion, *msgctl()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 26448 indicate the error.

## 26449 ERRORS

26450 The *msgctl()* function shall fail if:

26451 [EACCES] The argument *cmd* is IPC\_STAT and the calling process does not have read  
 26452 permission; see Section 2.7 (on page 541).

26453 [EINVAL] The value of *msqid* is not a valid message queue identifier; or the value of *cmd*  
 26454 is not a valid command.

26455 [EPERM] The argument *cmd* is IPC\_RMID or IPC\_SET and the effective user ID of the  
 26456 calling process is not equal to that of a process with appropriate privileges  
 26457 and it is not equal to the value of **msg\_perm.cuid** or **msg\_perm.uid** in the data  
 26458 structure associated with *msqid*.

26459 [EPERM] The argument *cmd* is IPC\_SET, an attempt is being made to increase to the  
26460 value of *msg\_qbytes*, and the effective user ID of the calling process does not  
26461 have appropriate privileges.

26462 **EXAMPLES**

26463 None.

26464 **APPLICATION USAGE**

26465 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
26466 (IPC). Application developers who need to use IPC should design their applications so that  
26467 modules using the IPC routines described in Section 2.7 (on page 541) can be easily modified to  
26468 use the alternative interfaces.

26469 **RATIONALE**

26470 None.

26471 **FUTURE DIRECTIONS**

26472 None.

26473 **SEE ALSO**

26474 *mq\_close()*, *mq\_getattr()*, *mq\_notify()*, *mq\_open()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*,  
26475 *mq\_unlink()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
26476 *<sys/msg.h>*, Section 2.7 (on page 541)

26477 **CHANGE HISTORY**

26478 First released in Issue 2. Derived from Issue 2 of the SVID.

26479 **Issue 4**

26480 The function is no longer marked as OPTIONAL FUNCTIONALITY.

26481 Inclusion of the *<sys/types.h>* and *<sys/ipc.h>* headers is removed from the SYNOPSIS section.

26482 A FUTURE DIRECTIONS section is added warning application developers about migration to  
26483 IEEE 1003.4 interfaces for interprocess communication.

26484 The [ENOSYS] error is removed from the ERRORS section.

26485 **Issue 5**

26486 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
26487 DIRECTIONS to a new APPLICATION USAGE section.

## 26488 NAME

26489 msgget — get the XSI message queue identifier

## 26490 SYNOPSIS

26491 XSI #include &lt;sys/msg.h&gt;

26492 int msgget(key\_t key, int msgflg);

26493

## 26494 DESCRIPTION

26495 The *msgget()* function operates on XSI message queues (see the Base Definitions volume of  
 26496 IEEE Std. 1003.1-200x, Section 3.226, Message Queue). It is unspecified whether this function  
 26497 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 26498 page 543).

26499 The *msgget()* function shall return the message queue identifier associated with the argument  
 26500 *key*.

26501 A message queue identifier, associated message queue, and data structure (see <sys/msg.h>), are  
 26502 created for the argument *key* if one of the following is true:

- 26503 • The argument *key* is equal to `IPC_PRIVATE`.
- 26504 • The argument *key* does not already have a message queue identifier associated with it, and  
 26505 (*msgflg* & `IPC_CREAT`) is non-zero.

26506 Upon creation, the data structure associated with the new message queue identifier is initialized  
 26507 as follows:

- 26508 • `msg_perm.cuid`, `msg_perm.uid`, `msg_perm.cgid`, and `msg_perm.gid` are set equal to the  
 26509 effective user ID and effective group ID, respectively, of the calling process.
- 26510 • The low-order 9 bits of `msg_perm.mode` are set equal to the low-order 9 bits of *msgflg*.
- 26511 • `msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime`, and `msg_rtime` are set equal to 0.
- 26512 • `msg_ctime` is set equal to the current time.
- 26513 • `msg_qbytes` is set equal to the system limit.

## 26514 RETURN VALUE

26515 Upon successful completion, *msgget()* shall return a non-negative integer, namely a message  
 26516 queue identifier. Otherwise, it shall return `-1` and set *errno* to indicate the error.

## 26517 ERRORS

26518 The *msgget()* function shall fail if:

- |                         |          |                                                                                                                                                                                                           |
|-------------------------|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 26519<br>26520<br>26521 | [EACCES] | A message queue identifier exists for the argument <i>key</i> , but operation<br>permission as specified by the low-order 9 bits of <i>msgflg</i> would not be granted;<br>see Section 2.7 (on page 541). |
| 26522<br>26523          | [EEXIST] | A message queue identifier exists for the argument <i>key</i> but (( <i>msgflg</i> &<br><code>IPC_CREAT</code> ) && ( <i>msgflg</i> & <code>IPC_EXCL</code> )) is non-zero.                               |
| 26524<br>26525          | [ENOENT] | A message queue identifier does not exist for the argument <i>key</i> and ( <i>msgflg</i> &<br><code>IPC_CREAT</code> ) is 0.                                                                             |
| 26526<br>26527<br>26528 | [ENOSPC] | A message queue identifier is to be created but the system-imposed limit on<br>the maximum number of allowed message queue identifiers system-wide<br>would be exceeded.                                  |

**26529 EXAMPLES**

26530 None.

**26531 APPLICATION USAGE**

26532 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
26533 (IPC). Application developers who need to use IPC should design their applications so that  
26534 modules using the IPC routines described in Section 2.7 (on page 541) can be easily modified to  
26535 use the alternative interfaces.

**26536 RATIONALE**

26537 None.

**26538 FUTURE DIRECTIONS**

26539 None.

**26540 SEE ALSO**

26541 *mq\_close()*, *mq\_getattr()*, *mq\_notify()*, *mq\_open()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*,  
26542 *mq\_unlink()*, *msgctl()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
26543 `<sys/msg.h>`, Section 2.7 (on page 541)

**26544 CHANGE HISTORY**

26545 First released in Issue 2. Derived from Issue 2 of the SVID.

**26546 Issue 4**

26547 The function is no longer marked as OPTIONAL FUNCTIONALITY.

26548 Inclusion of the `<sys/types.h>` and `<sys/ipc.h>` headers is removed from the SYNOPSIS section.

26549 The [ENOSYS] error is removed from the ERRORS section.

**26550 Issue 5**

26551 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
26552 DIRECTIONS to a new APPLICATION USAGE section.

## 26553 NAME

26554 msgrcv — XSI message receive operation

## 26555 SYNOPSIS

26556 XSI 

```
#include <sys/msg.h>
```

26557 

```
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
26558 int msgflg);
```

26559

## 26560 DESCRIPTION

26561 The *msgrcv()* function operates on XSI message queues (see the Base Definitions volume of  
26562 IEEE Std. 1003.1-200x, Section 3.226, Message Queue). It is unspecified whether this function  
26563 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
26564 page 543).

26565 The *msgrcv()* function shall read a message from the queue associated with the message queue  
26566 identifier specified by *msqid* and place it in the user-defined buffer pointed to by *msgp*.

26567 The application shall ensure that the argument *msgp* points to a user-defined buffer that contains  
26568 first a field of type **long** specifying the type of the message, and then a data portion that holds  
26569 the data bytes of the message. The structure below is an example of what this user-defined  
26570 buffer might look like:

```
26571 struct mymsg {
26572 long mtype; /* Message type. */
26573 char mtext[1]; /* Message text. */
26574 }
```

26575 The structure member *mtype* is the received message's type as specified by the sending process.

26576 The structure member *mtext* is the text of the message.

26577 The argument *msgsz* specifies the size in bytes of *mtext*. The received message is truncated to  
26578 *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & MSG\_NOERROR) is non-zero. The truncated  
26579 part of the message is lost and no indication of the truncation is given to the calling process.

26580 If the value of *msgsz* is greater than {SSIZE\_MAX}, the result is implementation-defined.

26581 The argument *msgtyp* specifies the type of message requested as follows:

- 26582 • If *msgtyp* is 0, the first message on the queue is received.
- 26583 • If *msgtyp* is greater than 0, the first message of type *msgtyp* is received.
- 26584 • If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the  
26585 absolute value of *msgtyp* is received.

26586 The argument *msgflg* specifies the action to be taken if a message of the desired type is not on the  
26587 queue. These are as follows:

- 26588 • If (*msgflg* & IPC\_NOWAIT) is non-zero, the calling thread shall return immediately with a  
26589 return value of -1 and *errno* set to [ENOMSG].
- 26590 • If (*msgflg* & IPC\_NOWAIT) is 0, the calling thread shall suspend execution until one of the  
26591 following occurs:
  - 26592 — A message of the desired type is placed on the queue.
  - 26593 — The message queue identifier *msqid* is removed from the system; when this occurs, *errno*  
26594 shall be set equal to [EIDRM] and -1 shall be returned.

26595 — The calling thread receives a signal that is to be caught; in this case a message is not  
 26596 received and the calling thread resumes execution in the manner prescribed in *sigaction()*.

26597 Upon successful completion, the following actions are taken with respect to the data structure  
 26598 associated with *msqid*:

- 26599 • **msg\_qnum** is decremented by 1.
- 26600 • **msg\_lrpid** is set equal to the process ID of the calling process.
- 26601 • **msg\_rtime** is set equal to the current time.

#### 26602 RETURN VALUE

26603 Upon successful completion, *msgrcv()* shall return a value equal to the number of bytes actually  
 26604 placed into the buffer *mtext*. Otherwise, no message shall be received, *msgrcv()* shall return  
 26605 (**ssize\_t**)−1, and *errno* shall be set to indicate the error.

#### 26606 ERRORS

26607 The *msgrcv()* function shall fail if:

- |       |          |                                                                                                 |  |
|-------|----------|-------------------------------------------------------------------------------------------------|--|
| 26608 | [E2BIG]  | The value of <i>mtext</i> is greater than <i>msgsz</i> and ( <i>msgflg</i> & MSG_NOERROR) is 0. |  |
| 26609 | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page                 |  |
| 26610 |          | 541).                                                                                           |  |
| 26611 | [EIDRM]  | The message queue identifier <i>msqid</i> is removed from the system.                           |  |
| 26612 | [EINTR]  | The <i>msgrcv()</i> function was interrupted by a signal.                                       |  |
| 26613 | [EINVAL] | <i>msqid</i> is not a valid message queue identifier.                                           |  |
| 26614 | [ENOMSG] | The queue does not contain a message of the desired type and ( <i>msgflg</i> &                  |  |
| 26615 |          | IPC_NOWAIT) is non-zero.                                                                        |  |

#### 26616 EXAMPLES

##### 26617 Receiving a Message

26618 The following example receives the first message on the queue (based on the value of the  
 26619 *msgtype* argument, 0). The queue is identified by the *msqid* argument (assuming that the value  
 26620 has previously been set). This call specifies that an error should be reported if no message is  
 26621 available, but not if the message is too large. The message size is calculated directly using the  
 26622 *sizeof* operator.

```

26623 #include <sys/msg.h>
26624 ...
26625 int result;
26626 int msqid;
26627 struct message {
26628 long type;
26629 char text[20];
26630 } msg;
26631 long msgtyp = 0;
26632 ...
26633 result = msgrcv(msqid, (void *) &msg, sizeof(msg.text),
26634 msgtyp, MSG_NOERROR | IPC_NOWAIT);

```

**26635 APPLICATION USAGE**

26636 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
26637 (IPC). Application developers who need to use IPC should design their applications so that  
26638 modules using the IPC routines described in Section 2.7 (on page 541) can be easily modified to  
26639 use the alternative interfaces.

**26640 RATIONALE**

26641 None.

**26642 FUTURE DIRECTIONS**

26643 None.

**26644 SEE ALSO**

26645 *mq\_close()*, *mq\_getattr()*, *mq\_notify()*, *mq\_open()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*,  
26646 *mq\_unlink()*, *msgctl()*, *msgget()*, *msgsnd()*, *sigaction()*, the Base Definitions volume of  
26647 IEEE Std. 1003.1-200x, <sys/msg.h>, Section 2.7 (on page 541)

**26648 CHANGE HISTORY**

26649 First released in Issue 2. Derived from Issue 2 of the SVID.

**26650 Issue 4**

26651 The function is no longer marked as OPTIONAL FUNCTIONALITY.

26652 Inclusion of the <sys/types.h> and <sys/ipc.h> headers is removed from the SYNOPSIS section.

26653 The [ENOSYS] error is removed from the ERRORS section.

26654 A FUTURE DIRECTIONS section is added warning application developers about migration to  
26655 IEEE 1003.4 interfaces for interprocess communication.

**26656 Issue 5**

26657 The type of the return value is changed from **int** to **ssize\_t**, and a warning is added to the  
26658 DESCRIPTION about values of *msgsz* larger the {SSIZE\_MAX}.

26659 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
26660 DIRECTIONS to the APPLICATION USAGE section.

**26661 Issue 6**

26662 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.



## 26663 NAME

26664 msgsnd — XSI message send operation

## 26665 SYNOPSIS

26666 XSI 

```
#include <sys/msg.h>
```

26667 

```
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

26668

## 26669 DESCRIPTION

26670 The *msgsnd()* function operates on XSI message queues (see the Base Definitions volume of  
 26671 IEEE Std. 1003.1-200x, Section 3.226, Message Queue). It is unspecified whether this function  
 26672 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 26673 page 543).

26674 The *msgsnd()* function is used to send a message to the queue associated with the message  
 26675 queue identifier specified by *msqid*.

26676 The application shall ensure that the argument *msgp* points to a user-defined buffer that contains  
 26677 first a field of type **long** specifying the type of the message, and then a data portion that holds  
 26678 the data bytes of the message. The structure below is an example of what this user-defined  
 26679 buffer might look like:

```
26680 struct mymsg {
26681 long mtype; /* Message type. */
26682 char mtext[1]; /* Message text. */
26683 }
```

26684 The structure member *mtype* is a non-zero positive type **long** that can be used by the receiving  
 26685 process for message selection.

26686 The structure member *mtext* is any text of length *msgsz* bytes. The argument *msgsz* can range  
 26687 from 0 to a system-imposed maximum.

26688 The argument *msgflg* specifies the action to be taken if one or more of the following are true:

- 26689 • The number of bytes already on the queue is equal to *msg\_qbytes*; see `<sys/msg.h>`.
- 26690 • The total number of messages on all queues system-wide is equal to the system-imposed  
 26691 limit.

26692 These actions are as follows:

- 26693 • If (*msgflg* & IPC\_NOWAIT) is non-zero, the message shall not be sent and the calling thread  
 26694 shall return immediately.
- 26695 • If (*msgflg* & IPC\_NOWAIT) is 0, the calling thread shall suspend execution until one of the  
 26696 following occurs:
  - 26697 — The condition responsible for the suspension no longer exists, in which case the message  
 26698 is sent.
  - 26699 — The message queue identifier *msqid* is removed from the system; when this occurs, *errno*  
 26700 shall be set equal to [EIDRM] and `-1` shall be returned.
  - 26701 — The calling thread receives a signal that is to be caught; in this case the message is not  
 26702 sent and the calling thread resumes execution in the manner prescribed in *sigaction()*.

26703 Upon successful completion, the following actions are taken with respect to the data structure  
 26704 associated with *msqid*; see `<sys/msg.h>`:

- 26705           • **msg\_qnum** is incremented by 1.
- 26706           • **msg\_lspid** is set equal to the process ID of the calling process.
- 26707           • **msg\_stime** is set equal to the current time.

**26708 RETURN VALUE**

26709           Upon successful completion, *msgsnd()* shall return 0; otherwise, no message shall be sent,  
 26710           *msgsnd()* shall return -1, and *errno* shall be set to indicate the error.

**26711 ERRORS**

26712           The *msgsnd()* function shall fail if:

- |                         |          |                                                                                                                                                                                                       |
|-------------------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 26713<br>26714          | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page 541).                                                                                                                 |
| 26715<br>26716          | [EAGAIN] | The message cannot be sent for one of the reasons cited above and ( <i>msgflg</i> & IPC_NOWAIT) is non-zero.                                                                                          |
| 26717                   | [EIDRM]  | The message queue identifier <i>msqid</i> is removed from the system.                                                                                                                                 |
| 26718                   | [EINTR]  | The <i>msgsnd()</i> function was interrupted by a signal.                                                                                                                                             |
| 26719<br>26720<br>26721 | [EINVAL] | The value of <i>msqid</i> is not a valid message queue identifier, or the value of <i>mtype</i> is less than 1; or the value of <i>msgsz</i> is less than 0 or greater than the system-imposed limit. |

**26722 EXAMPLES****26723 Sending a Message**

26724           The following example sends a message to the queue identified by the *msqid* argument  
 26725           (assuming that value has previously been set). This call specifies that an error should be  
 26726           reported if no message is available. The message size is calculated directly using the *sizeof*  
 26727           operator.

```

26728 #include <sys/msg.h>
26729 ...
26730 int result;
26731 int msqid;
26732 struct message {
26733 long type;
26734 char text[20];
26735 } msg;

26736 msg.type = 1;
26737 strcpy(msg.text, "This is message 1");
26738 ...
26739 result = msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);

```

**26740 APPLICATION USAGE**

26741           The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
 26742           (IPC). Application developers who need to use IPC should design their applications so that  
 26743           modules using the IPC routines described in Section 2.7 (on page 541) can be easily modified to  
 26744           use the alternative interfaces.

26745 **RATIONALE**

26746 None.

26747 **FUTURE DIRECTIONS**

26748 None.

26749 **SEE ALSO**

26750 *mq\_close()*, *mq\_getattr()*, *mq\_notify()*, *mq\_open()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*,  
26751 *mq\_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*, *sigaction()*, the Base Definitions volume of  
26752 IEEE Std. 1003.1-200x, <sys/msg.h>, Section 2.7 (on page 541)

26753 **CHANGE HISTORY**

26754 First released in Issue 2. Derived from Issue 2 of the SVID.

26755 **Issue 4**

26756 The function is no longer marked as OPTIONAL FUNCTIONALITY.

26757 Inclusion of the &lt;sys/types.h&gt; and &lt;sys/ipc.h&gt; headers is removed from the SYNOPSIS section.

26758 Also the type of argument *msgp* is changed from **void\*** to **const void\***.

26759 In the DESCRIPTION, the example of a message buffer is changed:

- 26760 • Explicitly to define the first member as being of type **long**
- 26761 • To define the size of the message array *mtext*

26762 The [ENOSYS] error is removed from the ERRORS section.

26763 A FUTURE DIRECTIONS section is added warning application developers about migration to

26764 IEEE 1003.4 interfaces for interprocess communication.

26765 **Issue 5**

26766 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
26767 DIRECTIONS to a new APPLICATION USAGE section.

26768 **Issue 6**

26769 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

26770 **NAME**26771 `msync` — synchronize memory with physical storage26772 **SYNOPSIS**26773 MF SIO `#include <sys/mman.h>`26774 `int msync(void *addr, size_t len, int flags);`

26775

26776 **DESCRIPTION**

26777 The `msync()` function shall write all modified data to permanent storage locations, if any, in  
 26778 those whole pages containing any part of the address space of the process starting at address  
 26779 `addr` and continuing for `len` bytes. If no such storage exists, `msync()` need not have any effect. If  
 26780 requested, the `msync()` function then invalidates cached copies of data.

26781 The implementation shall require that `addr` be a multiple of the page size as returned by  
 26782 `sysconf()`.

26783 For mappings to files, the `msync()` function ensures that all write operations are completed as  
 26784 defined for synchronized I/O data integrity completion. It is unspecified whether the  
 26785 implementation also writes out other file attributes. When the `msync()` function is called on  
 26786 MAP\_PRIVATE mappings, any modified data shall not be written to the underlying object and  
 26787 shall not cause such data to be made visible to other processes. It is unspecified whether data in  
 26788 SHM|TYM MAP\_PRIVATE mappings has any permanent storage locations. The effect of `msync()` on a  
 26789 shared memory object or a typed memory object is unspecified. The behavior of this function is  
 26790 unspecified if the mapping was not established by a call to `mmap()`.

26791 The `flags` argument is constructed from the bitwise-inclusive OR of one or more of the following  
 26792 flags defined in the header `<sys/mman.h>`:

26793

26794

26795

26796

26797

| Symbolic Constant | Description                  |
|-------------------|------------------------------|
| MS_ASYNC          | Perform asynchronous writes. |
| MS_SYNC           | Perform synchronous writes.  |
| MS_INVALIDATE     | Invalidate cached data.      |

26798 When MS\_ASYNC is specified, `msync()` shall return immediately once all the write operations  
 26799 are initiated or queued for servicing; when MS\_SYNC is specified, `msync()` shall not return until  
 26800 all write operations are completed as defined for synchronized I/O data integrity completion.  
 26801 Either MS\_ASYNC or MS\_SYNC is specified, but not both.

26802 When MS\_INVALIDATE is specified, `msync()` invalidates all cached copies of mapped data that  
 26803 are inconsistent with the permanent storage locations such that subsequent references shall  
 26804 obtain data that was consistent with the permanent storage locations sometime between the call  
 26805 to `msync()` and the first subsequent memory reference to the data.

26806 If `msync()` causes any write to a file, the file's `st_ctime` and `st_mtime` fields shall be marked for  
 26807 update.

26808 **RETURN VALUE**

26809 Upon successful completion, `msync()` shall return 0; otherwise, it shall return `-1` and set `errno` to  
 26810 indicate the error.

26811 **ERRORS**

26812 The `msync()` function shall fail if:

26813 [EBUSY] Some or all of the addresses in the range starting at `addr` and continuing for `len`  
 26814 bytes are locked, and MS\_INVALIDATE is specified.

- 26815 [EINVAL] The value of *flags* is invalid.
- 26816 [EINVAL] The value of *addr* is not a multiple of the page size, {PAGESIZE}.
- 26817 [ENOMEM] The addresses in the range starting at *addr* and continuing for *len* bytes are  
26818 outside the range allowed for the address space of a process or specify one or  
26819 more pages that are not mapped.

**26820 EXAMPLES**

26821 None.

**26822 APPLICATION USAGE**

26823 The *msync()* function is only supported if the Memory Mapped Files option and the  
26824 Synchronized Input and Output option are supported, and thus need not be available on all  
26825 implementations.

26826 The *msync()* function should be used by programs that require a memory object to be in a  
26827 known state; for example, in building transaction facilities.

26828 Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees  
26829 that *msync()* is the only control over when pages are or are not written to disk.

**26830 RATIONALE**

26831 The *msync()* function is used to write out data in a mapped region to the permanent storage for  
26832 the underlying object. The call to *msync()* ensures data integrity of the file.

26833 After the data is written out, any cached data may be invalidated if the MS\_INVALIDATE flag  
26834 was specified. This is useful on systems that do not support read/write consistency.

**26835 FUTURE DIRECTIONS**

26836 None.

**26837 SEE ALSO**

26838 *mmap()*, *sysconf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/mman.h>

**26839 CHANGE HISTORY**

26840 First released in Issue 4, Version 2.

**26841 Issue 5**

26842 Moved from X/OPEN UNIX extension to BASE.

26843 Aligned with *msync()* in the POSIX Realtime Extension as follows:

- 26844 • The DESCRIPTION is extensively reworded.
- 26845 • [EBUSY] and a new form of [EINVAL] are added as mandatory error conditions.

**26846 Issue 6**

26847 The *msync()* function is marked as part of the Memory Mapped Files and Synchronized Input  
26848 and Output options.

26849 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 26850 • The [EBUSY] mandatory error condition is added.

26851 The following new requirements on POSIX implementations derive from alignment with the  
26852 Single UNIX Specification:

- 26853 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of  
26854 the page size.
- 26855 • The second [EINVAL] error condition is made mandatory.

26856  
26857

The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by adding reference to typed memory objects.

26858 **NAME**

26859           munlock — unlock a range of process address space

26860 **SYNOPSIS**

26861 MLR       #include &lt;sys/mman.h&gt;

26862           int munlock(const void \* *addr*, size\_t *len*);

26863

26864 **DESCRIPTION**26865           Refer to *mlock()*.

26866 **NAME**

26867           munlockall — unlock the address space of a process

26868 **SYNOPSIS**

26869 ML       #include <sys/mman.h>

26870       int munlockall(void);

26871

26872 **DESCRIPTION**

26873       Refer to *mlockall()*.



26874 **NAME**26875 `munmap` — unmap pages of memory26876 **SYNOPSIS**26877 MF|SHM `#include <sys/mman.h>`26878 `int munmap(void *addr, size_t len);`

26879

26880 **DESCRIPTION**

26881 The `munmap()` function shall remove any mappings for those entire pages containing any part of  
 26882 the address space of the process starting at `addr` and continuing for `len` bytes. Further references  
 26883 to these pages result in the generation of a SIGSEGV signal to the process. If there are no  
 26884 mappings in the specified address range, then `munmap()` has no effect.

26885 The implementation shall require that `addr` be a multiple of the page size {PAGESIZE}.

26886 If a mapping to be removed was private, any modifications made in this address range shall be  
 26887 discarded.

26888 ML|MLR Any memory locks (see `mlock()` and `mlockall()`) associated with this address range shall be  
 26889 removed, as if by an appropriate call to `munlock()`.

26890 TYM If a mapping removed from a typed memory object causes the corresponding address range of  
 26891 the memory pool to be inaccessible by any process in the system except through allocatable  
 26892 mappings (that is, mappings of typed memory objects opened with the  
 26893 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag), then that range of the memory pool shall  
 26894 become deallocated and may become available to satisfy future typed memory allocation  
 26895 requests.

26896 A mapping removed from a typed memory object opened with the  
 26897 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag shall not affect in any way the availability of  
 26898 that typed memory for allocation.

26899 The behavior of this function is unspecified if the mapping was not established by a call to  
 26900 `mmap()`.

26901 **RETURN VALUE**

26902 Upon successful completion, `munmap()` shall return 0; otherwise, it shall return -1 and set `errno`  
 26903 to indicate the error.

26904 **ERRORS**

26905 The `munmap()` function shall fail if:

26906 [EINVAL] Addresses in the range [`addr`,`addr+len`) are outside the valid range for the  
 26907 address space of a process.

26908 [EINVAL] The `len` argument is 0.

26909 [EINVAL] The `addr` argument is not a multiple of the page size as returned by `sysconf()`.

26910 **EXAMPLES**

26911 None.

26912 **APPLICATION USAGE**

26913 The *munmap()* function is only supported if the Memory Mapped Files option or the Shared  
26914 Memory Objects option is supported.

26915 **RATIONALE**26916 The *munmap()* function corresponds to SVR4, just as the *mmap()* function does.

26917 It is possible that an application has applied process memory locking to a region that contains  
26918 shared memory. If this has occurred, the *munmap()* call ignores those locks and, if necessary,  
26919 causes those locks to be removed.

26920 **FUTURE DIRECTIONS**

26921 None.

26922 **SEE ALSO**

26923 *mlock()*, *mlockall()*, *mmap()*, *posix\_typed\_mem\_open()*, *sysconf()*, the Base Definitions volume of  
26924 IEEE Std. 1003.1-200x, <signal.h>, <sys/mman.h>

26925 **CHANGE HISTORY**

26926 First released in Issue 4, Version 2.

26927 **Issue 5**

26928 Moved from X/OPEN UNIX extension to BASE.

26929 Aligned with *munmap()* in the POSIX Realtime Extension as follows:

- 26930 • The DESCRIPTION is extensively reworded.
- 26931 • The SIGBUS error is no longer permitted to be generated.

26932 **Issue 6**

26933 The *munmap()* function is marked as part of the Memory Mapped Files and Shared Memory  
26934 Objects option.

26935 The following new requirements on POSIX implementations derive from alignment with the  
26936 Single UNIX Specification:

- 26937 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of  
26938 the page size.
- 26939 • The [EINVAL] error conditions are added.

26940 The following changes are made for alignment with IEEE Std. 1003.1j-2000:

- 26941 • Semantics for typed memory objects are added to the DESCRIPTION.
- 26942 • The *posix\_typed\_mem\_open()* function is added to the SEE ALSO section.

26943 **NAME**

26944 nan, nanf, nanl — return quiet NaN

26945 **SYNOPSIS**

26946 #include &lt;math.h&gt;

26947 double nan(const char \*tagp);

26948 float nanf(const char \*tagp);

26949 long double nanl(const char \*tagp);

26950 **DESCRIPTION**

26951 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 26952 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26953 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

26954 The function call *nan("n-char-sequence")* shall be equivalent to:26955 strtod("NAN(*n-char-sequence*)", (char \*\*) NULL);26956 The function call *nan("")* shall be equivalent to:

26957 strtod("NAN()", (char \*\*) NULL)

26958 If *tagp* does not point to an *n-char* sequence or an empty string, the function call shall be  
 26959 equivalent to:

26960 strtod("NAN", (char \*\*) NULL)

26961 Function calls to *nanf()* and *nanl()* are equivalent to the corresponding function calls to *strtof()*  
 26962 and *strtold()*.

26963 **RETURN VALUE**26964 These functions shall return a quiet NaN, if available, with content indicated through *tagp*.

26965 If the implementation does not support quiet NaNs, these functions shall return zero.

26966 **ERRORS**

26967 No errors are defined.

26968 **EXAMPLES**

26969 None.

26970 **APPLICATION USAGE**

26971 None.

26972 **RATIONALE**

26973 None.

26974 **FUTURE DIRECTIONS**

26975 None.

26976 **SEE ALSO**

26977 *strtod()*, <REFERENCE UNDEFINED>(strtold), the Base Definitions volume of  
 26978 IEEE Std. 1003.1-200x, <math.h>

26979 **CHANGE HISTORY**

26980 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

26981 **NAME**26982 nanosleep — high resolution sleep (**REALTIME**)26983 **SYNOPSIS**26984 TMR 

```
#include <time.h>
```

26985 

```
int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
```

26986

26987 **DESCRIPTION**

26988 The *nanosleep()* function shall cause the current thread to be suspended from execution until  
 26989 either the time interval specified by the *rqtp* argument has elapsed or a signal is delivered to the  
 26990 calling thread, and its action is to invoke a signal-catching function or to terminate the process.  
 26991 The suspension time may be longer than requested because the argument value is rounded up to  
 26992 an integer multiple of the sleep resolution or because of the scheduling of other activity by the  
 26993 system. But, except for the case of being interrupted by a signal, the suspension time shall not be  
 26994 less than the time specified by *rqtp*, as measured by the system clock, **CLOCK\_REALTIME**.

26995 The use of the *nanosleep()* function has no effect on the action or blockage of any signal.26996 **RETURN VALUE**26997 If the *nanosleep()* function returns because the requested time has elapsed, its return value shall  
26998 be zero.

26999 If the *nanosleep()* function returns because it has been interrupted by a signal, the function shall  
 27000 return a value of  $-1$  and set *errno* to indicate the interruption. If the *rmtp* argument is non-NULL,  
 27001 the **timespec** structure referenced by it is updated to contain the amount of time remaining in  
 27002 the interval (the requested time minus the time actually slept). If the *rmtp* argument is NULL, the  
 27003 remaining time is not returned.

27004 If *nanosleep()* fails, it shall return a value of  $-1$  and set *errno* to indicate the error.27005 **ERRORS**27006 The *nanosleep()* function shall fail if:27007 [EINTR] The *nanosleep()* function was interrupted by a signal.27008 [EINVAL] The *rqtp* argument specified a nanosecond value less than zero or greater than  
27009 or equal to 1000 million.27010 **EXAMPLES**

27011 None.

27012 **APPLICATION USAGE**

27013 None.

27014 **RATIONALE**

27015 It is common to suspend execution of a process for an interval in order to poll the status of a  
 27016 non-interrupting function. A large number of actual needs can be met with a simple extension to  
 27017 *sleep()* that provides finer resolution.

27018 In the POSIX.1-1990 standard and SVR4, it is possible to implement such a routine, but the  
 27019 frequency of wakeup is limited by the resolution of the *alarm()* and *sleep()* functions. In 4.3 BSD,  
 27020 it is possible to write such a routine using no static storage and reserving no system facilities.  
 27021 Although it is possible to write a function with similar functionality to *sleep()* using the  
 27022 remainder of the timers function, such a function requires the use of signals and the reservation  
 27023 of some signal number. This volume of IEEE Std. 1003.1-200x requires that *nanosleep()* be non-  
 27024 intrusive of the signals function.

27025 The *nanosleep()* function shall return a value of 0 on success and –1 on failure or if interrupted.  
27026 This latter case is different from *sleep()*. This was done because the remaining time is returned  
27027 via an argument structure pointer, *rmtp*, instead of as the return value.

27028 **FUTURE DIRECTIONS**

27029 None.

27030 **SEE ALSO**

27031 *sleep()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**time.h**>

27032 **CHANGE HISTORY**

27033 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

27034 **Issue 6**

27035 The *nanosleep()* function is marked as part of the Timers option.

27036 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
27037 implementation does not support the Timers option.

27038 **NAME**

27039 nearbyint, nearbyintf, nearbyintl — floating-point rounding functions

27040 **SYNOPSIS**

27041 #include <math.h>

27042 double nearbyint(double x);

27043 float nearbyintf(float x);

27044 long double nearbyintl(long double x);

27045 **DESCRIPTION**

27046 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
27047 conflict between the requirements described here and the ISO C standard is unintentional. This  
27048 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

27049 These functions shall round their argument to an integer value in floating-point format, using  
27050 the current rounding direction and without raising the inexact floating-point exception.

27051 An application wishing to check for error situations should set *errno* to 0 before calling these  
27052 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

27053 **RETURN VALUE**

27054 Upon successful completion, these functions shall return the rounded integer value.

27055 If *x* is  $\pm\text{Inf}$ , these functions shall return *x*.

27056 If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

27057 **ERRORS**

27058 These functions may fail if:

27059 [EDOM] The value of *x* is NaN.

27060 **EXAMPLES**

27061 None.

27062 **APPLICATION USAGE**

27063 None.

27064 **RATIONALE**

27065 None.

27066 **FUTURE DIRECTIONS**

27067 None.

27068 **SEE ALSO**

27069 The Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

27070 **CHANGE HISTORY**

27071 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27072 **NAME**

27073 nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl — next representable  
 27074 double-precision floating-point number

27075 **SYNOPSIS**

```
27076 xSI #include <math.h>
27077 double nextafter(double x, double y);
27078 float nextafterf(float x, float y);
27079 long double nextafterl(long double x, long double y);
27080 double nexttoward(double x, long double y);
27081 float nexttowardf(float x, long double y);
27082 long double nexttowardl(long double x, long double y);
27083
```

27084 **DESCRIPTION**

27085 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 27086 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27087 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

27088 The *nextafter()*, *nextafterf()*, and *nextafterl()* functions shall compute the next representable  
 27089 double-precision floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*,  
 27090 *nextafter()* returns the largest representable floating-point number less than *x*. The *nextafter()*,  
 27091 *nextafterf()*, and *nextafterl()* functions shall return *y* if *x* equals *y*.

27092 The *nexttoward()*, *nexttowardf()*, and *nexttowardl()* functions are equivalent to the corresponding  
 27093 *nextafter()* functions, except that the second parameter has type **long double** and the functions  
 27094 return *y* converted to the type of the function if *x* equals *y*.

27095 An application wishing to check for error situations should set *errno* to 0 before calling  
 27096 *nextafter()*. If *errno* is non-zero on return, or the value NaN is returned, an error has occurred.

27097 **RETURN VALUE**

27098 These functions shall return the next representable double-precision floating-point value  
 27099 following *x* in the direction of *y*. The *nextafter()*, *nextafterf()*, and *nextafterl()* functions shall  
 27100 return *y* if *x* equals *y*.

27101 If *x* or *y* is NaN, then *nextafter()* shall return NaN and may set *errno* to [EDOM].

27102 If *x* is finite and the correct function value would overflow, HUGE\_VAL shall be returned and  
 27103 *errno* set to [ERANGE].

27104 **ERRORS**

27105 These functions shall fail if:

27106 [ERANGE] The correct value would overflow.

27107 These functions may fail if:

27108 [EDOM] The *x* or *y* argument is NaN.

27109 **EXAMPLES**

27110 None.

27111 **APPLICATION USAGE**

27112 None.

27113 **RATIONALE**

27114 None.

27115 **FUTURE DIRECTIONS**

27116 None.

27117 **SEE ALSO**27118 The Base Definitions volume of IEEE Std. 1003.1-200x, `<math.h>`27119 **CHANGE HISTORY**

27120 First released in Issue 4, Version 2.

27121 **Issue 5**

27122 Moved from X/OPEN UNIX extension to BASE.

27123 **Issue 6**27124 The `nextafterf()`, `nextafterl()`, `nexttoward()`, `nexttowardf()`, `nexttowardl()` functions are added for  
27125 alignment with the ISO/IEC 9899:1999 standard.



27126 **NAME**

27127 nftw — walk a file tree

27128 **SYNOPSIS**27129 XSI 

```
#include <ftw.h>
```

```
27130 int nftw(const char *path, int (*fn)(const char *,
27131 const struct stat *, int, struct FTW *), int depth, int flags);
27132
```

27133 **DESCRIPTION**

27134 The *nftw()* function shall recursively descend the directory hierarchy rooted in *path*. The *nftw()*  
 27135 function has a similar effect to *ftw()* except that it takes an additional argument *flags*, which is a  
 27136 bitwise-inclusive OR of zero or more of the following flags:

27137 **FTW\_CHDIR** If set, *nftw()* shall change the current working directory to each directory as it  
 27138 reports files in that directory. If clear, *nftw()* shall not change the current  
 27139 working directory.

27140 **FTW\_DEPTH** If set, *nftw()* shall report all files in a directory before reporting the directory  
 27141 itself. If clear, *nftw()* shall report any directory before reporting the files in that  
 27142 directory.

27143 **FTW\_MOUNT** If set, *nftw()* shall only report files in the same file system as *path*. If clear,  
 27144 *nftw()* shall report all files encountered during the walk.

27145 **FTW\_PHYS** If set, *nftw()* shall perform a physical walk and shall not follow symbolic links.

27146 If **FTW\_PHYS** is clear and **FTW\_DEPTH** is set, *nftw()* shall follow links instead of reporting  
 27147 them, but shall not report any directory that would be a descendant of itself. If **FTW\_PHYS** is  
 27148 clear and **FTW\_DEPTH** is clear, *nftw()* shall follow links instead of reporting them, but shall not  
 27149 report the contents of any directory that would be a descendant of itself.

27150 At each file it encounters, *nftw()* shall call the user-supplied function *fn* with four arguments:

- 27151 • The first argument is the path name of the object.
- 27152 • The second argument is a pointer to the **stat** buffer containing information on the object.
- 27153 • The third argument is an integer giving additional information. Its value is one of the  
 27154 following:

27155 **FTW\_F** The object is a file.

27156 **FTW\_D** The object is a directory.

27157 **FTW\_DP** The object is a directory and subdirectories have been visited. (This condition  
 27158 shall only occur if the **FTW\_DEPTH** flag is included in *flags*.)

27159 **FTW\_SL** The object is a symbolic link. (This condition shall only occur if the **FTW\_PHYS**  
 27160 flag is included in *flags*.)

27161 **FTW\_SLN** The object is a symbolic link that does not name an existing file. (This  
 27162 condition shall only occur if the **FTW\_PHYS** flag is not included in *flags*.)

27163 **FTW\_DNR** The object is a directory that cannot be read. The *fn* function shall not be called  
 27164 for any of its descendants.

27165 **FTW\_NS** The *stat()* function failed on the object because of lack of appropriate  
 27166 permission. The **stat** buffer passed to *fn* is undefined. Failure of *stat()* for any  
 27167 other reason is considered an error and *nftw()* shall return  $-1$ .

27168       • The fourth argument is a pointer to an **FTW** structure. The value of **base** is the offset of the  
 27169       object's file name in the path name passed as the first argument to *fn*. The value of **level**  
 27170       indicates depth relative to the root of the walk, where the root level is 0.

27171       The argument *depth* sets the maximum number of file descriptors that are shall be by *nftw()*  
 27172       while traversing the file tree. At most one file descriptor shall be used for each directory level.

#### 27173 RETURN VALUE

27174       The *nftw()* function shall continue until the first of the following conditions occurs:

- 27175       • An invocation of *fn* shall return a non-zero value, in which case *nftw()* shall return that value.
- 27176       • The *nftw()* function detects an error other than [EACCES] (see FTW\_DNR and FTW\_NS  
 27177       above), in which case *nftw()* shall return  $-1$  and set *errno* to indicate the error.
- 27178       • The tree is exhausted, in which case *nftw()* shall return 0.

#### 27179 ERRORS

27180       The *nftw()* function shall fail if:

27181       [EACCES]       Search permission is denied for any component of *path* or read permission is  
 27182       denied for *path*, or *fn* returns  $-1$  and does not reset *errno*.

27183       [ELOOP]        A loop exists in symbolic links encountered during resolution of the *path*  
 27184       argument.

27185       [ENAMETOOLONG]

27186       The length of the *path* argument exceeds {PATH\_MAX} or a path name  
 27187       component is longer than {NAME\_MAX}.

27188       [ENOENT]        A component of *path* does not name an existing file or *path* is an empty string.

27189       [ENOTDIR]       A component of *path* is not a directory.

27190       The *nftw()* function may fail if:

27191       [ELOOP]        More than {SYMLOOP\_MAX} symbolic links were encountered during  
 27192       resolution of the *path* argument.

27193       [EMFILE]        {OPEN\_MAX} file descriptors are currently open in the calling process.

27194       [ENAMETOOLONG]

27195       Path name resolution of a symbolic link produced an intermediate result  
 27196       whose length exceeds {PATH\_MAX}.

27197       [ENFILE]        Too many files are currently open in the system.

27198       In addition, *errno* may be set if the function pointed by *fn* causes *errno* to be set.

#### 27199 EXAMPLES

27200       The following example walks the **/tmp** directory and its subdirectories, calling the *nftw()*  
 27201       function for every directory entry, to a maximum of 5 levels deep.

```
27202 #include <ftw.h>
```

```
27203 ...
```

```
27204 int nftwfunc(const char *, const struct stat *, int, struct FTW *);
```

```
27205 int nftwfunc(const char *filename, const struct stat *statptr,
27206 int fileflags, struct FTW *pftw)
```

```
27207 {
```

```
27208 return 0;
```

```
27209 }
```

```
27210 ...
27211 char *startpath = "/tmp";
27212 int depth = 5;
27213 int flags = FTW_CHDIR | FTW_DEPTH | FTW_MOUNT;
27214 int ret;

27215 ret = nftw(startpath, nftwfunc, depth, flags);
```

**27216 APPLICATION USAGE**

27217 None.

**27218 RATIONALE**

27219 None.

**27220 FUTURE DIRECTIONS**

27221 None.

**27222 SEE ALSO**

27223 *lstat()*, *opendir()*, *readdir()*, *stat()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ftw.h> |

**27224 CHANGE HISTORY**

27225 First released in Issue 4, Version 2.

**27226 Issue 5**

27227 Moved from X/OPEN UNIX extension to BASE.

27228 In the DESCRIPTION, the definition of the *depth* argument is clarified.

**27229 Issue 6**

27230 The Open Group Base Resolution bwg97-003 is applied. |

27231 The wording of the mandatory [ELOOP] error condition is updated, and a second optional |

27232 [ELOOP] error condition is added. |

27233 **NAME**

27234 nice — change the nice value of a process

27235 **SYNOPSIS**27236 XSI `#include <unistd.h>`27237 `int nice(int incr);`

27238

27239 **DESCRIPTION**

27240 The *nice()* function shall add the value of *incr* to the nice value of the calling process. A process' nice value is a non-negative number for which a more positive value results in less favorable scheduling.

27243 A maximum nice value of  $2*\{\text{NZERO}\}-1$  and a minimum nice value of 0 are imposed by the system. Requests for values above or below these limits result in the nice value being set to the corresponding limit. Only a process with appropriate privileges can lower the nice value.

27246 PS|TPS Calling the *nice()* function has no effect on the priority of processes or threads with policy SCHED\_FIFO or SCHED\_RR. The effect on processes or threads with other scheduling policies is implementation-defined.

27249 The nice value set with *nice()* shall be applied to the process. If the process is multi-threaded, the nice value shall affect all system scope threads in the process.

27251 As  $-1$  is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *nice()*, and if it returns  $-1$ , check to see if *errno* is non-zero.

27254 **RETURN VALUE**

27255 Upon successful completion, *nice()* shall return the new nice value  $-\{\text{NZERO}\}$ . Otherwise,  $-1$  shall be returned, the process' nice value shall not be changed, and *errno* shall be set to indicate the error.

27258 **ERRORS**27259 The *nice()* function shall fail if:

27260 [EPERM] The *incr* argument is negative and the calling process does not have appropriate privileges.

27262 **EXAMPLES**27263 **Changing the Nice Value**

27264 The following example adds the value of the *incr* argument,  $-20$ , to the nice value of the calling process.

27266 `#include <unistd.h>`27267 `...`27268 `int incr = -20;`27269 `int ret;`27270 `ret = nice(incr);`27271 **APPLICATION USAGE**

27272 None.

27273 **RATIONALE**

27274 None.

27275 **FUTURE DIRECTIONS**

27276 None.

27277 **SEE ALSO**

27278 The Base Definitions volume of IEEE Std. 1003.1-200x, &lt;limits.h&gt;, &lt;unistd.h&gt;

27279 **CHANGE HISTORY**

27280 First released in Issue 1. Derived from Issue 1 of the SVID.

27281 **Issue 4**

27282 The &lt;unistd.h&gt; header is added to the SYNOPSIS section.

27283 A statement is added to the DESCRIPTION indicating that the nice value can only be lowered by  
27284 a process with appropriate privileges.27285 **Issue 4, Version 2**27286 The RETURN VALUE section is updated for X/OPEN UNIX conformance to define that the  
27287 process' nice value is not changed if an error is detected.27288 **Issue 5**27289 A statement is added to the description indicating the effects of this function on the different  
27290 scheduling policies and multi-threaded processes.

27291 **NAME**

27292 nl\_langinfo — language information

27293 **SYNOPSIS**

27294 XSI #include &lt;langinfo.h&gt;

27295 char \*nl\_langinfo(nl\_item item);

27296

27297 **DESCRIPTION**

27298 The *nl\_langinfo()* function shall return a pointer to a string containing information relevant to  
 27299 the particular language or cultural area defined in the program's locale (see <langinfo.h>). The  
 27300 manifest constant names and values of *item* are defined in <langinfo.h>. For example:

27301 nl\_langinfo(ABDAY\_1)

27302 would return a pointer to the string "Dom" if the identified language was Portuguese, and  
 27303 "Sun" if the identified language was English.

27304 Calls to *setlocale()* with a category corresponding to the category of *item* (see <langinfo.h>), or to  
 27305 the category *LC\_ALL*, may overwrite the array pointed to by the return value.

27306 The *nl\_langinfo()* function need not be reentrant. A function that is not required to be reentrant is  
 27307 not required to be thread-safe.

27308 **RETURN VALUE**

27309 In a locale where *langinfo* data is not defined, *nl\_langinfo()* shall return a pointer to the  
 27310 corresponding string in the POSIX locale. In all locales, *nl\_langinfo()* shall return a pointer to an  
 27311 empty string if *item* contains an invalid setting.

27312 This pointer may point to static data that may be overwritten on the next call.

27313 **ERRORS**

27314 No errors are defined.

27315 **EXAMPLES**27316 **Getting Date and Time Formatting Information**

27317 The following example returns a pointer to a string containing date and time formatting  
 27318 information, as defined in the *LC\_TIME* category of the current locale.

27319 #include &lt;time.h&gt;

27320 #include &lt;langinfo.h&gt;

27321 ...

27322 strftime(datestring, sizeof(datestring), nl\_langinfo(D\_T\_FMT), tm);

27323 ...

27324 **APPLICATION USAGE**

27325 The array pointed to by the return value should not be modified by the program, but may be  
 27326 modified by further calls to *nl\_langinfo()*.

27327 **RATIONALE**

27328 None.

27329 **FUTURE DIRECTIONS**

27330 None.

27331 **SEE ALSO**

27332 *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <langinfo.h>, <nl\_types.h>, the  
27333 Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

27334 **CHANGE HISTORY**

27335 First released in Issue 2.

27336 **Issue 4**

27337 The <nl\_types.h> header is removed from the SYNOPSIS section.

27338 **Issue 5**

27339 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

27340 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

27341 **NAME**

27342       nrand48 — generate uniformly distributed pseudo-random non-negative long integers

27343 **SYNOPSIS**

27344 xSI     #include <stdlib.h>

27345       long nrand48(unsigned short xsubi[3]);

27346

27347 **DESCRIPTION**

27348       Refer to *drand48()*.



27349 **NAME**

27350           ntohl — convert values between host and network byte order

27351 **SYNOPSIS**

27352           #include <arpa/inet.h>

27353           uint32\_t ntohl(uint32\_t *netlong*);

27354 **DESCRIPTION**

27355           Refer to *htonl()*.

27356 **NAME**

27357       ntohs — convert values between host and network byte order

27358 **SYNOPSIS**

27359       #include <arpa/inet.h>

27360       uint16\_t ntohs(uint16\_t *netshort*);

27361 **DESCRIPTION**

27362       Refer to *htonl()*.

## 27363 NAME

27364 open — open a file

## 27365 SYNOPSIS

27366 OH #include &lt;sys/stat.h&gt;

27367 #include &lt;fcntl.h&gt;

27368 int open(const char \*path, int oflag, ... );

## 27369 DESCRIPTION

27370 The *open()* function establishes the connection between a file and a file descriptor. It creates an  
 27371 open file description that refers to a file and a file descriptor that refers to that open file  
 27372 description. The file descriptor is used by other I/O functions to refer to that file. The *path*  
 27373 argument points to a path name naming the file.

27374 The *open()* function shall return a file descriptor for the named file that is the lowest file  
 27375 descriptor not currently open for that process. The open file description is new, and therefore the  
 27376 file descriptor does not share it with any other process in the system. The FD\_CLOEXEC file  
 27377 descriptor flag associated with the new file descriptor shall be cleared.

27378 The file offset used to mark the current position within the file shall be set to the beginning of the  
 27379 file.

27380 The file status flags and file access modes of the open file description shall be set according to  
 27381 the value of *oflag*.

27382 Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list,  
 27383 defined in <fcntl.h>. Applications shall specify exactly one of the first three values (file access  
 27384 modes) below in the value of *oflag*:

27385 O\_RDONLY Open for reading only.

27386 O\_WRONLY Open for writing only.

27387 O\_RDWR Open for reading and writing. The result is undefined if this flag is applied to  
27388 a FIFO.

27389 Any combination of the following may be used:

27390 O\_APPEND If set, the file offset shall be set to the end of the file prior to each write.

27391 O\_CREAT If the file exists, this flag has no effect except as noted under O\_EXCL below.  
 27392 Otherwise, the file is created; the user ID of the file shall be set to the effective  
 27393 user ID of the process; the group ID of the file shall be set to the group ID of  
 27394 the file's parent directory or to the effective group ID of the process; and the  
 27395 access permission bits (see <sys/stat.h>) of the file mode shall be set to the  
 27396 value of the third argument taken as type **mode\_t** modified as follows: a  
 27397 bitwise AND is performed on the file-mode bits and the corresponding bits in  
 27398 the complement of the process' file mode creation mask. Thus, all bits in the  
 27399 file mode whose corresponding bit in the file mode creation mask is set are  
 27400 cleared. When bits other than the file permission bits are set, the effect is  
 27401 unspecified. The third argument does not affect whether the file is open for  
 27402 reading, writing, or for both.

27403 SIO O\_DSYNC Write I/O operations on the file descriptor complete as defined by  
27404 synchronized I/O data integrity completion

27405 O\_EXCL If O\_CREAT and O\_EXCL are set, *open()* shall fail if the file exists. The check  
 27406 for the existence of the file and the creation of the file if it does not exist shall  
 27407 be atomic with respect to other threads executing *open()* naming the same file

|           |            |                                                                                                                            |
|-----------|------------|----------------------------------------------------------------------------------------------------------------------------|
| 27408     |            | name in the same directory with O_EXCL and O_CREAT set. If O_EXCL and                                                      |
| 27409     |            | O_CREAT are set, and <i>path</i> names a symbolic link, <i>open()</i> shall fail and set                                   |
| 27410     |            | <i>errno</i> to [EEXIST], regardless of the contents of the symbolic link. If O_EXCL is                                    |
| 27411     |            | set and O_CREAT is not set, the result is undefined.                                                                       |
| 27412     | O_NOCTTY   | If set and <i>path</i> identify a terminal device, <i>open()</i> shall not cause the terminal                              |
| 27413     |            | device to become the controlling terminal for the process.                                                                 |
| 27414     | O_NONBLOCK | When opening a FIFO with O_RDONLY or O_WRONLY set:                                                                         |
| 27415     |            | • If O_NONBLOCK is set, an <i>open()</i> for reading-only shall return without                                             |
| 27416     |            | delay. An <i>open()</i> for writing-only shall return an error if no process                                               |
| 27417     |            | currently has the file open for reading.                                                                                   |
| 27418     |            | • If O_NONBLOCK is clear, an <i>open()</i> for reading-only shall block the                                                |
| 27419     |            | calling thread until a thread opens the file for writing. An <i>open()</i> for                                             |
| 27420     |            | writing-only shall block the calling thread until a thread opens the file for                                              |
| 27421     |            | reading.                                                                                                                   |
| 27422     |            | When opening a block special or character special file that supports non-                                                  |
| 27423     |            | blocking opens:                                                                                                            |
| 27424     |            | • If O_NONBLOCK is set, the <i>open()</i> function shall return without blocking                                           |
| 27425     |            | for the device to be ready or available. Subsequent behavior of the device                                                 |
| 27426     |            | is device-specific.                                                                                                        |
| 27427     |            | • If O_NONBLOCK is clear, the <i>open()</i> function shall block the calling                                               |
| 27428     |            | thread until the device is ready or available before returning.                                                            |
| 27429     |            | Otherwise, the behavior of O_NONBLOCK is unspecified.                                                                      |
| 27430 SIO | O_RSYNC    | Read I/O operations on the file descriptor complete at the same level of                                                   |
| 27431     |            | integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC                                                    |
| 27432     |            | and O_RSYNC are set in <i>oflag</i> , all I/O operations on the file descriptor                                            |
| 27433     |            | complete as defined by synchronized I/O data integrity completion. If both                                                 |
| 27434     |            | O_SYNC and O_RSYNC are set in flags, all I/O operations on the file                                                        |
| 27435     |            | descriptor complete as defined by synchronized I/O file integrity completion.                                              |
| 27436 SIO | O_SYNC     | Write I/O operations on the file descriptor complete as defined by                                                         |
| 27437     |            | synchronized I/O file integrity completion.                                                                                |
| 27438     | O_TRUNC    | If the file exists and is a regular file, and the file is successfully opened                                              |
| 27439     |            | O_RDWR or O_WRONLY, its length is truncated to 0, and the mode and                                                         |
| 27440     |            | owner are unchanged. It shall have no effect on FIFO special files or terminal                                             |
| 27441     |            | device files. Its effect on other file types is implementation-defined. The result                                         |
| 27442     |            | of using O_TRUNC with O_RDONLY is undefined.                                                                               |
| 27443     |            | If O_CREAT is set and the file did not previously exist, upon successful completion, <i>open()</i> shall                   |
| 27444     |            | mark for update the <i>st_atime</i> , <i>st_ctime</i> , and <i>st_mtime</i> fields of the file and the <i>st_ctime</i> and |
| 27445     |            | <i>st_mtime</i> fields of the parent directory.                                                                            |
| 27446     |            | If O_TRUNC is set and the file did previously exist, upon successful completion, <i>open()</i> shall                       |
| 27447     |            | mark for update the <i>st_ctime</i> and <i>st_mtime</i> fields of the file.                                                |
| 27448 SIO |            | If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.                            |
| 27449     |            |                                                                                                                            |
| 27450 XSR |            | If <i>path</i> refers to a STREAMS file, <i>oflag</i> may be constructed from O_NONBLOCK OR'ed with                        |
| 27451     |            | either O_RDONLY, O_WRONLY, or O_RDWR. Other flag values are not applicable to                                              |
| 27452     |            | STREAMS devices and have no effect on them. The value O_NONBLOCK affects the operation                                     |

|       |                     |                                                                                                                   |
|-------|---------------------|-------------------------------------------------------------------------------------------------------------------|
| 27453 |                     | of STREAMS drivers and certain functions applied to file descriptors associated with STREAMS                      |
| 27454 |                     | files. For STREAMS drivers, the implementation of O_NONBLOCK is device-specific.                                  |
| 27455 |                     | If <i>path</i> names the master side of a pseudo-terminal device, then it is unspecified whether <i>open()</i>    |
| 27456 |                     | locks the slave side so that it cannot be opened. Portable applications shall call <i>unlockpt()</i> before       |
| 27457 |                     | opening the slave side.                                                                                           |
| 27458 |                     | The largest value that can be represented correctly in an object of type <b>off_t</b> shall be established        |
| 27459 |                     | as the offset maximum in the open file description.                                                               |
| 27460 | <b>RETURN VALUE</b> |                                                                                                                   |
| 27461 |                     | Upon successful completion, the function shall open the file and return a non-negative integer                    |
| 27462 |                     | representing the lowest numbered unused file descriptor. Otherwise, <b>-1</b> shall be returned and               |
| 27463 |                     | <i>errno</i> set to indicate the error. No files shall be created or modified if the function returns <b>-1</b> . |
| 27464 | <b>ERRORS</b>       |                                                                                                                   |
| 27465 |                     | The <i>open()</i> function shall fail if:                                                                         |
| 27466 | [EACCES]            | Search permission is denied on a component of the path prefix, or the file                                        |
| 27467 |                     | exists and the permissions specified by <i>oflag</i> are denied, or the file does not                             |
| 27468 |                     | exist and write permission is denied for the parent directory of the file to be                                   |
| 27469 |                     | created, or O_TRUNC is specified and write permission is denied.                                                  |
| 27470 | [EEXIST]            | O_CREAT and O_EXCL are set, and the named file exists.                                                            |
| 27471 | [EINTR]             | A signal was caught during <i>open()</i> .                                                                        |
| 27472 | SIO [EINVAL]        | The implementation does not support synchronized I/O for this file.                                               |
| 27473 | XSR [EIO]           | The <i>path</i> argument names a STREAMS file and a hangup or error occurred                                      |
| 27474 |                     | during the <i>open()</i> .                                                                                        |
| 27475 | [EISDIR]            | The named file is a directory and <i>oflag</i> includes O_WRONLY or O_RDWR.                                       |
| 27476 | [ELOOP]             | A loop exists in symbolic links encountered during resolution of the <i>path</i>                                  |
| 27477 |                     | argument.                                                                                                         |
| 27478 | [EMFILE]            | {OPEN_MAX} file descriptors are currently open in the calling process.                                            |
| 27479 | [ENAMETOOLONG]      |                                                                                                                   |
| 27480 |                     | The length of the <i>path</i> argument exceeds {PATH_MAX} or a path name                                          |
| 27481 |                     | component is longer than {NAME_MAX}.                                                                              |
| 27482 | [ENFILE]            | The maximum allowable number of files is currently open in the system.                                            |
| 27483 | [ENOENT]            | O_CREAT is not set and the named file does not exist; or O_CREAT is set and                                       |
| 27484 |                     | either the path prefix does not exist or the <i>path</i> argument points to an empty                              |
| 27485 |                     | string.                                                                                                           |
| 27486 | XSR [ENOSR]         | The <i>path</i> argument names a STREAMS-based file and the system is unable to                                   |
| 27487 |                     | allocate a STREAM.                                                                                                |
| 27488 | [ENOSPC]            | The directory or file system that would contain the new file cannot be                                            |
| 27489 |                     | expanded, the file does not exist, and O_CREAT is specified.                                                      |
| 27490 | [ENOTDIR]           | A component of the path prefix is not a directory.                                                                |
| 27491 | [ENXIO]             | O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no                                              |
| 27492 |                     | process has the file open for reading.                                                                            |
| 27493 | [ENXIO]             | The named file is a character special or block special file, and the device                                       |
| 27494 |                     | associated with this special file does not exist.                                                                 |

|           |                |                                                                                                                                                                                                                        |
|-----------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 27495     | [EOVERFLOW]    | The named file is a regular file and the size of the file cannot be represented correctly in an object of type <code>off_t</code> .                                                                                    |
| 27496     |                |                                                                                                                                                                                                                        |
| 27497     | [EROFS]        | The named file resides on a read-only file system and either <code>O_WRONLY</code> , <code>O_RDWR</code> , <code>O_CREAT</code> (if file does not exist), or <code>O_TRUNC</code> is set in the <i>oflag</i> argument. |
| 27498     |                |                                                                                                                                                                                                                        |
| 27499     |                |                                                                                                                                                                                                                        |
| 27500     |                | The <code>open()</code> function may fail if:                                                                                                                                                                          |
| 27501 XSR | [EAGAIN]       | The <i>path</i> argument names the slave side of a pseudo-terminal device that is locked.                                                                                                                              |
| 27502     |                |                                                                                                                                                                                                                        |
| 27503     | [EINVAL]       | The value of the <i>oflag</i> argument is not valid.                                                                                                                                                                   |
| 27504     | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.                                                                                                                 |
| 27505     |                |                                                                                                                                                                                                                        |
| 27506     | [ENAMETOOLONG] | As a result of encountering a symbolic link in resolution of the <i>path</i> argument, the length of the substituted path name string exceeded {PATH_MAX}.                                                             |
| 27507     |                |                                                                                                                                                                                                                        |
| 27508     |                |                                                                                                                                                                                                                        |
| 27509 XSR | [ENOMEM]       | The <i>path</i> argument names a STREAMS file and the system is unable to allocate resources.                                                                                                                          |
| 27510     |                |                                                                                                                                                                                                                        |
| 27511     | [ETXTBSY]      | The file is a pure procedure (shared text) file that is being executed and <i>oflag</i> is <code>O_WRONLY</code> or <code>O_RDWR</code> .                                                                              |
| 27512     |                |                                                                                                                                                                                                                        |

## 27513 EXAMPLES

### 27514 Opening a File for Writing by the Owner

27515 The following example opens the file `/tmp/file`, either by creating it (if it does not already exist),  
 27516 or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file,  
 27517 the access permission bits in the file mode of the file are set to permit reading and writing by the  
 27518 owner, and to permit reading only by group members and others.

27519 If the call to `open()` is successful, the file is opened for writing.

```
27520 #include <fcntl.h>
27521 ...
27522 int fd;
27523 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
27524 char *filename = "/tmp/file";
27525 ...
27526 fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, mode);
27527 ...
```

### 27528 Opening a File Using an Existence Check

27529 The following example uses the `open()` function to try to create the `LOCKFILE` file and open it  
 27530 for writing. Because the `open()` function specifies the `O_EXCL` flag, the call fails if the file already  
 27531 exists. In that case, the program assumes that someone else is updating the password file and  
 27532 exits.

```
27533 #include <fcntl.h>
27534 #include <stdio.h>
27535 #include <stdlib.h>
```

```

27536 #define LOCKFILE "/etc/ptmp"
27537 ...
27538 int pfd; /* Integer for file descriptor returned by open() call. */
27539 ...
27540 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
27541 S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
27542 {
27543 fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
27544 exit(1);
27545 }
27546 ...

```

### 27547 **Opening a File for Writing**

27548 The following example opens a file for writing, creating the file if it does not already exist. If the  
 27549 file does exist, the system truncates the file to zero bytes.

```

27550 #include <fcntl.h>
27551 #include <stdio.h>
27552 #include <stdlib.h>
27553
27553 #define LOCKFILE "/etc/ptmp"
27554 ...
27555 int pfd;
27556 char filename[PATH_MAX+1];
27557 ...
27558 if ((pfd = open(filename, O_WRONLY | O_CREAT | O_TRUNC,
27559 S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
27560 {
27561 perror("Cannot open output file\n"); exit(1);
27562 }
27563 ...

```

### 27564 **APPLICATION USAGE**

27565 None.

### 27566 **RATIONALE**

27567 Except as specified in this volume of IEEE Std. 1003.1-200x, the flags allowed in *oflag* are not  
 27568 mutually-exclusive and any number of them may be used simultaneously.

27569 Some implementations permit opening FIFOs with O\_RDWR. Since FIFOs could be  
 27570 implemented in other ways, and since two file descriptors can be used to the same effect, this  
 27571 possibility is left as undefined.

27572 See *getgroups()* about the group of a newly created file.

27573 The use of *open()* to create a regular file is preferable to the use of *creat()*, because the latter is  
 27574 redundant and included only for historical reasons.

27575 The use of the O\_TRUNC flag on FIFOs and directories (pipes cannot be *open()*-ed) must be  
 27576 permissible without unexpected side effects (for example, *creat()* on a FIFO must not remove  
 27577 data). Because terminal special files might have type-ahead data stored in the buffer, O\_TRUNC  
 27578 should not affect their content, particularly if a program that normally opens a regular file  
 27579 should open the current controlling terminal instead. Other file types, particularly  
 27580 implementation-defined ones, are left implementation-defined.

27581 IEEE Std. 1003.1-200x permits [EACCES] to be returned for conditions other than those explicitly  
27582 listed.

27583 The O\_NOCTTY flag was added to allow applications to avoid unintentionally acquiring a  
27584 controlling terminal as a side effect of opening a terminal file. This volume of  
27585 IEEE Std. 1003.1-200x does not specify how a controlling terminal is acquired, but it allows an  
27586 implementation to provide this on *open()* if the O\_NOCTTY flag is not set and other conditions  
27587 specified in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal  
27588 Interface are met. The O\_NOCTTY flag is an effective no-op if the file being opened is not a  
27589 terminal device.

27590 In historical implementations the value of O\_RDONLY is zero. Because of that, it is not possible  
27591 to detect the presence of O\_RDONLY and another option. Future implementations should  
27592 encode O\_RDONLY and O\_WRONLY as bit flags so that:

27593 O\_RDONLY | O\_WRONLY == O\_RDWR

27594 In general, the *open()* function follows the symbolic link if *path* names a symbolic link. However,  
27595 the *open()* function, when called with O\_CREAT and O\_EXCL, is required to fail with [EEXIST]  
27596 if *path* names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This  
27597 behavior is required so that privileged applications can create a new file in a known location  
27598 without the possibility that a symbolic link might cause the file to be created in a different  
27599 location.

27600 For example, a privileged application that must create a file with a predictable name in a user-  
27601 writable directory, such as the user's home directory, could be compromised if the user creates a  
27602 symbolic link with that name that refers to a nonexistent file in a system directory. If the user can  
27603 influence the contents of a file, the user could compromise the system by creating a new system  
27604 configuration or spool file that would then be interpreted by the system. The test for a symbolic  
27605 link which refers to a nonexistent file must be atomic with the creation of a new file.

#### 27606 FUTURE DIRECTIONS

27607 None.

#### 27608 SEE ALSO

27609 *chmod()*, *close()*, *creat()*, *dup()*, *fcntl()*, *lseek()*, *read()*, *umask()*, *unlockpt()*, *write()*, the Base  
27610 Definitions volume of IEEE Std. 1003.1-200x, <fcntl.h>, <sys/stat.h>, <sys/types.h>

#### 27611 CHANGE HISTORY

27612 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 27613 Issue 4

27614 The <sys/types.h> and <sys/stat.h> headers are now marked as optional (OH); these headers do  
27615 not need to be included on XSI-conformant systems.

27616 O\_NDELAY is removed from the list of *oflag* values (this flag was marked WITHDRAWN in  
27617 Issue 3).

27618 The [ENXIO] error (for the condition where the file is a character or block special file and the  
27619 associated device does not exist) and the [EINVAL] error are marked as extensions.

27620 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 27621 • The type of argument *path* is changed from **char\*** to **const char\***.
- 27622 • Various wording changes are made to the DESCRIPTION to improve clarity and to align the  
27623 text with the ISO POSIX-1 standard.

27624 The following changes are incorporated for alignment with the FIPS requirements:



- 27625           • In the DESCRIPTION, the description of O\_CREAT is amended and the relevant part marked  
27626           as an extension.
- 27627           • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path  
27628           name component is larger than {NAME\_MAX} is now defined as mandatory and marked as  
27629           an extension.
- 27630 **Issue 4, Version 2**
- 27631           The following changes are incorporated for X/OPEN UNIX conformance:
- 27632           • The DESCRIPTION is updated to define the use of open flags with STREAMS files, and to  
27633           identify special considerations when opening the master side of a pseudo-terminal.
- 27634           • In the ERRORS section, the [EIO], [ELOOP], and [ENOSR] mandatory error conditions are  
27635           added, and the [EAGAIN], [ENAMETOOLONG], and [ENOMEM] optional error conditions  
27636           are added.
- 27637 **Issue 5**
- 27638           The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
27639           Threads Extension.
- 27640           Large File Summit extensions are added.
- 27641 **Issue 6**
- 27642           In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.
- 27643           The following changes are made for alignment with the ISO POSIX-1: 1996 standard:
- 27644           • The [ENAMETOOLONG] error is restored as an error dependent on \_POSIX\_NO\_TRUNC.  
27645           This is since behavior may vary from one file system to another.
- 27646           The following new requirements on POSIX implementations derive from alignment with the  
27647           Single UNIX Specification:
- 27648           • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
27649           required for conforming implementations of previous POSIX specifications, it was not  
27650           required for UNIX applications.
- 27651           • In the DESCRIPTION, O\_CREAT is amended to state that the group ID of the file is set to the  
27652           group ID of the file's parent directory or to the effective group ID of the process. This is a  
27653           FIPS requirement.
- 27654           • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file  
27655           description. This change is to support large files.
- 27656           • In the ERRORS section, the [E\_OVERFLOW] condition is added. This change is to support  
27657           large files.
- 27658           • The [ENXIO] mandatory error condition is added.
- 27659           • The [EINVAL], [ENAMETOOLONG], and [ETXTBSY] optional error conditions are added.
- 27660           The DESCRIPTION and ERRORS sections are updated so that items related to the optional XSI  
27661           STREAMS Option Group are marked.
- 27662           The following changes were made to align with the IEEE P1003.1a draft standard:
- 27663           • An explanation is added of the effect of the O\_CREAT and O\_EXCL flags when the path  
27664           refers to a symbolic link.
- 27665           • The [ELOOP] optional error condition is added.

27666 The DESCRIPTION is updated to avoid use of the term “must” for application requirements. |

27667 The DESCRIPTION of O\_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48. |

27668 **NAME**

27669            opendir — open a directory

27670 **SYNOPSIS**

27671            #include &lt;dirent.h&gt;

27672            DIR \*opendir(const char \*dirname);

27673 **DESCRIPTION**

27674            The *opendir()* function shall open a directory stream corresponding to the directory named by  
 27675            the *dirname* argument. The directory stream is positioned at the first entry. If the type **DIR** is  
 27676            implemented using a file descriptor, applications shall only be able to open up to a total of  
 27677            {OPEN\_MAX} files and directories.

27678 **RETURN VALUE**

27679            Upon successful completion, *opendir()* shall return a pointer to an object of type **DIR**.  
 27680            Otherwise, a null pointer shall be returned and *errno* set to indicate the error.

27681 **ERRORS**27682            The *opendir()* function shall fail if:

27683            [EACCES]            Search permission is denied for the component of the path prefix of *dirname* or  
 27684            read permission is denied for *dirname*.

27685            [ELOOP]            A loop exists in symbolic links encountered during resolution of the *dirname*  
 27686            argument.

27687            [ENAMETOOLONG]       The length of the *dirname* argument exceeds {PATH\_MAX} or a path name  
 27688            component is longer than {NAME\_MAX}.

27690            [ENOENT]           A component of *dirname* does not name an existing directory or *dirname* is an  
 27691            empty string.

27692            [ENOTDIR]           A component of *dirname* is not a directory.

27693            The *opendir()* function may fail if:

27694            [ELOOP]            More than {SYMLOOP\_MAX} symbolic links were encountered during  
 27695            resolution of the *dirname* argument.

27696            [EMFILE]            {OPEN\_MAX} file descriptors are currently open in the calling process.

27697            [ENAMETOOLONG]       As a result of encountering a symbolic link in resolution of the *dirname*  
 27698            argument, the length of the substituted path name string exceeded  
 27699            {PATH\_MAX}.

27701            [ENFILE]           Too many files are currently open in the system.

27702 **EXAMPLES**

27703 **Open a Directory Stream**

27704 The following program fragment demonstrates how the *opendir()* function is used.

```

27705 #include <sys/types.h>
27706 #include <dirent.h>
27707 #include <libgen.h>
27708 ...
27709 DIR *dir;
27710 struct dirent *dp;
27711 ...
27712 if ((dir = opendir(".")) == NULL) {
27713 perror("Cannot open .");
27714 exit(1);
27715 }
27716 while ((dp = readdir(dir)) != NULL) {
27717 ...

```

27718 **APPLICATION USAGE**

27719 The *opendir()* function should be used in conjunction with *readdir()*, *closedir()*, and *rewinddir()* to  
 27720 examine the contents of the directory (see the EXAMPLES section in *readdir()*). This method is  
 27721 recommended for portability.

27722 **RATIONALE**

27723 Based on historical implementations, the rules about file descriptors apply to directory streams  
 27724 as well. However, this volume of IEEE Std. 1003.1-200x does not mandate that the directory  
 27725 stream be implemented using file descriptors. The description of *closedir()* clarifies that if a file  
 27726 descriptor is used for the directory stream, it is mandatory that *closedir()* deallocate the file  
 27727 descriptor. When a file descriptor is used to implement the directory stream, it behaves as if the  
 27728 FD\_CLOEXEC had been set for the file descriptor.

27729 The directory entries for dot and dot-dot are optional. This volume of IEEE Std. 1003.1-200x does  
 27730 not provide a way to test *a priori* for their existence because an application that is portable must  
 27731 be written to look for (and usually ignore) those entries. Writing code that presumes that they  
 27732 are the first two entries does not always work, as many implementations permit them to be  
 27733 other than the first two entries, with a “normal” entry preceding them. There is negligible value  
 27734 in providing a way to determine what the implementation does because the code to deal with  
 27735 dot and dot-dot must be written in any case and because such a flag would add to the list of  
 27736 those flags (which has proven in itself to be objectionable) and might be abused.

27737 Since the structure and buffer allocation, if any, for directory operations are defined by the  
 27738 implementation, this volume of IEEE Std. 1003.1-200x imposes no portability requirements for  
 27739 erroneous program constructs, erroneous data, or the use of indeterminate values such as the  
 27740 use or referencing of a *dirp* value or a **dirent** structure value after a directory stream has been  
 27741 closed or after a *fork()* or one of the *exec* function calls.

27742 **FUTURE DIRECTIONS**

27743 None.

27744 **SEE ALSO**

27745 *closedir()*, *lstat()*, *readdir()*, *rewinddir()*, *symlink()*, the Base Definitions volume of  
 27746 IEEE Std. 1003.1-200x, **<dirent.h>**, **<limits.h>**, **<sys/types.h>**

27747 **CHANGE HISTORY**

27748 First released in Issue 2.

27749 **Issue 4**27750 The `<sys/types.h>` header is now marked as optional (OH); this header need not be included on  
27751 XSI-conformant systems.27752 In the DESCRIPTION, the following sentence is moved to the Base Definitions volume of  
27753 IEEE Std. 1003.1-200x: “The type **DIR**, which is defined in `<dirent.h>`, represents a *directory*  
27754 *stream*, which is an ordered sequence of all directory entries in a particular directory.”

27755 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 27756
- The type of argument *dirname* is changed from `char*` to `const char*`.
  - The generation of an [ENOENT] error when *dirname* points to an empty string is made  
27757 mandatory.  
27758

27759 The following change is incorporated for alignment with the FIPS requirements:

- 27760
- In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path  
27761 name component is larger than {NAME\_MAX} is now defined as mandatory and marked as  
27762 an extension.

27763 **Issue 4, Version 2**

27764 The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- 27765
- It states that [ELOOP] is returned if too many symbolic links are encountered during path  
27766 name resolution.
  - A second [ENAMETOOLONG] condition is defined that may report excessive length of an  
27767 intermediate result of path name resolution of a symbolic link.  
27768

27769 **Issue 6**27770 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

27771 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 27772
- The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.  
27773 This is since behavior may vary from one file system to another.

27774 The following new requirements on POSIX implementations derive from alignment with the  
27775 Single UNIX Specification:

- 27776
- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
27777 required for conforming implementations of previous POSIX specifications, it was not  
27778 required for UNIX applications.
  - The [ELOOP] mandatory error condition is added.  
27779
  - A second [ENAMETOOLONG] is added as an optional error condition.  
27780

27781 The following changes were made to align with the IEEE P1003.1a draft standard:

- 27782
- The [ELOOP] optional error condition is added.

27783 **NAME**

27784            **openlog** — open a connection to the logging facility

27785 **SYNOPSIS**

27786 XSI        #include <syslog.h>

27787            void openlog(const char \**ident*, int *logopt*, int *facility*);

27788

27789 **DESCRIPTION**

27790            Refer to *closelog*().

27791 **NAME**

27792           optarg, opterr, optind, optopt — options parsing variables

27793 **SYNOPSIS**

27794           #include <unistd.h>

27795           extern char \*optarg;

27796           extern int opterr, optind, optopt;

27797 **DESCRIPTION**

27798           Refer to *getopt()*.

27799 **NAME**

27800 pathconf — get configurable path name variables

27801 **SYNOPSIS**

27802 #include <unistd.h>

27803 long pathconf(const char \**path*, int *name*);

27804 **DESCRIPTION**

27805 Refer to *fpathconf()*.



27806 **NAME**

27807           *pause* — suspend the thread until a signal is received

27808 **SYNOPSIS**

27809           #include <unistd.h>

27810           int *pause*(void);

27811 **DESCRIPTION**

27812           The *pause*() function shall suspend the calling thread until delivery of a signal whose action is  
27813           either to execute a signal-catching function or to terminate the process.

27814           If the action is to terminate the process, *pause*() shall not return.

27815           If the action is to execute a signal-catching function, *pause*() shall return after the signal-catching  
27816           function returns.

27817 **RETURN VALUE**

27818           Since *pause*() suspends thread execution indefinitely unless interrupted by a signal, there is no  
27819           successful completion return value. A value of -1 shall be returned and *errno* set to indicate the  
27820           error.

27821 **ERRORS**

27822           The *pause*() function shall fail if:

27823           [EINTR]           A signal is caught by the calling process and control is returned from the  
27824           signal-catching function.

27825 **EXAMPLES**

27826           None.

27827 **APPLICATION USAGE**

27828           Many common uses of *pause*() have timing windows. The scenario involves checking a  
27829           condition related to a signal and, if the signal has not occurred, calling *pause*(). When the signal  
27830           occurs between the check and the call to *pause*(), the process often blocks indefinitely. The  
27831           *sigprocmask*() and *sigsuspend*() functions can be used to avoid this type of problem.

27832 **RATIONALE**

27833           None.

27834 **FUTURE DIRECTIONS**

27835           None.

27836 **SEE ALSO**

27837           *sigsuspend*(), the Base Definitions volume of IEEE Std. 1003.1-200x, <unistd.h>

27838 **CHANGE HISTORY**

27839           First released in Issue 1. Derived from Issue 1 of the SVID.

27840 **Issue 4**

27841           The <unistd.h> header is added to the SYNOPSIS section.

27842           In the RETURN VALUE section, the text is expanded to indicate that process execution is  
27843           suspended indefinitely “unless interrupted by a signal”.

27844           The following change is incorporated for alignment with the ISO POSIX-1 standard:

27845           • The argument list is explicitly defined as **void**.

27846 **Issue 5**

27847 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

27848 **Issue 6**

27849 The APPLICATION USAGE section is added.

27850 **NAME**

27851 pclose — close a pipe stream to or from a process

27852 **SYNOPSIS**

27853 #include &lt;stdio.h&gt;

27854 int pclose(FILE \**stream*);27855 **DESCRIPTION**

27856 The *pclose()* function shall close a stream that was opened by *popen()*, wait for the command to  
 27857 terminate, and return the termination status of the process that was running the command  
 27858 language interpreter. However, if a call caused the termination status to be unavailable to  
 27859 *pclose()*, then *pclose()* shall return  $-1$  with *errno* set to [ECHILD] to report this situation. This can  
 27860 happen if the application calls one of the following functions:

- 27861 • *wait()*
- 27862 • *waitpid()* with a *pid* argument less than or equal to 0 or equal to the process ID of the  
 27863 command line interpreter
- 27864 • Any other function not defined in this volume of IEEE Std. 1003.1-200x that could do one of  
 27865 the above

27866 In any case, *pclose()* shall not return before the child process created by *popen()* has terminated.

27867 If the command language interpreter cannot be executed, the child termination status returned  
 27868 by *pclose()* is as if the command language interpreter terminated using *exit(127)* or *\_exit(127)*.

27869 The *pclose()* function shall not affect the termination status of any child of the calling process  
 27870 other than the one created by *popen()* for the associated stream.

27871 If the argument *stream* to *pclose()* is not a pointer to a stream created by *popen()*, the result of  
 27872 *pclose()* is undefined.

27873 **RETURN VALUE**

27874 Upon successful return, *pclose()* shall return the termination status of the command language  
 27875 interpreter. Otherwise, *pclose()* shall return  $-1$  and set *errno* to indicate the error.

27876 **ERRORS**

27877 The *pclose()* function shall fail if:

27878 [ECHILD] The status of the child process could not be obtained, as described above.

27879 **EXAMPLES**

27880 None.

27881 **APPLICATION USAGE**

27882 None.

27883 **RATIONALE**

27884 There is a requirement that *pclose()* not return before the child process terminates. This is  
 27885 intended to disallow implementations that return [EINTR] if a signal is received while waiting.  
 27886 If *pclose()* returned before the child terminated, there would be no way for the application to  
 27887 discover which child used to be associated with the stream, and it could not do the cleanup  
 27888 itself.

27889 If the stream pointed to by *stream* was not created by *popen()*, historical implementations of  
 27890 *pclose()* return  $-1$  without setting *errno*. To avoid requiring *pclose()* to set *errno* in this case,  
 27891 IEEE Std. 1003.1-200x makes the behavior unspecified. An application should not use *pclose()* to  
 27892 close any stream that was not created by *popen()*.

27893 Some historical implementations of *pclose()* either block or ignore the signals SIGINT, SIGQUIT,  
 27894 and SIGHUP while waiting for the child process to terminate. Since this behavior is not  
 27895 described for the *pclose()* function in IEEE Std. 1003.1-200x, such implementations are not  
 27896 conforming. Also, some historical implementations return [EINTR] if a signal is received, even  
 27897 though the child process has not terminated. Such implementations are also considered non-  
 27898 conforming.

27899 Consider, for example, an application that uses:

```
27900 popen("command", "r")
```

27901 to start *command*, which is part of the same application. The parent writes a prompt to its  
 27902 standard output (presumably the terminal) and then reads from the stream. The child reads the  
 27903 response from the user, does some transformation on the response (path name expansion,  
 27904 perhaps) and writes the result to its standard output. The parent process reads the result from  
 27905 the pipe, does something with it, and prints another prompt. The cycle repeats. Assuming that  
 27906 both processes do appropriate buffer flushing, this would be expected to work.

27907 To conform to IEEE Std. 1003.1-200x, *pclose()* must use *waitpid()*, or some similar function,  
 27908 instead of *wait()*.

27909 The code sample below illustrates how the *pclose()* function might be implemented on a system  
 27910 conforming to IEEE Std. 1003.1-200x.

```
27911 int pclose(FILE *stream)
27912 {
27913 int stat;
27914 pid_t pid;
27915
27916 pid = <pid for process created for stream by popen(>
27917 (void) fclose(stream);
27918 while (waitpid(pid, &stat, 0) == -1) {
27919 if (errno != EINTR){
27920 stat = -1;
27921 break;
27922 }
27923 }
27924 return(stat);
27925 }
```

#### 27925 FUTURE DIRECTIONS

27926 None.

#### 27927 SEE ALSO

27928 *fork()*, *popen()*, *waitpid()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<stdio.h>`

#### 27929 CHANGE HISTORY

27930 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 27931 Issue 4

27932 The following changes are incorporated for alignment with the ISO POSIX-2 standard:

- 27933 • The function is no longer marked as an extension.
- 27934 • The simple DESCRIPTION given in Issue 3 is replaced with a more complete description in  
 27935 this issue. In particular, interactions between this function and *wait()* and *waitpid()* are  
 27936 defined.

27937 **NAME**

27938           perror — write error messages to standard error

27939 **SYNOPSIS**

27940           #include &lt;stdio.h&gt;

27941           void perror(const char \*s);

27942 **DESCRIPTION**

27943 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
 27944       conflict between the requirements described here and the ISO C standard is unintentional. This  
 27945       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

27946       The *perror()* function maps the error number accessed through the symbol *errno* to a language-  
 27947       dependent error message, which shall be written to the standard error stream as follows:

- 27948           • First (if *s* is not a null pointer and the character pointed to by *s* is not the null byte), the string  
 27949           pointed to by *s* followed by a colon and a <space> character.
- 27950           • Then an error message string followed by a <newline> character.

27951       The contents of the error message strings are the same as those returned by *strerror()* with  
 27952       argument *errno*.

27953 cx       The *perror()* function shall mark the file associated with the standard error stream as having  
 27954       been written (*st\_ctime*, *st\_mtime* marked for update) at some time between its successful  
 27955       completion and *exit()*, *abort()*, or the completion of *fflush()* or *fclose()* on *stderr*.

27956       The *perror()* function shall not change the orientation of the standard error stream.

27957 **RETURN VALUE**27958           The *perror()* function shall return no value.27959 **ERRORS**

27960           No errors are defined.

27961 **EXAMPLES**27962           **Printing an Error Message for a Function**

27963       The following example replaces *bufptr* with a buffer that is the necessary size. If an error occurs,  
 27964       the *perror()* function prints a message and the program exits.

```

27965 #include <stdio.h>
27966 #include <stdlib.h>
27967 ...
27968 char *bufptr;
27969 size_t szbuf;
27970 ...
27971 if ((bufptr = malloc(szbuf)) == NULL) {
27972 perror("malloc"); exit(2);
27973 }
27974 ...
```

27975 **APPLICATION USAGE**

27976           None.

27977 **RATIONALE**

27978 None.

27979 **FUTURE DIRECTIONS**

27980 None.

27981 **SEE ALSO**27982 *strerror()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**stdio.h**>27983 **CHANGE HISTORY**

27984 First released in Issue 1. Derived from Issue 1 of the SVID.

27985 **Issue 4**27986 The language for error message strings was given as implementation-defined in Issue 3. In this  
27987 issue, they are defined as language-dependent.

27988 The following change is incorporated for alignment with the ISO POSIX-1 standard:

27989 • A paragraph is added to the DESCRIPTION defining the effects of this function on the  
27990 *st\_ctime* and *st\_mtime* fields of the standard error stream.

27991 The following change is incorporated for alignment with the ISO C standard:

27992 • The type of argument *s* is changed from **char\*** to **const char\***.27993 **Issue 5**27994 A paragraph is added to the DESCRIPTION indicating that *perror()* does not change the  
27995 orientation of the standard error stream.27996 **Issue 6**

27997 Extensions beyond the ISO C standard are now marked.

27998 **NAME**

27999 pipe — create an interprocess channel

28000 **SYNOPSIS**

28001 #include <unistd.h>

28002 int pipe(int *fildes*[2]);

28003 **DESCRIPTION**

28004 The *pipe()* function shall create a pipe and place two file descriptors, one each into the  
28005 arguments *fildes*[0] and *fildes*[1], that refer to the open file descriptions for the read and write  
28006 ends of the pipe. Their integer values shall be the two lowest available at the time of the *pipe()*  
28007 call. The O\_NONBLOCK and FD\_CLOEXEC flags shall be clear on both file descriptors. (The  
28008 *fcntl()* function can be used to set both these flags.)

28009 Data can be written to the file descriptor *fildes*[1] and read from the file descriptor *fildes*[0]. A  
28010 read on the file descriptor *fildes*[0] shall access data written to the file descriptor *fildes*[1] on a  
28011 first-in-first-out basis. It is unspecified whether *fildes*[0] is also open for writing and whether  
28012 *fildes*[1] is also open for reading.

28013 A process has the pipe open for reading (correspondingly writing) if it has a file descriptor open  
28014 that refers to the read end, *fildes*[0] (write end, *fildes*[1]).

28015 Upon successful completion, *pipe()* shall mark for update the *st\_atime*, *st\_ctime*, and *st\_mtime*  
28016 fields of the pipe.

28017 **RETURN VALUE**

28018 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
28019 indicate the error.

28020 **ERRORS**

28021 The *pipe()* function shall fail if:

28022 [EMFILE] More than {OPEN\_MAX} minus two file descriptors are already in use by this  
28023 process.

28024 [ENFILE] The number of simultaneously open files in the system would exceed a  
28025 system-imposed limit.

28026 **EXAMPLES**

28027 None.

28028 **APPLICATION USAGE**

28029 None.

28030 **RATIONALE**

28031 The wording carefully avoids using the verb “to open” in order to avoid any implication of use  
28032 of *open()*; see also *write()*.

28033 **FUTURE DIRECTIONS**

28034 None.

28035 **SEE ALSO**

28036 *fcntl()*, *read()*, *write()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <*fcntl.h*>,  
28037 <*unistd.h*>

28038 **CHANGE HISTORY**

28039 First released in Issue 1. Derived from Issue 1 of the SVID.

28040 **Issue 4**

28041       The <**unistd.h**> header is added to the SYNOPSIS section.

28042 **Issue 4, Version 2**

28043       The DESCRIPTION is updated for X/OPEN UNIX conformance to indicate that certain  
28044       dispositions of *fildev[0]* and *fildev[1]* are unspecified.

28045 **Issue 6**

28046       The following new requirements on POSIX implementations derive from alignment with the  
28047       Single UNIX Specification:

- 28048       • The DESCRIPTION is updated to indicate that certain dispositions of *fildev[0]* and *fildev[1]*  
28049       are unspecified.



## 28050 NAME

28051 poll — input/output multiplexing

## 28052 SYNOPSIS

28053 XSI #include &lt;poll.h&gt;

28054 int poll(struct pollfd *fds*[], nfd\_t *nfds*, int *timeout*);

28055

## 28056 DESCRIPTION

28057 The *poll()* function provides applications with a mechanism for multiplexing input/output over  
 28058 a set of file descriptors. For each member of the array pointed to by *fds*, *poll()* examines the given  
 28059 file descriptor for the event(s) specified in *events*. The number of **pollfd** structures in the *fds*  
 28060 array is specified by *nfds*. The *poll()* function identifies those file descriptors on which an  
 28061 application can read or write data, or on which certain events have occurred.

28062 The *fds* argument specifies the file descriptors to be examined and the events of interest for each  
 28063 file descriptor. It is a pointer to an array with one member for each open file descriptor of  
 28064 interest. The array's members are **pollfd** structures within which *fd* specifies an open file  
 28065 descriptor and *events* and *revents* are bitmasks constructed by OR'ing a combination of the  
 28066 following event flags:

28067 XSR POLLIN Data other than high-priority data may be read without blocking. For  
 28068 STREAMS, this flag is set in *revents* even if the message is of zero length.

28069 XSR POLLRDNORM Normal data (priority band equals 0) may be read without blocking. For  
 28070 STREAMS, this flag is set in *revents* even if the message is of zero length.

28071 XSR POLLRDBAND Data from a non-zero priority band may be read without blocking. For  
 28072 STREAMS, this flag is set in *revents* even if the message is of zero length.

28073 XSR POLLPRI High-priority data may be received without blocking. For STREAMS, this flag  
 28074 is set in *revents* even if the message is of zero length.

28075 POLLOUT Normal data (priority band equals 0) may be written without blocking.

28076 POLLWRNORM Same as POLLOUT.

28077 POLLWRBAND Priority data (priority band >0) may be written. This event only examines  
 28078 bands that have been written to at least once.

28079 POLLERR An error has occurred on the device or stream. This flag is only valid in the  
 28080 *revents* bitmask; it is ignored in the *events* member.

28081 POLLHUP The device has been disconnected. This event and POLLOUT are mutually-  
 28082 exclusive; a stream can never be writable if a hangup has occurred. However,  
 28083 this event and POLLIN, POLLRDNORM, POLLRDBAND, or POLLPRI are not  
 28084 mutually-exclusive. This flag is only valid in the *revents* bitmask; it is ignored  
 28085 in the *events* member.

28086 POLLNVAL The specified *fd* value is invalid. This flag is only valid in the *revents* member;  
 28087 it is ignored in the *events* member.

28088 If the value of *fd* is less than 0, *events* is ignored, and *revents* is set to 0 in that entry on return from  
 28089 *poll()*.

28090 In each **pollfd** structure, *poll()* clears the *revents* member, except that where the application  
 28091 requested a report on a condition by setting one of the bits of *events* listed above, *poll()* sets the  
 28092 corresponding bit in *revents* if the requested condition is true. In addition, *poll()* sets the  
 28093 POLLHUP, POLLERR, and POLLNVAL flag in *revents* if the condition is true, even if the

- 28094 application did not set the corresponding bit in *events*.
- 28095 If none of the defined events have occurred on any selected file descriptor, *poll()* waits at least  
28096 *timeout* milliseconds for an event to occur on any of the selected file descriptors. If the value of  
28097 *timeout* is 0, *poll()* shall return immediately. If the value of *timeout* is -1, *poll()* shall block until a  
28098 requested event occurs or until the call is interrupted.
- 28099 Implementations may place limitations on the granularity of timeout intervals. If the requested  
28100 timeout interval requires a finer granularity than the implementation supports, the actual  
28101 timeout interval shall be rounded up to the next supported value.
- 28102 The *poll()* function is not affected by the O\_NONBLOCK flag.
- 28103 XSR The *poll()* function supports regular files, terminal and pseudo-terminal devices, STREAMS-  
28104 based files, FIFOs, pipes, and sockets. The behavior of *poll()* on elements of *fds* that refer to other  
28105 types of file is unspecified.
- 28106 Regular files always poll TRUE for reading and writing.
- 28107 The *poll()* function supports sockets.
- 28108 A file descriptor for a socket that is listening for connections shall indicate that it is ready for  
28109 reading, once connections are available. A file descriptor for a socket that is connecting  
28110 asynchronously shall indicate that it is ready for writing, once a connection has been established.
- 28111 **RETURN VALUE**
- 28112 Upon successful completion, *poll()* shall return a non-negative value. A positive value indicates  
28113 the total number of file descriptors that have been selected (that is, file descriptors for which the  
28114 *revents* member is non-zero). A value of 0 indicates that the call timed out and no file descriptors  
28115 have been selected. Upon failure, *poll()* shall return -1 and set *errno* to indicate the error.
- 28116 **ERRORS**
- 28117 The *poll()* function shall fail if:
- |           |          |                                                                                      |
|-----------|----------|--------------------------------------------------------------------------------------|
| 28118     | [EAGAIN] | The allocation of internal data structures failed but a subsequent request may       |
| 28119     |          | succeed.                                                                             |
| 28120     | [EINTR]  | A signal was caught during <i>poll()</i> .                                           |
| 28121 XSR | [EINVAL] | The <i>nfds</i> argument is greater than {OPEN_MAX}, or one of the <i>fd</i> members |
| 28122     |          | refers to a STREAM or multiplexer that is linked (directly or indirectly)            |
| 28123     |          | downstream from a multiplexer.                                                       |
- 28124 **EXAMPLES**
- 28125 **Checking for Events on a Stream**
- 28126 The following example opens a pair of STREAMS devices and then waits for either one to  
28127 become writable. This example proceeds as follows:
- 28128 1. Sets the *timeout* parameter to 500 milliseconds.
  - 28129 2. Opens the STREAMS devices */dev/dev0* and */dev/dev1*, and then polls them, specifying  
28130 POLLOUT and POLLWRBAND as the events of interest.
- 28131 The STREAMS device names */dev/dev0* and */dev/dev1* are only examples of how  
28132 STREAMS devices can be named; STREAMS naming conventions may vary among  
28133 systems conforming to the IEEE Std. 1003.1-200x.
- 28134 3. Uses the *ret* variable to determine whether an event has occurred on either of the two  
28135 STREAMS. The *poll()* function is given 500 milliseconds to wait for an event to occur (if it

- 28136 has not occurred prior to the *poll()* call).
- 28137 4. Checks the returned value of *ret*. If a positive value is returned, one of the following can  
28138 be done:
- 28139 a. Priority data can be written to the open STREAM on priority bands greater than 0,  
28140 because the POLLWRBAND event occurred on the open STREAM (*fds*[0] or *fds*[1]).
  - 28141 b. Data can be written to the open STREAM on priority-band 0, because the POLLOUT  
28142 event occurred on the open STREAM (*fds*[0] or *fds*[1]).
- 28143 5. If the returned value is not a positive value, permission to write data to the open STREAM  
28144 (on any priority band) is denied.
- 28145 6. If the POLLHUP event occurs on the open STREAM (*fds*[0] or *fds*[1]), the device on the  
28146 open STREAM has disconnected.

```

28147 #include <stropts.h>
28148 #include <poll.h>
28149 ...
28150 struct pollfd fds[2];
28151 int timeout_msecs = 500;
28152 int ret;
28153 int i;

28154 /* Open STREAMS device. */
28155 fds[0].fd = open("/dev/dev0", ...);
28156 fds[1].fd = open("/dev/dev1", ...);
28157 fds[0].events = POLLOUT | POLLWRBAND;
28158 fds[1].events = POLLOUT | POLLWRBAND;

28159 ret = poll(fds, 2, timeout_msecs);

28160 if (ret > 0) {
28161 /* An event on one of the fds has occurred. */
28162 for (i=0; i<2; i++) {
28163 if (fds[i].revents & POLLWRBAND) {
28164 /* Priority data may be written on device number i. */
28165 ...
28166 }
28167 if (fds[i].revents & POLLOUT) {
28168 /* Data may be written on device number i. */
28169 ...
28170 }
28171 if (fds[i].revents & POLLHUP) {
28172 /* A hangup has occurred on device number i. */
28173 ...
28174 }
28175 }
28176 }

```

#### 28177 APPLICATION USAGE

28178 None.

**28179 RATIONALE**

28180 None.

**28181 FUTURE DIRECTIONS**

28182 None.

**28183 SEE ALSO**

28184 *getmsg()*, *putmsg()*, *read()*, *select()*, *write()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |

28185 **<poll.h>**, **<stropts.h>**, Section 2.6 (on page 539)

**28186 CHANGE HISTORY**

28187 First released in Issue 4, Version 2.

**28188 Issue 5**

28189 Moved from X/OPEN UNIX extension to BASE.

28190 The description of POLLWRBAND is updated.

**28191 Issue 6**

28192 Text referring to sockets is added to the DESCRIPTION.

28193 Text relating to the XSI STREAMS Option Group is marked.

28194 **NAME**

28195 popen — initiate pipe streams to or from a process

28196 **SYNOPSIS**

28197 #include &lt;stdio.h&gt;

28198 FILE \*popen(const char \*command, const char \*mode);

28199 **DESCRIPTION**28200 The *popen()* function shall execute the command specified by the string *command*. It creates a  
28201 pipe between the calling program and the executed command, and returns a pointer to a stream  
28202 that can be used to either read from or write to the pipe.28203 The environment of the executed command shall be as if a child process were created within the  
28204 *popen()* call using the *fork()* function, and the child used *execl()* to invoke a command line  
28205 interpreter.28206 If the implementation supports the Shell and Utilities volume of IEEE Std. 1003.1-200x, the  
28207 environment of the executed command is as if a child process were created within the *popen()*  
28208 call using *fork()*, and the child invoked the *sh* utility using the call:28209 `execl(shell_path, "sh", "-c", command, (char *)0);`28210 where *shell\_path* is an unspecified path name for the *sh* utility.28211 The *popen()* function ensures that any streams from previous *popen()* calls that remain open in  
28212 the parent process are closed in the new child process.28213 The *mode* argument to *popen()* is a string that specifies I/O mode:

- 28214 1. If *mode* is *r*, when the child process is started, its file descriptor `STDOUT_FILENO` shall be  
28215 the writable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,  
28216 where *stream* is the stream pointer returned by *popen()*, shall be the readable end of the  
28217 pipe.
- 28218 2. If *mode* is *w*, when the child process is started its file descriptor `STDIN_FILENO` shall be  
28219 the readable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,  
28220 where *stream* is the stream pointer returned by *popen()*, shall be the writable end of the  
28221 pipe.
- 28222 3. If *mode* is any other value, the result is undefined.

28223 After *popen()*, both the parent and the child process shall be capable of executing independently  
28224 before either terminates.

28225 Pipe streams are byte-oriented.

28226 **RETURN VALUE**28227 Upon successful completion, *popen()* shall return a pointer to an open stream that can be used to  
28228 read or write to the pipe. Otherwise, it shall return a null pointer and may set *errno* to indicate  
28229 the error.28230 **ERRORS**28231 The *popen()* function may fail if:28232 [EMFILE] {FOPEN\_MAX} or {STREAM\_MAX} streams are currently open in the calling  
28233 process.28234 [EINVAL] The *mode* argument is invalid.28235 The *popen()* function may also set *errno* values as described by *fork()* or *pipe()*.

28236 **EXAMPLES**

28237 None.

28238 **APPLICATION USAGE**28239 Because open files are shared, a mode *r* command can be used as an input filter and a mode *w*  
28240 command as an output filter.28241 Buffered reading before opening an input filter may leave the standard input of that filter  
28242 mispositioned. Similar problems with an output filter may be prevented by careful buffer  
28243 flushing; for example, with *fflush()*.28244 A stream opened by *popen()* should be closed by *pclose()*.28245 The behavior of *popen()* is specified for values of *mode* of *r* and *w*. Other modes such as *rb* and  
28246 *wb* might be supported by specific implementations, but these would not be portable features.  
28247 Note that historical implementations of *popen()* only check to see if the first character of *mode* is  
28248 *r*. Thus, a *mode* of *robert the robot* would be treated as *mode r*, and a *mode* of *anything else* would be  
28249 treated as *mode w*.28250 If the application calls *waitpid()* or *waitid()* with a *pid* argument greater than 0, and it still has a  
28251 stream that was called with *popen()* open, it must ensure that *pid* does not refer to the process  
28252 started by *popen()*.28253 To determine whether or not the environment specified in the Shell and Utilities volume of  
28254 IEEE Std. 1003.1-200x is present, use the function call:28255 `sysconf(_SC_2_VERSION)`28256 (See *sysconf()*).28257 **RATIONALE**28258 The *popen()* function should not be used by programs that have set user (or group) ID privileges.  
28259 The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used instead.  
28260 This prevents any unforeseen manipulation of the environment of the user that could cause  
28261 execution of commands not anticipated by the calling program.28262 If the original and *popen()*ed processes both intend to read or write or read and write a common  
28263 file, and either will be using FILE-type C functions (*fread()*, *fwrite()*, and so on), the rules for  
28264 sharing file handles must be observed (see Section 2.5.1 (on page 535)).28265 Because open files are shared, a mode *r* argument can be used as an input filter and a mode *w*  
28266 argument as an output filter.28267 The behavior of *popen()* is specified for modes of *r* and *w*. Other modes such as *rb* and *wb* might  
28268 be supported by specific implementations, but these would not be portable features. Note that  
28269 historical implementations of *popen()* only check to see if the first character of *mode* is '*r*'.  
28270 Thus, a *mode* of *robert the robot* would be treated as *mode r*, and a *mode* of *anything else* would be  
28271 treated as *mode w*.28272 If the application calls *waitpid()* with a *pid* argument greater than zero, and it still has a  
28273 *popen()*ed stream open, it must ensure that *pid* does not refer to the process started by *popen()*.28274 **FUTURE DIRECTIONS**

28275 None.

28276 **SEE ALSO**28277 *pclose()*, *pipe()*, *sysconf()*, *system()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
28278 `<stdio.h>`, the Shell and Utilities volume of IEEE Std. 1003.1-200x

28279 **CHANGE HISTORY**

28280 First released in Issue 1. Derived from Issue 1 of the SVID.

28281 **Issue 4**

28282 The APPLICATION USAGE section is extended. Only notes about buffer flushing are retained  
28283 from Issue 3.

28284 The following changes are incorporated for alignment with the ISO POSIX-2 standard:

- 28285 • The function is no longer marked as an extension.
- 28286 • The type of arguments *command* and *mode* are changed from **char\*** to **const char\***.
- 28287 • The DESCRIPTION is completely rewritten for alignment with the ISO POSIX-2 standard,  
28288 although it describes essentially the same functionality as Issue 3.
- 28289 • The *sh* utility defined in the Shell and Utilities volume of IEEE Std. 1003.1-200x is no longer  
28290 required in all circumstances.
- 28291 • The ERRORS section is added.

28292 **Issue 5**

28293 A statement is added to the DESCRIPTION indicating that pipe streams are byte-oriented.

28294 **Issue 6**

28295 The following new requirements on POSIX implementations derive from alignment with the  
28296 Single UNIX Specification:

- 28297 • The optional [EMFILE] error condition is added.

28298 The following changes were made to align with the IEEE P1003.1a draft standard:

- 28299 • The DESCRIPTION is adjusted to reflect the behavior on systems that do not support the  
28300 Shell option.

28301 **NAME**

28302        **posix\_fadvise** — file advisory information

28303 **SYNOPSIS**

28304 ADV     #include <fcntl.h>

28305        int posix\_fadvise(int *fd*, off\_t *offset*, size\_t *len*, int *advice*);

28306

28307 **DESCRIPTION**

28308        The *posix\_fadvise()* function provides advice to the implementation on the expected behavior of  
 28309        the application with respect to the data in the file associated with the open file descriptor, *fd*,  
 28310        starting at *offset* and continuing for *len* bytes. The specified range need not currently exist in the  
 28311        file. If *len* is zero, all data following *offset* is specified. The implementation may use this  
 28312        information to optimize handling of the specified data. The *posix\_fadvise()* function has no effect  
 28313        on the semantics of other operations on the specified data, although it may affect the  
 28314        performance of other operations.

28315        The advice to be applied to the data is specified by the *advice* parameter and may be one of the  
 28316        following values:

28317        **POSIX\_FADV\_NORMAL**

28318            Specifies that the application has no advice to give on its behavior with respect to the  
 28319            specified data. It is the default characteristic if no advice is given for an open file.

28320        **POSIX\_FADV\_SEQUENTIAL**

28321            Specifies that the application expects to access the specified data sequentially from lower  
 28322            offsets to higher offsets.

28323        **POSIX\_FADV\_RANDOM**

28324            Specifies that the application expects to access the specified data in a random order.

28325        **POSIX\_FADV\_WILLNEED**

28326            Specifies that the application expects to access the specified data in the near future.

28327        **POSIX\_FADV\_DONTNEED**

28328            Specifies that the application expects that it will not access the specified data in the near  
 28329            future.

28330        **POSIX\_FADV\_NOREUSE**

28331            Specifies that the application expects to access the specified data once and then not reuse it  
 28332            thereafter.

28333        These values shall be defined in <fcntl.h>.

28334 **RETURN VALUE**

28335        Upon successful completion, *posix\_fadvise()* shall return zero; otherwise, an error number shall  
 28336        be returned to indicate the error.

28337 **ERRORS**

28338        The *posix\_fadvise()* function shall fail if:

28339        [EBADF]        The *fd* argument is not a valid file descriptor.

28340        [EINVAL]       The value of *advice* is invalid.

28341        [ESPIPE]       The *fd* argument is associated with a pipe or FIFO.



28342 **EXAMPLES**

28343           None.

28344 **APPLICATION USAGE**

28345           The *posix\_fadvise()* function is part of the Advisory Information option and need not be provided |  
28346           on all implementations.

28347 **RATIONALE**

28348           None.

28349 **FUTURE DIRECTIONS**

28350           None.

28351 **SEE ALSO**28352           *posix\_madvise()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <fcntl.h> |28353 **CHANGE HISTORY**

28354           First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

28355           In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

28356 **NAME**

28357 posix\_fallocate — file space control

28358 **SYNOPSIS**

28359 ADV #include <fcntl.h>

28360 int posix\_fallocate(int fd, off\_t offset, size\_t len);

28361

28362 **DESCRIPTION**

28363 The *posix\_fallocate()* function ensures that any required storage for regular file data starting at  
 28364 *offset* and continuing for *len* bytes is allocated on the file system storage media. If *posix\_fallocate()*  
 28365 returns successfully, subsequent writes to the specified file data shall not fail due to the lack of  
 28366 free space on the file system storage media.

28367 If the *offset+len* is beyond the current file size, then *posix\_fallocate()* shall adjust the file size to  
 28368 *offset+len*. Otherwise, the file size shall not be changed.

28369 It is implementation-defined whether a previous *posix\_fadvise()* call influences allocation  
 28370 strategy.

28371 Space allocated via *posix\_fallocate()* shall be freed by a successful call to *creat()* or *open()* that  
 28372 truncates the size of the file. Space allocated via *posix\_fallocate()* may be freed by a successful call  
 28373 to *ftruncate()* that reduces the file size to a size smaller than *offset+len*.

28374 **RETURN VALUE**

28375 Upon successful completion, *posix\_fallocate()* shall return zero; otherwise, an error number shall  
 28376 be returned to indicate the error.

28377 **ERRORS**

28378 The *posix\_fallocate()* function shall fail if:

- 28379 [EBADF] The *fd* argument is not a valid file descriptor.
- 28380 [EBADF] The *fd* argument references a file that was opened without write permission.
- 28381 [EFBIG] The value of *offset+len* is greater than the maximum file size.
- 28382 [EINTR] A signal was caught during execution.
- 28383 [EINVAL] The *len* argument was zero or the *offset* argument was less than zero.
- 28384 [EIO] An I/O error occurred while reading from or writing to a file system.
- 28385 [ENODEV] The *fd* argument does not refer to a regular file.
- 28386 [ENOSPC] There is insufficient free space remaining on the file system storage media.
- 28387 [ESPIPE] The *fd* argument is associated with a pipe or FIFO.

28388 **EXAMPLES**

28389 None.

28390 **APPLICATION USAGE**

28391 The *posix\_fallocate()* function is part of the Advisory Information option and need not be  
 28392 provided on all implementations.

28393 **RATIONALE**

28394 None.

28395 **FUTURE DIRECTIONS**

28396       None.

28397 **SEE ALSO**28398       *creat()*, *ftruncate()*, *open()*, *unlink()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |  
28399       <fcntl.h>28400 **CHANGE HISTORY**

28401       First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

28402       In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

28403 **NAME**

28404 posix\_madvise — memory advisory information and alignment control

28405 **SYNOPSIS**

28406 ADV #include <sys/mman.h>

28407 int posix\_madvise(void \*addr, size\_t len, int advice);

28408

28409 **DESCRIPTION**

28410 MF|SHM The *posix\_madvise()* function need only be supported if either the Memory Mapped Files or the  
28411 Shared Memory Objects options are supported.

28412 The *posix\_madvise()* function provides advice to the implementation on the expected behavior of  
28413 the application with respect to the data in the memory starting at address *addr*, and continuing  
28414 for *len* bytes. The implementation may use this information to optimize handling of the specified  
28415 data. The *posix\_madvise()* function has no effect on the semantics of access to memory in the  
28416 specified range, although it may affect the performance of access.

28417 The implementation may require that *addr* be a multiple of the page size, which is the value  
28418 returned by *sysconf()* when the name value *\_SC\_PAGESIZE* is used.

28419 The advice to be applied to the memory range is specified by the *advice* parameter and may be  
28420 one of the following values:

28421 POSIX\_MADV\_NORMAL

28422 Specifies that the application has no advice to give on its behavior with respect to the  
28423 specified range. It is the default characteristic if no advice is given for a range of memory.

28424 POSIX\_MADV\_SEQUENTIAL

28425 Specifies that the application expects to access the specified range sequentially from lower  
28426 addresses to higher addresses.

28427 POSIX\_MADV\_RANDOM

28428 Specifies that the application expects to access the specified range in a random order.

28429 POSIX\_MADV\_WILLNEED

28430 Specifies that the application expects to access the specified range in the near future.

28431 POSIX\_MADV\_DONTNEED

28432 Specifies that the application expects that it will not access the specified range in the near  
28433 future.

28434 These values shall be defined in <sys/mman.h>.

28435 **RETURN VALUE**

28436 Upon successful completion, *posix\_madvise()* shall return zero; otherwise, an error number shall  
28437 be returned to indicate the error.

28438 **ERRORS**

28439 The *posix\_madvise()* function shall fail if:

28440 [EINVAL] The value of *advice* is invalid.

28441 [ENOMEM] Addresses in the range starting at *addr* and continuing for *len* bytes are partly  
28442 or completely outside the range allowed for the address space of the calling  
28443 process.

28444 The *posix\_madvise()* function may fail if:

28445 [EINVAL] The value of *addr* is not a multiple of the value returned by *sysconf()* when the  
28446 name value *\_SC\_PAGESIZE* is used.

28447 [EINVAL] The value of *len* is zero.

28448 **EXAMPLES**

28449 None.

28450 **APPLICATION USAGE**

28451 The *posix\_madvise()* function is part of the Advisory Information option and need not be |  
28452 provided on all implementations.

28453 **RATIONALE**

28454 None.

28455 **FUTURE DIRECTIONS**

28456 None.

28457 **SEE ALSO**

28458 *mmap()*, *posix\_fadvise()*, *sysconf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |  
28459 <**sys/mmap.h**>

28460 **CHANGE HISTORY**

28461 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

28462 IEEE PASC Interpretation 1003.1 #102 is included. |

28463 **NAME**

28464 posix\_mem\_offset — find offset and length of a mapped typed memory block

28465 **SYNOPSIS**

28466 TYM #include <sys/mman.h>

```
28467 int posix_mem_offset(const void *restrict addr, size_t len,
28468 off_t *restrict off, size_t *restrict contig_len,
28469 int *restrict fildes);
28470
```

28471 **DESCRIPTION**

28472 The *posix\_mem\_offset()* function returns in the variable pointed to by *off* a value that identifies the offset (or location), within a memory object, of the memory block currently mapped at *addr*.  
 28473 The function shall return in the variable pointed to by *fildes*, the descriptor used (via *mmap()*) to  
 28474 establish the mapping which contains *addr*. If that descriptor was closed since the mapping was  
 28475 established, the returned value of *fildes* shall be  $-1$ . The *len* argument specifies the length of the  
 28476 block of the memory object the user wishes the offset for; upon return, the value pointed to by  
 28477 *contig\_len* shall equal either *len*, or the length of the largest contiguous block of the memory  
 28478 object that is currently mapped to the calling process starting at *addr*, whichever is smaller.  
 28479

28480 If the memory object mapped at *addr* is a typed memory object, then if the *off* and *contig\_len*  
 28481 values obtained by calling *posix\_mem\_offset()* are used in a call to *mmap()* with a file descriptor  
 28482 that refers to the same memory pool as *fildes* (either through the same port or through a different  
 28483 port), and that was opened with neither the POSIX\_TYPED\_MEM\_ALLOCATE nor the  
 28484 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG flag, the typed memory area that is mapped shall  
 28485 be exactly the same area that was mapped at *addr* in the address space of the process that called  
 28486 *posix\_mem\_offset()*.

28487 If the memory object specified by *fildes* is not a typed memory object, then the behavior of this  
 28488 function is implementation-defined.

28489 **RETURN VALUE**

28490 Upon successful completion, the *posix\_mem\_offset()* function shall return zero; otherwise, the  
 28491 corresponding error status value shall be returned.

28492 **ERRORS**

28493 The *posix\_mem\_offset()* function shall fail if:

28494 [EACCES] The process has not mapped a memory object supported by this function at  
 28495 the given address *addr*.

28496 This function shall not return an error code of [EINTR].

28497 **EXAMPLES**

28498 None.

28499 **APPLICATION USAGE**

28500 None.

28501 **RATIONALE**

28502 None.

28503 **FUTURE DIRECTIONS**

28504 None.

28505 **SEE ALSO**

28506 *mmap()*, *posix\_typed\_mem\_open()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |  
28507 `<sys/mman.h>`

28508 **CHANGE HISTORY**

28509 First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.

28510 **NAME**

28511 posix\_memalign — aligned memory allocation

28512 **SYNOPSIS**

28513 ADV #include <stdlib.h>

28514 int posix\_memalign(void \*\*memptr, size\_t alignment, size\_t size);

28515

28516 **DESCRIPTION**

28517 The *posix\_memalign()* function allocates *size* bytes aligned on a boundary specified by *alignment*,  
 28518 and returns a pointer to the allocated memory in *memptr*. The value of *alignment* shall be a  
 28519 multiple of *sizeof(void\*)*, that is also a power of two. Upon successful completion, the value  
 28520 pointed to by *memptr* shall be a multiple of *alignment*.

28521 CX The *free()* function shall deallocate memory that has previously been allocated by  
 28522 *posix\_memalign()*.

28523 **RETURN VALUE**

28524 Upon successful completion, *posix\_memalign()* shall return zero; otherwise, an error number  
 28525 shall be returned to indicate the error.

28526 **ERRORS**

28527 The *posix\_memalign()* function shall fail if:

28528 [EINVAL] The value of the alignment parameter is not a power of two multiple of  
 28529 *sizeof(void\*)*.

28530 [ENOMEM] There is insufficient memory available with the requested alignment.

28531 **EXAMPLES**

28532 None.

28533 **APPLICATION USAGE**

28534 The *posix\_memalign()* function is part of the Advisory Information option and need not be  
 28535 provided on all implementations.

28536 **RATIONALE**

28537 None.

28538 **FUTURE DIRECTIONS**

28539 None.

28540 **SEE ALSO**

28541 *free()*, *malloc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>

28542 **CHANGE HISTORY**

28543 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

28544 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.



28545 **NAME**28546 posix\_spawn, posix\_spawnnp — spawn a process (**REALTIME**)28547 **SYNOPSIS**28548 SPN 

```
#include <spawn.h>
```

```
28549 int posix_spawn(pid_t *restrict pid, const char *restrict path,
28550 const posix_spawn_file_actions_t *file_actions,
28551 const posix_spawnattr_t *restrict attrp,
28552 char *restrict const argv[restrict],
28553 char *restrict const envp[restrict]);
28554 int posix_spawnnp(pid_t *restrict pid, const char *restrict file,
28555 const posix_spawn_file_actions_t *file_actions,
28556 const posix_spawnattr_t *restrict attrp, char *const argv[restrict],
28557 char * const envp[restrict]);
28558
```

28559 **DESCRIPTION**

28560 The *posix\_spawn()* and *posix\_spawnnp()* functions shall create a new process (child process) from  
 28561 the specified process image. The new process image is constructed from a regular executable file  
 28562 called the new process image file.

28563 When a C program is executed as the result of this call, it shall be entered as a C language  
 28564 function call as follows:

```
28565 int main(int argc, char *argv[]);
```

28566 where *argc* is the argument count and *argv* is an array of character pointers to the arguments  
 28567 themselves. In addition, the following variable:

```
28568 extern char **environ;
```

28569 is initialized as a pointer to an array of character pointers to the environment strings.

28570 The argument *argv* is an array of character pointers to null-terminated strings. The last member  
 28571 of this array shall be a null pointer and is not counted in *argc*. These strings constitute the  
 28572 argument list available to the new process image. The value in *argv[0]* should point to a file  
 28573 name that is associated with the process image being started by the *posix\_spawn()* or  
 28574 *posix\_spawnnp()* function.

28575 The argument *envp* is an array of character pointers to null-terminated strings. These strings  
 28576 constitute the environment for the new process image. The environment array is terminated by a  
 28577 null pointer.

28578 The number of bytes available for the child process' combined argument and environment lists  
 28579 is {ARG\_MAX}. The implementation shall specify in the system documentation (see the Base  
 28580 Definitions volume of IEEE Std. 1003.1-200x, Chapter 2, Conformance) whether any list  
 28581 overhead, such as length words, null terminators, pointers, or alignment bytes, is included in  
 28582 this total.

28583 The *path* argument to *posix\_spawn()* is a path name that identifies the new process image file to  
 28584 execute.

28585 The *file* parameter to *posix\_spawnnp()* shall be used to construct a path name that identifies the  
 28586 new process image file. If the *file* parameter contains a slash character, the *file* parameter shall be  
 28587 used as the path name for the new process image file. Otherwise, the path prefix for this file shall  
 28588 be obtained by a search of the directories passed as the environment variable *PATH* (see the Base  
 28589 Definitions volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables). If this  
 28590 environment variable is not defined, the results of the search are implementation-defined.

28591 If *file\_actions* is a null pointer, then file descriptors open in the calling process shall remain open  
 28592 in the child process, except for those whose close-on-exec flag FD\_CLOEXEC is set (see *fcntl()*).  
 28593 For those file descriptors that remain open, all attributes of the corresponding open file  
 28594 descriptions, including file locks (see *fcntl()*), shall remain unchanged.

28595 If *file\_actions* is not NULL, then the file descriptors open in the child process shall be those open  
 28596 in the calling process as modified by the spawn file actions object pointed to by *file\_actions* and  
 28597 the FD\_CLOEXEC flag of each remaining open file descriptor after the spawn file actions have  
 28598 been processed. The effective order of processing the spawn file actions shall be:

- 28599 1. The set of open file descriptors for the child process shall initially be the same set as is  
 28600 open for the calling process. All attributes of the corresponding open file descriptions,  
 28601 including file locks (see *fcntl()*), shall remain unchanged.
- 28602 2. The signal mask, signal default actions, and the effective user and group IDs for the child  
 28603 process shall be changed as specified in the attributes object referenced by *attrp*.
- 28604 3. The file actions specified by the spawn file actions object shall be performed in the order in  
 28605 which they were added to the spawn file actions object.
- 28606 4. Any file descriptor that has its FD\_CLOEXEC flag set (see *fcntl()*) shall be closed.

28607 The **posix\_spawnattr\_t** spawn attributes object type is defined in **<spawn.h>**. It shall contain at  
 28608 least the attributes defined below.

28609 If the POSIX\_SPAWN\_SETPGROUP flag is set in the *spawn\_flags* attribute of the object  
 28610 referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is non-zero, then the  
 28611 child's process group shall be as specified in the *spawn-pgroup* attribute of the object referenced  
 28612 by *attrp*.

28613 As a special case, if the POSIX\_SPAWN\_SETPGROUP flag is set in the *spawn\_flags* attribute of  
 28614 the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is set to zero,  
 28615 then the child shall be in a new process group with a process group ID equal to its process ID.

28616 If the POSIX\_SPAWN\_SETPGROUP flag is not set in the *spawn\_flags* attribute of the object  
 28617 referenced by *attrp*, the new child process shall inherit the parent's process group.

28618 PS If the POSIX\_SPAWN\_SETSCHEDPARAM flag is set in the *spawn\_flags* attribute of the object  
 28619 referenced by *attrp*, but POSIX\_SPAWN\_SETSCHEDULER is not set, the new process image  
 28620 shall initially have the scheduling policy of the calling process with the scheduling parameters  
 28621 specified in the *spawn-schedparam* attribute of the object referenced by *attrp*.

28622 If the POSIX\_SPAWN\_SETSCHEDULER flag is set in *spawn\_flags* attribute of the object  
 28623 referenced by *attrp* (regardless of the setting of the POSIX\_SPAWN\_SETSCHEDPARAM flag),  
 28624 the new process image shall initially have the scheduling policy specified in the *spawn-*  
 28625 *schedpolicy* attribute of the object referenced by *attrp* and the scheduling parameters specified in  
 28626 the *spawn-schedparam* attribute of the same object.

28627 The POSIX\_SPAWN\_RESETEUIDS flag in the *spawn\_flags* attribute of the object referenced by *attrp*  
 28628 governs the effective user ID of the child process. If this flag is not set, the child process inherits  
 28629 the parent process' effective user ID. If this flag is set, the child process' effective user ID is reset  
 28630 to the parent's real user ID. In either case, if the set-user-ID mode bit of the new process image  
 28631 file is set, the effective user ID of the child process will become that file's owner ID before the  
 28632 new process image begins execution.

28633 The POSIX\_SPAWN\_RESETEGID flag in the *spawn\_flags* attribute of the object referenced by *attrp*  
 28634 also governs the effective group ID of the child process. If this flag is not set, the child process  
 28635 inherits the parent process' effective group ID. If this flag is set, the child process' effective group  
 28636 ID is reset to the parent's real group ID. In either case, if the set-group-ID mode bit of the new

- 28637 process image file is set, the effective group ID of the child process will become that file's group  
28638 ID before the new process image begins execution.
- 28639 If the POSIX\_SPAWN\_SETSIGMASK flag is set in the *spawn-flags* attribute of the object  
28640 referenced by *attrp*, the child process shall initially have the signal mask specified in the *spawn-*  
28641 *sigmask* attribute of the object referenced by *attrp*.
- 28642 If the POSIX\_SPAWN\_SETSIGDEF flag is set in the *spawn-flags* attribute of the object referenced  
28643 by *attrp*, the signals specified in the *spawn-sigdefault* attribute of the same object shall be set to  
28644 their default actions in the child process. Signals set to the default action in the parent process  
28645 shall be set to the default action in the child process.
- 28646 Signals set to be caught by the calling process shall be set to the default action in the child  
28647 process.
- 28648 Signals set to be ignored by the calling process image shall be set to be ignored by the child  
28649 process, unless otherwise specified by the POSIX\_SPAWN\_SETSIGDEF flag being set in the  
28650 *spawn-flags* attribute of the object referenced by *attrp* and the signals being indicated in the  
28651 *spawn-sigdefault* attribute of the object referenced by *attrp*.
- 28652 If the value of the *attrp* pointer is NULL, then the default values are used.
- 28653 All process attributes, other than those influenced by the attributes set in the object referenced  
28654 by *attrp* as specified above or by the file descriptor manipulations specified in *file\_actions*, shall  
28655 appear in the new process image as though *fork()* had been called to create a child process and  
28656 then a member of the *exec* family of functions had been called by the child process to execute the  
28657 new process image.
- 28658 THR It is implementation-defined whether the fork handlers are run when *posix\_spawn()* or  
28659 *posix\_spawnp()* is called.
- 28660 **RETURN VALUE**
- 28661 Upon successful completion, *posix\_spawn()* and *posix\_spawnp()* shall return the process ID of the  
28662 child process to the parent process, in the variable pointed to by a non-NULL *pid* argument, and  
28663 shall return zero as the function return value. Otherwise, no child process shall be created, the  
28664 value stored into the variable pointed to by a non-NULL *pid* is unspecified, and an error number  
28665 shall be returned as the function return value to indicate the error. If the *pid* argument is a null  
28666 pointer, the process ID of the child is not returned to the caller.
- 28667 **ERRORS**
- 28668 The *posix\_spawn()* and *posix\_spawnp()* functions may fail if:
- 28669 [EINVAL] The value specified by *file\_actions* or *attrp* is invalid.
- 28670 If this error occurs after the calling process successfully returns from the *posix\_spawn()* or  
28671 *posix\_spawnp()* function, the child process may exit with exit status 127.
- 28672 If *posix\_spawn()* or *posix\_spawnp()* fail for any of the reasons that would cause *fork()* or one of  
28673 the *exec* family of functions to fail, an error value shall be returned as described by *fork()* and  
28674 *exec*, respectively (or, if the error occurs after the calling process successfully returns, the child  
28675 process exits with exit status 127).
- 28676 If POSIX\_SPAWN\_SETPGROUP is set in the *spawn-flags* attribute of the object referenced by  
28677 *attrp*, and *posix\_spawn()* or *posix\_spawnp()* fails while changing the child's process group, an  
28678 error value shall be returned as described by *setpgid()* (or, if the error occurs after the calling  
28679 process successfully returns, the child process exits with exit status 127).
- 28680 PS If POSIX\_SPAWN\_SETSCHEDPARAM is set and POSIX\_SPAWN\_SETSCHEDULER is not set  
28681 in the *spawn-flags* attribute of the object referenced by *attrp*, then if *posix\_spawn()* or

28682 *posix\_spawnnp()* fails for any of the reasons that would cause *sched\_setparam()* to fail, an error  
 28683 value shall be returned as described by *sched\_setparam()* (or, if the error occurs after the calling  
 28684 process successfully returns, the child process exits with exit status 127).

28685 If POSIX\_SPAWN\_SETSCHEDULER is set in the *spawn-flags* attribute of the object referenced by  
 28686 *attrp*, and if *posix\_spawn()* or *posix\_spawnnp()* fails for any of the reasons that would cause  
 28687 *sched\_setscheduler()* to fail, an error value shall be returned as described by *sched\_setscheduler()*  
 28688 (or, if the error occurs after the calling process successfully returns, the child process exits with  
 28689 exit status 127)>

28690 If the *file\_actions* argument is not NULL, and specifies any *close*, *dup2*, or *open* actions to be  
 28691 performed, and if *posix\_spawn()* or *posix\_spawnnp()* fails for any of the reasons that would cause  
 28692 *close()*, *dup2()*, or *open()* to fail, an error value shall be returned as described by *close()*, *dup2()*,  
 28693 and *open()*, respectively (or, if the error occurs after the calling process successfully returns, the  
 28694 child process exits with exit status 127). An open file action may, by itself, result in any of the  
 28695 errors described by *close()* or *dup2()*, in addition to those described by *open()*.

28696 **EXAMPLES**

28697 None.

28698 **APPLICATION USAGE**

28699 These functions are part of the Spawn option and need not be provided on all implementations.

28700 **RATIONALE**

28701 The POSIX *fork()* function is difficult or impossible to implement without swapping or dynamic  
 28702 address translation for the following reasons:

- 28703 • Swapping is generally too slow for a realtime environment.
- 28704 • Dynamic address translation is not available everywhere POSIX might be useful.
- 28705 • Processes are too useful to simply option out of POSIX whenever it must run without  
 28706 address translation or other MMU services,

28707 POSIX needs process creation and file execution primitives that can be efficiently implemented  
 28708 without address translation or other MMU services.

28709 This function shall be called *posix\_spawn()*. A closely related function, *posix\_spawnnp()*, is  
 28710 included for completeness.

28711 The *posix\_spawn()* function is implementable as a library routine, but both *posix\_spawn()* and  
 28712 *posix\_spawnnp()* are designed as kernel operations. Also, although they may be an efficient  
 28713 replacement for many *fork()/exec* pairs, their goal is to provide useful process creation  
 28714 primitives for systems that have difficulty with *fork()*, not to provide drop-in replacements for  
 28715 *fork()/exec*.

28716 This view of the role of *posix\_spawn()* and *posix\_spawnnp()* influenced the design of their API. It  
 28717 does not attempt to provide the full functionality of *fork()/exec* in which arbitrary user-specified  
 28718 operations of any sort are permitted between the creation of the child process and the execution  
 28719 of the new process image; any attempt to reach that level would need to provide a programming  
 28720 language as parameters. Instead, *posix\_spawn()* and *posix\_spawnnp()* are process creation  
 28721 primitives like the *Start\_Process* and *Start\_Process\_Search* Ada language bindings package  
 28722 *POSIX\_Process\_Primitives* and also like those in many operating systems that are not UNIX  
 28723 systems, but with some POSIX-specific additions.

28724 To achieve its coverage goals, *posix\_spawn()* and *posix\_spawnnp()* have control of six types of  
 28725 inheritance: file descriptors, process group ID, user and group ID, signal mask, scheduling, and  
 28726 whether each signal ignored in the parent will remain ignored in the child, or be reset to its  
 28727 default action in the child.

28728 Control of file descriptors is required to allow an independently written child process image to  
 28729 access data streams opened by and even generated or read by the parent process without being  
 28730 specifically coded to know which parent files and file descriptors are to be used. Control of the  
 28731 process group ID is required to control how the child process' job control relates to that of the  
 28732 parent.

28733 Control of the signal mask and signal defaulting is sufficient to support the implementation of  
 28734 *system()*. Although support for *system()* is not explicitly one of the goals for *posix\_spawn()* and  
 28735 *posix\_spawnnp()*, it is covered under the "at least 50%" coverage goal.

28736 The intention is that the normal file descriptor inheritance across *fork()*, the subsequent effect of  
 28737 the specified spawn file actions, and the normal file descriptor inheritance across one of the *exec*  
 28738 family of functions should fully specify open file inheritance. The implementation need make no  
 28739 decisions regarding the set of open file descriptors when the child process image begins  
 28740 execution, those decisions having already been made by the caller and expressed as the set of  
 28741 open file descriptors and their *FD\_CLOEXEC* flags at the time of the call and the spawn file  
 28742 actions object specified in the call. We have been assured that in cases where the POSIX  
 28743 *Start\_Process* Ada primitives have been implemented in a library, this method of controlling file  
 28744 descriptor inheritance may be implemented very easily.

28745 We can identify several problems with *posix\_spawn()* and *posix\_spawnnp()*, but there does not  
 28746 appear to be a solution that introduces fewer problems. Environment modification for child  
 28747 process attributes not specifiable via the *attrp* or *file\_actions* arguments must be done in the  
 28748 parent process, and since the parent generally wants to save its context, it is more costly than  
 28749 similar functionality with *fork()/exec*. It is also complicated to modify the environment of a  
 28750 multi-threaded process temporarily, since all threads must agree when it is safe for the  
 28751 environment to be changed. However, this cost is only borne by those invocations of  
 28752 *posix\_spawn()* and *posix\_spawnnp()* that use the additional functionality. Since extensive  
 28753 modifications are not the usual case, and are particularly unlikely in time-critical code, keeping  
 28754 much of the environment control out of *posix\_spawn()* and *posix\_spawnnp()* is appropriate design.

28755 The *posix\_spawn()* and *posix\_spawnnp()* functions do not have all the power of *fork()/exec*. This is  
 28756 to be expected. The *fork()* function is a wonderfully powerful operation. We do not expect to  
 28757 duplicate its functionality in a simple, fast function with no special hardware requirements. It is  
 28758 worth noting that *posix\_spawn()* and *posix\_spawnnp()* are very similar to the process creation  
 28759 operations on many operating systems that are not UNIX systems.

## 28760 Requirements

28761 The requirements for *posix\_spawn()* and *posix\_spawnnp()* are:

- 28762 • They must be implementable without an MMU or unusual hardware.
- 28763 • They must be compatible with existing POSIX standards.

28764 Additional goals are:

- 28765 • They should be efficiently implementable.
- 28766 • They should be able to replace at least 50% of typical executions of *fork()*.
- 28767 • A system with *posix\_spawn()* and *posix\_spawnnp()* and without *fork()* should be useful, at least  
 28768 for realtime applications.
- 28769 • A system with *fork()* and the *exec* family should be able to implement *posix\_spawn()* and  
 28770 *posix\_spawnnp()* as library routines.

28771 **Two-Syntax**

28772 POSIX *exec* has several calling sequences with approximately the same functionality. These  
 28773 appear to be required for compatibility with existing practice. Since the existing practice for the  
 28774 *posix\_spawn\*()* functions is otherwise substantially unlike POSIX, we feel that simplicity  
 28775 outweighs compatibility. There are, therefore, only two names for the *posix\_spawn\*()* functions.

28776 The parameter list does not differ between *posix\_spawn()* and *posix\_spawnp()*; *posix\_spawnp()*  
 28777 interprets the second parameter more elaborately than *posix\_spawn()*.

28778 **Compatibility with POSIX.5 (Ada)**

28779 The *Start\_Process* and *Start\_Process\_Search* procedures from the *POSIX\_Process\_Primitives*  
 28780 package from the Ada language binding to POSIX.1 encapsulate *fork()* and *exec* functionality in a  
 28781 manner similar to that of *posix\_spawn()* and *posix\_spawnp()*. Originally, in keeping with our  
 28782 simplicity goal, the standard developers had limited the capabilities of *posix\_spawn()* and  
 28783 *posix\_spawnp()* to a subset of the capabilities of *Start\_Process* and *Start\_Process\_Search*; certain  
 28784 non-default capabilities were not supported. However, based on suggestions by the ballot group  
 28785 to improve file descriptor mapping or drop it, and on the advice of an Ada Language Bindings  
 28786 working group member, the standard developers decided that *posix\_spawn()* and *posix\_spawnp()*  
 28787 should be sufficiently powerful to implement *Start\_Process* and *Start\_Process\_Search*. The  
 28788 rationale is that if the Ada language binding to such a primitive had already been approved as  
 28789 an IEEE standard, there can be little justification for not approving the functionally-equivalent  
 28790 parts of a C binding. The only three capabilities provided by *posix\_spawn()* and *posix\_spawnp()*  
 28791 that are not provided by *Start\_Process* and *Start\_Process\_Search* are optionally specifying the  
 28792 child's process group ID, the set of signals to be reset to default signal handling in the child  
 28793 process, and the child's scheduling policy and parameters.

28794 For the Ada language binding for *Start\_Process* to be implemented with *posix\_spawn()*, that  
 28795 binding would need to explicitly pass an empty signal mask and the parent's environment to  
 28796 *posix\_spawn()* whenever the caller of *Start\_Process* allowed these arguments to default, since  
 28797 *posix\_spawn()* does not provide such defaults. The ability of *Start\_Process* to mask user-specified  
 28798 signals during its execution is functionally unique to the Ada language binding and must be  
 28799 dealt with in the binding separately from the call to *posix\_spawn()*.

28800 **Process Group**

28801 The process group inheritance field can be used to join the child process with an existing process  
 28802 group. By assigning a value of zero to the *spawn-pgroup* attribute of the object referenced by  
 28803 *attrp*, the *setpgid()* mechanism will place the child process in a new process group.

28804 **Threads**

28805 Without the *posix\_spawn()* and *posix\_spawnp()* functions, systems without address translation  
 28806 can still use threads to give an abstraction of concurrency. In many cases, thread creation  
 28807 suffices, but it is not always a good substitute. The *posix\_spawn()* and *posix\_spawnp()* functions  
 28808 are considerably "heavier" than thread creation. Processes have several important attributes that  
 28809 threads do not. Even without address translation, a process may have base-and-bound memory  
 28810 protection. Each process has a process environment including security attributes and file  
 28811 capabilities, and powerful scheduling attributes. Processes abstract the behavior of non-  
 28812 uniform-memory-architecture multi-processors better than threads, and they are more  
 28813 convenient to use for activities that are not closely linked.

28814 The *posix\_spawn()* and *posix\_spawnp()* functions may not bring support for multiple processes to  
 28815 every configuration. Process creation is not the only piece of operating system support required  
 28816 to support multiple processes. The total cost of support for multiple processes may be quite high

28817 in some circumstances. Existing practice shows that support for multiple processes is  
 28818 uncommon and threads are common among “tiny kernels”. There should, therefore, probably  
 28819 continue to be AEPs for operating systems with only one process.

### 28820 **Asynchronous Error Notification**

28821 A library implementation of *posix\_spawn()* or *posix\_spawnp()* may not be able to detect all  
 28822 possible errors before it forks the child process. IEEE Std. 1003.1-200x provides for an error  
 28823 indication returned from a child process which could not successfully complete the spawn  
 28824 operation via a special exit status which may be detected using the status value returned by  
 28825 *wait()* and *waitpid()*.

28826 The *stat\_val* interface and the macros used to interpret it are not well suited to the purpose of  
 28827 returning API errors, but they are the only path available to a library implementation. Thus, an  
 28828 implementation may cause the child process to exit with exit status 127 for any error detected  
 28829 during the spawn process after the *posix\_spawn()* or *posix\_spawnp()* function has successfully  
 28830 returned.

28831 The standard developers had proposed using two additional macros to interpret *stat\_val*. The  
 28832 first, WIFSPAWNFAIL, would have detected a status that indicated that the child exited because  
 28833 of an error detected during the *posix\_spawn()* or *posix\_spawnp()* operations rather than during  
 28834 actual execution of the child process image; the second, WSPAWNERRNO, would have  
 28835 extracted the error value if WIFSPAWNFAIL indicated a failure. Unfortunately, the ballot group  
 28836 strongly opposed this because it would make a library implementation of *posix\_spawn()* or  
 28837 *posix\_spawnp()* dependent on kernel modifications to *waitpid()* to be able to embed special  
 28838 information in *stat\_val* to indicate a spawn failure.

28839 The 8 bits of child process exit status that are guaranteed by IEEE Std. 1003.1-200x to be  
 28840 accessible to the waiting parent process are insufficient to disambiguate a spawn error from any  
 28841 other kind of error that may be returned by an arbitrary process image. No other bits of the exit  
 28842 status are required to be visible in *stat\_val*, so these macros could not be strictly implemented at  
 28843 the library level. Reserving an exit status of 127 for such spawn errors is consistent with the use  
 28844 of this value by *system()* and *popen()* to signal failures in these operations that occur after the  
 28845 function has returned but before a shell is able to execute. The exit status of 127 does not  
 28846 uniquely identify this class of error, nor does it provide any detailed information on the nature  
 28847 of the failure. Note that a kernel implementation of *posix\_spawn()* or *posix\_spawnp()* is permitted  
 28848 (and encouraged) to return any possible error as the function value, thus providing more  
 28849 detailed failure information to the parent process.

28850 Thus, no special macros are available to isolate asynchronous *posix\_spawn()* or *posix\_spawnp()*  
 28851 errors. Instead, errors detected by the *posix\_spawn()* or *posix\_spawnp()* operations in the context  
 28852 of the child process before the new process image executes are reported by setting the child’s  
 28853 exit status to 127. The calling process may use the WIFEXITED and WEXITSTATUS macros on  
 28854 the *stat\_val* stored by the *wait()* or *waitpid()* functions to detect spawn failures to the extent that  
 28855 other status values with which the child process image may exit (before the parent can  
 28856 conclusively determine that the child process image has begun execution) are distinct from exit  
 28857 status 127.

### 28858 **FUTURE DIRECTIONS**

28859 None.

### 28860 **SEE ALSO**

28861 *alarm()*, *chmod()*, *close()*, *dup()*, *exec*, *exit()*, *fcntl()*, *fork()*, *kill()*, *open()*,  
 28862 *posix\_spawn\_file\_actions\_addclose()*, *posix\_spawn\_file\_actions\_adddup2()*,  
 28863 *posix\_spawn\_file\_actions\_addopen()*, *posix\_spawn\_file\_actions\_destroy()*,  
 28864 *posix\_spawn\_file\_actions\_init()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*,

28865 *posix\_spawnattr\_getsigdefault()*, *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*,  
28866 *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*, *posix\_spawnattr\_getsigmask()*,  
28867 *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*, *posix\_spawnattr\_setpgroup()*,  
28868 *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
28869 *sched\_setparam()*, *sched\_setscheduler()*, *setpgid()*, *setuid()*, *stat()*, *times()*, *wait()*, the Base  
28870 Definitions volume of IEEE Std. 1003.1-200x, <spawn.h>

**28871 CHANGE HISTORY**

28872 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

28873 IEEE PASC Interpretation 1003.1 #103 is included, noting that the signal default actions are  
28874 changed as well as the signal mask in step 2.



28875 **NAME**

28876 posix\_spawn\_file\_actions\_addclose, posix\_spawn\_file\_actions\_addopen — add close or open  
 28877 action to spawn file actions object (**REALTIME**)

28878 **SYNOPSIS**

```
28879 SPN #include <spawn.h>
28880
28880 int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *
28881 file_actions, int fildes);
28882 int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict
28883 file_actions, int fildes, const char *restrict path,
28884 int oflag, mode_t mode);
28885
```

28886 **DESCRIPTION**

28887 A spawn file actions object is of type **posix\_spawn\_file\_actions\_t** (defined in **<spawn.h>**) and is  
 28888 used to specify a series of actions to be performed by a *posix\_spawn()* or *posix\_spawnp()*  
 28889 operation in order to arrive at the set of open file descriptors for the child process given the set of  
 28890 open file descriptors of the parent. IEEE Std. 1003.1-200x does not define comparison or  
 28891 assignment operators for the type **posix\_spawn\_file\_actions\_t**.

28892 A spawn file actions object, when passed to *posix\_spawn()* or *posix\_spawnp()*, shall specify how  
 28893 the set of open file descriptors in the calling process is transformed into a set of potentially open  
 28894 file descriptors for the spawned process. This transformation shall be as if the specified sequence  
 28895 of actions was performed exactly once, in the context of the spawned process (prior to execution  
 28896 of the new process image), in the order in which the actions were added to the object;  
 28897 additionally, when the new process image is executed, any file descriptor (from this new set)  
 28898 which has its FD\_CLOEXEC flag set will be closed (see *posix\_spawn()*).

28899 The *posix\_spawn\_file\_actions\_addclose()* function adds a *close* action to the object referenced by  
 28900 *file\_actions* that will cause the file descriptor *fildes* to be closed (as if *close(fildes)* had been called)  
 28901 when a new process is spawned using this file actions object.

28902 The *posix\_spawn\_file\_actions\_addopen()* function adds an *open* action to the object referenced by  
 28903 *file\_actions* that will cause the file named by *path* to be opened (as if *open(path, oflag, mode)* had  
 28904 been called, and the returned file descriptor, if not *fildes*, had been changed to *fildes*) when a new  
 28905 process is spawned using this file actions object. If *fildes* was already an open file descriptor, it  
 28906 shall be closed before the new file is opened.

28907 **Notes to Reviewers**

28908 *This section with side shading will not appear in the final copy. - Ed.*

28909 D3, XSH, ERN 448 says the description of the *posix\_spawn\_file\_actions\_addopen* function does  
 28910 not say whether the function has to make a copy of the path parameter or whether it can store  
 28911 the pointer and assume the application does not destroy the copy of the string. Add to the  
 28912 description: "The string pointed to by path can become invalid so the function has to make a  
 28913 copy."

28914 **RETURN VALUE**

28915 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 28916 be returned to indicate the error.

28917 **ERRORS**

28918 These functions shall fail if:

28919 [EBADF] The value specified by *fildes* is negative or greater than or equal to  
 28920 {OPEN\_MAX}.

28921 These functions may fail if:

28922 [EINVAL] The value specified by *file\_actions* is invalid.

28923 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

28924 It shall not be considered an error for the *files* argument passed to these functions to specify a  
 28925 file descriptor for which the specified operation could not be performed at the time of the call.  
 28926 Any such error will be detected when the associated file actions object is later used during a  
 28927 *posix\_spawn()* or *posix\_spawnnp()* operation.

28928 **EXAMPLES**

28929 None.

28930 **APPLICATION USAGE**

28931 These functions are part of the Spawn option and need not be provided on all implementations.

28932 **RATIONALE**

28933 A spawn file actions object may be initialized to contain an ordered sequence of *close()*, *dup2()*,  
 28934 and *open()* operations to be used by *posix\_spawn()* or *posix\_spawnnp()* to arrive at the set of open  
 28935 file descriptors inherited by the spawned process from the set of open file descriptors in the  
 28936 parent at the time of the *posix\_spawn()* or *posix\_spawnnp()* call. It had been suggested that the  
 28937 *close()* and *dup2()* operations alone are sufficient to rearrange file descriptors, and that files  
 28938 which need to be opened for use by the spawned process can be handled either by having the  
 28939 calling process open them before the *posix\_spawn()* or *posix\_spawnnp()* call (and close them after),  
 28940 or by passing file names to the spawned process (in *argv*) so that it may open them itself. The  
 28941 standard developers recommend that applications use one of these two methods when practical,  
 28942 since detailed error status on a failed open operation is always available to the application this  
 28943 way. However, the standard developers feel that allowing a spawn file actions object to specify  
 28944 open operations is still appropriate because:

- 28945 1. It is consistent with equivalent POSIX.5 (Ada) functionality.
- 28946 2. It supports the I/O redirection paradigm commonly employed by POSIX programs  
 28947 designed to be invoked from a shell. When such a program is the child process, it may not  
 28948 be designed to open files on its own.
- 28949 3. It allows file opens that might otherwise fail or violate file ownership/access rights if  
 28950 executed by the parent process.

28951 Regarding 2. above, note that the spawn open file action provides to *posix\_spawn()* and  
 28952 *posix\_spawnnp()* the same capability that the shell redirection operators provide to *system()*, only  
 28953 without the intervening execution of a shell; for example:

```
28954 system ("myprog <file1 3<file2");
```

28955 Regarding 3. above, note that if the calling process needs to open one or more files for access by  
 28956 the spawned process, but has insufficient spare file descriptors, then the open action is necessary  
 28957 to allow the *open()* to occur in the context of the child process after other file descriptors have  
 28958 been closed (that must remain open in the parent).

28959 Additionally, if a parent is executed from a file having a “set-user-id” mode bit set and the  
 28960 POSIX\_SPAWN\_RESETEUIDS flag is set in the spawn attributes, a file created within the parent  
 28961 process will (possibly incorrectly) have the parent’s effective user ID as its owner, whereas a file  
 28962 created via an *open()* action during *posix\_spawn()* or *posix\_spawnnp()* will have the parent’s real  
 28963 ID as its owner; and an open by the parent process may successfully open a file to which the real  
 28964 user should not have access or fail to open a file to which the real user should have access.

28965 **File Descriptor Mapping**

28966 The standard developers had originally proposed using an array which specified the mapping of  
28967 child file descriptors back to those of the parent. It was pointed out by the ballot group that it is  
28968 not possible to reshuffle file descriptors arbitrarily in a library implementation of *posix\_spawn()*  
28969 or *posix\_spawnnp()* without provision for one or more spare file descriptor entries (which simply  
28970 may not be available). Such an array requires that an implementation develop a complex  
28971 strategy to achieve the desired mapping without inadvertently closing the wrong file descriptor  
28972 at the wrong time.

28973 It was noted by a member of the Ada Language Bindings working group that the approved Ada  
28974 Language *Start\_Process* family of POSIX process primitives use a caller-specified set of file  
28975 actions to alter the normal *fork()/exec* semantics for inheritance of file descriptors in a very  
28976 flexible way, yet no such problems exist because the burden of determining how to achieve the  
28977 final file descriptor mapping is completely on the application. Furthermore, although the file  
28978 actions interface appears frightening at first glance, it is actually quite simple to implement in  
28979 either a library or the kernel.

28980 **FUTURE DIRECTIONS**

28981 None.

28982 **SEE ALSO**

28983 *close()*, *dup()*, *open()*, *posix\_spawn()*, *posix\_spawn\_file\_actions\_adddup2()*,  
28984 *posix\_spawn\_file\_actions\_destroy()*, *posix\_spawnnp()*, the Base Definitions volume of  
28985 IEEE Std. 1003.1-200x, <spawn.h>

28986 **CHANGE HISTORY**

28987 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

28988 **NAME**

28989        posix\_spawn\_file\_actions\_adddup2 — add dup2 action to spawn file actions object  
 28990        (**REALTIME**)

28991 **SYNOPSIS**

```
28992 SPN #include <spawn.h>

28993 int posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *
28994 file_actions, int fildes, int newfildes);
28995
```

28996 **DESCRIPTION**

28997        A spawn file actions object is as defined in *posix\_spawn\_file\_actions\_addclose()*.  
 28998        The *posix\_spawn\_file\_actions\_adddup2()* function adds a *dup2()* action to the object referenced by  
 28999        *file\_actions* that will cause the file descriptor *fildes* to be duplicated as *newfildes* (as if *dup2(fildes,*  
 29000        *newfildes)* had been called) when a new process is spawned using this file actions object.

29001 **RETURN VALUE**

29002        Upon successful completion, the *posix\_spawn\_file\_actions\_adddup2()* function shall return zero;  
 29003        otherwise, an error number shall be returned to indicate the error.

29004 **ERRORS**

29005        The *posix\_spawn\_file\_actions\_adddup2()* function shall fail if:

- 29006        [EBADF]        The value specified by *fildes* or *newfildes* is negative or greater than or equal to  
 29007        {OPEN\_MAX}.
- 29008        [ENOMEM]     Insufficient memory exists to add to the spawn file actions object.

29009        The *posix\_spawn\_file\_actions\_adddup2()* function may fail if:

- 29010        [EINVAL]     The value specified by *file\_actions* is invalid.

29011        It shall not be considered an error for the *fildes* argument passed to the  
 29012        *posix\_spawn\_file\_actions\_adddup2()* function to specify a file descriptor for which the specified  
 29013        operation could not be performed at the time of the call. Any such error will be detected when  
 29014        the associated file actions object is later used during a *posix\_spawn()* or *posix\_spawnnp()*  
 29015        operation.

29016 **EXAMPLES**

29017        None.

29018 **APPLICATION USAGE**

29019        The *posix\_spawn\_file\_actions\_adddup2()* function is part of the Spawn option and need not be  
 29020        provided on all implementations.

29021 **RATIONALE**

29022        Refer to the RATIONALE in *posix\_spawn\_file\_actions\_addclose()*.

29023 **FUTURE DIRECTIONS**

29024        None.

29025 **SEE ALSO**

29026        *dup()*, *posix\_spawn()*, *posix\_spawn\_file\_actions\_addclose()*, *posix\_spawn\_file\_actions\_destroy()*,  
 29027        *posix\_spawnnp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <spawn.h>

29028 **CHANGE HISTORY**

29029 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

29030 IEEE PASC Interpretation 1003.1 #104 is included, noting that the [EBADF] error can apply to the

29031 *newfildes* argument in addition to *fildes*.

29032 **NAME**

29033 posix\_spawn\_file\_actions\_addopen — add open action to spawn file actions object  
29034 (**REALTIME**)

29035 **SYNOPSIS**

```
29036 SPN #include <spawn.h>
29037 int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict
29038 file_actions, int fildes, const char *restrict path,
29039 int oflag, mode_t mode);
29040
```

29041 **DESCRIPTION**

29042 Refer to *posix\_spawn\_file\_actions\_addclose()*.

29043 **NAME**

29044 posix\_spawn\_file\_actions\_destroy, posix\_spawn\_file\_actions\_init — destroy and initialize  
 29045 spawn file actions object (**REALTIME**)

29046 **SYNOPSIS**

```
29047 SPN #include <spawn.h>
29048
29048 int posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *
29049 file_actions);
29050 int posix_spawn_file_actions_init(posix_spawn_file_actions_t *
29051 file_actions);
29052
```

29053 **DESCRIPTION**

29054 A spawn file actions object is as defined in *posix\_spawn\_file\_actions\_addclose()*.

29055 The *posix\_spawn\_file\_actions\_destroy()* function destroys the object referenced by *file\_actions*; the  
 29056 object becomes, in effect, uninitialized. An implementation may cause  
 29057 *posix\_spawn\_file\_actions\_destroy()* to set the object referenced by *file\_actions* to an invalid value. A  
 29058 destroyed spawn file actions object can be reinitialized using *posix\_spawn\_file\_actions\_init()*; the  
 29059 results of otherwise referencing the object after it has been destroyed are undefined.

29060 The *posix\_spawn\_file\_actions\_init()* function initializes the object referenced by *file\_actions* to  
 29061 contain no file actions for *posix\_spawn()* or *posix\_spawnnp()* to perform.

29062 The effect of initializing an already initialized spawn file actions object is undefined.

29063 **RETURN VALUE**

29064 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 29065 be returned to indicate the error.

29066 **ERRORS**

29067 The *posix\_spawn\_file\_actions\_init()* function shall fail if:

29068 [ENOMEM] Insufficient memory exists to initialize the spawn file actions object.

29069 The *posix\_spawn\_file\_actions\_destroy()* function may fail if:

29070 [EINVAL] The value specified by *file\_actions* is invalid.

29071 **EXAMPLES**

29072 None.

29073 **APPLICATION USAGE**

29074 These functions are part of the Spawn option and need not be provided on all implementations.

29075 **RATIONALE**

29076 Refer to the RATIONALE in *posix\_spawn\_file\_actions\_addclose()*.

29077 **FUTURE DIRECTIONS**

29078 None.

29079 **SEE ALSO**

29080 *posix\_spawn()*, *posix\_spawnnp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<spawn.h>**

29081 **CHANGE HISTORY**

29082 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

29083 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

29084 **NAME**

29085        `posix_spawn_file_actions_init` — initialize spawn file actions object (**REALTIME**)

29086 **SYNOPSIS**

29087 SPN        `#include <spawn.h>`

```
29088 int posix_spawn_file_actions_init(posix_spawn_file_actions_t *
29089 file_actions);
```

29090

29091 **DESCRIPTION**

29092        Refer to `posix_spawn_file_actions_destroy()`.



## 29093 NAME

29094 posix\_spawnattr\_destroy, posix\_spawnattr\_init — destroy and initialize spawn attributes object  
 29095 (REALTIME)

## 29096 SYNOPSIS

29097 SPN `#include <spawn.h>`

29098 `int posix_spawnattr_destroy(posix_spawnattr_t *attr);`

29099 `int posix_spawnattr_init(posix_spawnattr_t *attr);`

29100

## 29101 DESCRIPTION

29102 A spawn attributes object is of type **posix\_spawnattr\_t** (defined in `<spawn.h>`) and is used to  
 29103 specify the inheritance of process attributes across a spawn operation. IEEE Std. 1003.1-200x  
 29104 does not define comparison or assignment operators for the type **posix\_spawnattr\_t**.

29105 The `posix_spawnattr_destroy()` function destroys a spawn attributes object. The effect of  
 29106 subsequent use of the object is undefined until the object is reinitialized by another call to  
 29107 `posix_spawnattr_init()`. An implementation may cause `posix_spawnattr_destroy()` to set the object  
 29108 referenced by `attr` to an invalid value.

29109 The `posix_spawnattr_init()` function initializes a spawn attributes object `attr` with the default  
 29110 value for all of the individual attributes used by the implementation. The effect of initializing an  
 29111 already initialized spawn attributes option is undefined.

29112 Each implementation shall document the individual attributes it uses and their default values  
 29113 unless these values are defined by IEEE Std. 1003.1-200x. Attributes not defined by  
 29114 IEEE Std. 1003.1-200x, their default values, and the names of the associated functions to get and  
 29115 set those attribute values are implementation-defined.

29116 The resulting spawn attributes object (possibly modified by setting individual attribute values),  
 29117 is used to modify the behavior of `posix_spawn()` or `posix_spawnp()`. After a spawn attributes  
 29118 object has been used to spawn a process by a call to a `posix_spawn()` or `posix_spawnp()`, any  
 29119 function affecting the attributes object (including destruction) does not affect any process that  
 29120 has been spawned in this way.

## 29121 RETURN VALUE

29122 Upon successful completion, `posix_spawnattr_destroy()` and `posix_spawnattr_init()` shall return  
 29123 zero; otherwise, an error number shall be returned to indicate the error.

## 29124 ERRORS

29125 The `posix_spawnattr_init()` function shall fail if:

29126 [ENOMEM] Insufficient memory exists to initialize the spawn attributes object.

29127 The `posix_spawnattr_destroy()` function may fail if:

29128 [EINVAL] The value specified by `attr` is invalid.

## 29129 EXAMPLES

29130 None.

## 29131 APPLICATION USAGE

29132 These functions are part of the Spawn option and need not be provided on all implementations.

## 29133 RATIONALE

29134 The original spawn interface proposed in IEEE Std. 1003.1-200x defined the attributes that  
 29135 specify the inheritance of process attributes across a spawn operation as a structure. In order to  
 29136 be able to separate optional individual attributes under their appropriate options (that is, the  
 29137 `spawn-schedparam` and `spawn-schedpolicy` attributes depending upon the Process Scheduling

29138 option), and also for extensibility and consistency with the newer POSIX interfaces, the  
29139 attributes interface has been changed to an opaque data type. This interface now consists of the  
29140 type **posix\_spawnattr\_t**, representing a spawn attributes object, together with associated  
29141 functions to initialize or destroy the attributes object, and to set or get each individual attribute.  
29142 Although the new object-oriented interface is more verbose than the original structure, it is  
29143 simple to use, more extensible, and easy to implement.

## 29144 FUTURE DIRECTIONS

29145 None.

## 29146 SEE ALSO

29147 *posix\_spawn()*, *posix\_spawnattr\_getsigdefault()*, *posix\_spawnattr\_getflags()*,  
29148 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
29149 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*,  
29150 *posix\_spawnattr\_setpgroup()*, *posix\_spawnattr\_setsigmask()*, *posix\_spawnattr\_setschedpolicy()*,  
29151 *posix\_spawnattr\_setschedparam()*, *posix\_spawnnp()*, the Base Definitions volume of  
29152 IEEE Std. 1003.1-200x, <spawn.h>

## 29153 CHANGE HISTORY

29154 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

29155 IEEE PASC Interpretation 1003.1 #106 is included, noting that the effect of initializing an already  
29156 initialized spawn attributes option is undefined.

29157 **NAME**

29158 posix\_spawnattr\_getflags, posix\_spawnattr\_setflags — get and set spawn-flags attribute of  
 29159 spawn attributes object (**REALTIME**)

29160 **SYNOPSIS**

```
29161 SPN #include <spawn.h>
29162 int posix_spawnattr_getflags(const posix_spawnattr_t *restrict attr,
29163 short *restrict flags);
29164 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
29165
```

29166 **DESCRIPTION**

29167 The *spawn-flags* attribute is used to indicate which process attributes are to be changed in the  
 29168 new process image when invoking *posix\_spawn()* or *posix\_spawnp()*. It is the bitwise-inclusive  
 29169 OR of zero or more of the flags POSIX\_SPAWN\_RESETIDS, POSIX\_SPAWN\_SETPGROUP,  
 29170 PS POSIX\_SPAWN\_SETSIGDEF, and POSIX\_SPAWN\_SETSIGMASK,  
 29171 POSIX\_SPAWN\_SETSCHEDPARAM, and POSIX\_SPAWN\_SETSCHEDULER. In addition, if the  
 29172 Process Scheduling option is supported, the flags POSIX\_SPAWN\_SETSCHEDPARAM and  
 29173 POSIX\_SPAWN\_SETSCHEDULER shall also be supported. These flags are defined in  
 29174 <spawn.h>. The default value of this attribute shall be with no flags set.

29175 The *posix\_spawnattr\_getflags()* function obtains the value of the *spawn-flags* attribute from the  
 29176 attributes object referenced by *attr*.

29177 The *posix\_spawnattr\_setflags()* function is used to set the *spawn-flags* attribute in an initialized  
 29178 attributes object referenced by *attr*.

29179 **RETURN VALUE**

29180 Upon successful completion, *posix\_spawnattr\_getflags()* shall return zero and store the value of  
 29181 the *spawn-flags* attribute of *attr* into the object referenced by the *flags* parameter; otherwise, an  
 29182 error number shall be returned to indicate the error.

29183 Upon successful completion, *posix\_spawnattr\_setflags()* shall return zero; otherwise, an error  
 29184 number shall be returned to indicate the error.

29185 **ERRORS**

29186 These functions may fail if:

29187 [EINVAL] The value specified by *attr* is invalid.

29188 The *posix\_spawnattr\_setflags()* function may fail if:

29189 [EINVAL] The value of the attribute being set is not valid.

29190 **EXAMPLES**

29191 None.

29192 **APPLICATION USAGE**

29193 These functions are part of the Spawn option and need not be provided on all implementations.

29194 **RATIONALE**

29195 None.

29196 **FUTURE DIRECTIONS**

29197 None.

29198 **SEE ALSO**

29199 *posix\_spawn()*, *posix\_spawnattr\_destroy()* (on page 1405), *posix\_spawnattr\_init()* (on page 1419), *posix\_spawnattr\_getsigdefault()*,  
29200 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
29201 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setpgroup()*,  
29202 *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
29203 *posix\_spawnnp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <spawn.h>

29205 **CHANGE HISTORY**

29206 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

29207 **NAME**

29208 posix\_spawnattr\_getpgroup, posix\_spawnattr\_setpgroup — get and set spawn-pgroup attribute  
 29209 of spawn attributes object (**REALTIME**)

29210 **SYNOPSIS**

29211 SPN `#include <spawn.h>`

29212 `int posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict attr,`  
 29213 `pid_t *restrict pgroup);`

29214 `int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);`  
 29215

29216 **DESCRIPTION**

29217 The *spawn-pgroup* attribute represents the process group to be joined by the new process image  
 29218 in a spawn operation (if `POSIX_SPAWN_SETPGROUP` is set in the *spawn-flags* attribute). The  
 29219 default value of this attribute shall be zero.

29220 The *posix\_spawnattr\_getpgroup()* function obtains the value of the *spawn-pgroup* attribute from  
 29221 the attributes object referenced by *attr*.

29222 The *posix\_spawnattr\_setpgroup()* function is used to set the *spawn-pgroup* attribute in an  
 29223 initialized attributes object referenced by *attr*.

29224 **RETURN VALUE**

29225 Upon successful completion, *posix\_spawnattr\_getpgroup()* shall return zero and store the value of  
 29226 the *spawn-pgroup* attribute of *attr* into the object referenced by the *pgroup* parameter; otherwise,  
 29227 an error number shall be returned to indicate the error.

29228 Upon successful completion, *posix\_spawnattr\_setpgroup()* shall return zero; otherwise, an error  
 29229 number shall be returned to indicate the error.

29230 **ERRORS**

29231 These functions may fail if:

29232 [EINVAL] The value specified by *attr* is invalid.

29233 The *posix\_spawnattr\_setpgroup()* function may fail if:

29234 [EINVAL] The value of the attribute being set is not valid.

29235 **EXAMPLES**

29236 None.

29237 **APPLICATION USAGE**

29238 These functions are part of the Spawn option and need not be provided on all implementations.

29239 **RATIONALE**

29240 None.

29241 **FUTURE DIRECTIONS**

29242 None.

29243 **SEE ALSO**

29244 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*, *posix\_spawnattr\_getsigdefault()*,  
 29245 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
 29246 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*,  
 29247 *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
 29248 *posix\_spawnp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<spawn.h>`

29249 **CHANGE HISTORY**

29250 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

29251 **NAME**

29252 `posix_spawnattr_getschedparam`, `posix_spawnattr_setschedparam` — get and set spawn-  
 29253 `schedparam` attribute of spawn attributes object (**REALTIME**)

29254 **SYNOPSIS**

```
29255 SPN PS #include <spawn.h>
29256 #include <sched.h>

29257 int posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict attr,
29258 struct sched_param *restrict schedparam);
29259 int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
29260 const struct sched_param *restrict schedparam);
29261
```

29262 **DESCRIPTION**

29263 The *spawn-schedparam* attribute represents the scheduling parameters to be assigned to the new  
 29264 process image in a spawn operation (if `POSIX_SPAWN_SETSCHEDULER` or  
 29265 `POSIX_SPAWN_SETSCHEDPARAM` is set in the *spawn-flags* attribute). The default value of this  
 29266 attribute is unspecified.

29267 The *posix\_spawnattr\_getschedparam()* function obtains the value of the *spawn-schedparam* attribute  
 29268 from the attributes object referenced by *attr*.

29269 The *posix\_spawnattr\_setschedparam()* function is used to set the *spawn-schedparam* attribute in an  
 29270 initialized attributes object referenced by *attr*.

29271 **RETURN VALUE**

29272 Upon successful completion, *posix\_spawnattr\_getschedparam()* shall return zero and store the  
 29273 value of the *spawn-schedparam* attribute of *attr* into the object referenced by the *schedparam*  
 29274 parameter; otherwise, an error number shall be returned to indicate the error.

29275 Upon successful completion, *posix\_spawnattr\_setschedparam()* shall return zero; otherwise, an  
 29276 error number shall be returned to indicate the error.

29277 **ERRORS**

29278 These functions may fail if:

29279 [EINVAL] The value specified by *attr* is invalid.

29280 The *posix\_spawnattr\_setschedparam()* function may fail if:

29281 [EINVAL] The value of the attribute being set is not valid.

29282 **EXAMPLES**

29283 None.

29284 **APPLICATION USAGE**

29285 These functions are part of the Spawn and Process Scheduling options and need not be provided  
 29286 on all implementations.

29287 **RATIONALE**

29288 None.

29289 **FUTURE DIRECTIONS**

29290 None.

29291 **SEE ALSO**

29292 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*, *posix\_spawnattr\_getsigdefault()*,  
 29293 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedpolicy()*,  
 29294 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*,

29295 *posix\_spawnattr\_setpgroup()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
29296 *posix\_spawnnp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sched.h>, <spawn.h>

29297 **CHANGE HISTORY**

29298 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.



29299 **NAME**

29300 `posix_spawnattr_getschedpolicy`, `posix_spawnattr_setschedpolicy` — get and set spawn-  
 29301 `schedpolicy` attribute of spawn attributes object (**REALTIME**)

29302 **SYNOPSIS**

```
29303 SPN PS #include <spawn.h>
29304 #include <sched.h>

29305 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict attr,
29306 int *restrict schedpolicy);
29307 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
29308 int schedpolicy);
29309
```

29310 **DESCRIPTION**

29311 The *spawn-schedpolicy* attribute represents the scheduling policy to be assigned to the new  
 29312 process image in a spawn operation (if `POSIX_SPAWN_SETSCHEDULER` is set in the *spawn-*  
 29313 *flags* attribute). The default value of this attribute is unspecified.

29314 The *posix\_spawnattr\_getschedpolicy()* function obtains the value of the *spawn-schedpolicy* attribute  
 29315 from the attributes object referenced by *attr*.

29316 The *posix\_spawnattr\_setschedpolicy()* function is used to set the *spawn-schedpolicy* attribute in an  
 29317 initialized attributes object referenced by *attr*.

29318 **RETURN VALUE**

29319 Upon successful completion, *posix\_spawnattr\_getschedpolicy()* shall return zero and store the  
 29320 value of the *spawn-schedpolicy* attribute of *attr* into the object referenced by the *schedpolicy*  
 29321 parameter; otherwise, an error number shall be returned to indicate the error.

29322 Upon successful completion, *posix\_spawnattr\_setschedpolicy()* shall return zero; otherwise, an  
 29323 error number shall be returned to indicate the error.

29324 **ERRORS**

29325 These functions may fail if:

29326 [EINVAL] The value specified by *attr* is invalid.

29327 The *posix\_spawnattr\_setschedpolicy()* function may fail if:

29328 [EINVAL] The value of the attribute being set is not valid.

29329 **EXAMPLES**

29330 None.

29331 **APPLICATION USAGE**

29332 These functions are part of the Spawn and Process Scheduling options and need not be provided  
 29333 on all implementations.

29334 **RATIONALE**

29335 None.

29336 **FUTURE DIRECTIONS**

29337 None.

29338 **SEE ALSO**

29339 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*, *posix\_spawnattr\_getsigdefault()*,  
 29340 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*,  
 29341 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*,  
 29342 *posix\_spawnattr\_setpgroup()*, *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setsigmask()*,

- 29343            *posix\_spawnp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sched.h>, <spawn.h>
- 29344 **CHANGE HISTORY**
- 29345            First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

29346 **NAME**

29347 posix\_spawnattr\_getsigdefault, posix\_spawnattr\_setsigdefault — get and set spawn-sigdefault  
 29348 attribute of spawn attributes object (**REALTIME**)

29349 **SYNOPSIS**

```
29350 SPN #include <signal.h>
29351 #include <spawn.h>

29352 int posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict attr,
29353 sigset_t *restrict sigdefault);
29354 int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
29355 const sigset_t *restrict sigdefault);
29356
```

29357 **DESCRIPTION**

29358 The *spawn-sigdefault* attribute represents the set of signals to be forced to default signal handling  
 29359 in the new process image (if POSIX\_SPAWN\_SETSIGDEF is set in the *spawn-flags* attribute) by a  
 29360 spawn operation. The default value of this attribute shall be an empty signal set.

29361 The *posix\_spawnattr\_getsigdefault()* function obtains the value of the *spawn-sigdefault* attribute  
 29362 from the attributes object referenced by *attr*.

29363 The *posix\_spawnattr\_setsigdefault()* function is used to set the *spawn-sigdefault* attribute in an  
 29364 initialized attributes object referenced by *attr*.

29365 **RETURN VALUE**

29366 Upon successful completion, *posix\_spawnattr\_getsigdefault()* shall return zero and store the value  
 29367 of the *spawn-sigdefault* attribute of *attr* into the object referenced by the *sigdefault* parameter;  
 29368 otherwise, an error number shall be returned to indicate the error.

29369 Upon successful completion, *posix\_spawnattr\_setsigdefault()* shall return zero; otherwise, an error  
 29370 number shall be returned to indicate the error.

29371 **ERRORS**

29372 These functions may fail if:

29373 [EINVAL] The value specified by *attr* is invalid.

29374 The *posix\_spawnattr\_setsigdefault()* function may fail if:

29375 [EINVAL] The value of the attribute being set is not valid.

29376 **EXAMPLES**

29377 None.

29378 **APPLICATION USAGE**

29379 These functions are part of the Spawn option and need not be provided on all implementations.

29380 **RATIONALE**

29381 None.

29382 **FUTURE DIRECTIONS**

29383 None.

29384 **SEE ALSO**

29385 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*, *posix\_spawnattr\_getflags()*,  
 29386 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
 29387 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setflags()*, *posix\_spawnattr\_setpgroup()*,  
 29388 *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
 29389 *posix\_spawnnp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <signal.h>, <spawn.h>

29390 **CHANGE HISTORY**

29391 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

29392 **NAME**

29393 posix\_spawnattr\_getsigmask, posix\_spawnattr\_setsigmask — get and set spawn-sigmask  
 29394 attribute of spawn attributes object (**REALTIME**)

29395 **SYNOPSIS**

```
29396 SPN #include <signal.h>
29397 #include <spawn.h>

29398 int posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict attr,
29399 sigset_t *restrict sigmask);
29400 int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
29401 const sigset_t *restrict sigmask);
29402
```

29403 **DESCRIPTION**

29404 The *spawn-sigmask* attribute represents the signal mask in effect in the new process image of a  
 29405 spawn operation (if `POSIX_SPAWN_SETSIGMASK` is set in the *spawn-flags* attribute). The  
 29406 default value of this attribute is unspecified.

29407 The *posix\_spawnattr\_getsigmask()* function obtains the value of the *spawn-sigmask* attribute from  
 29408 the attributes object referenced by *attr*.

29409 The *posix\_spawnattr\_setsigmask()* function is used to set the *spawn-sigmask* attribute in an  
 29410 initialized attributes object referenced by *attr*.

29411 **RETURN VALUE**

29412 Upon successful completion, *posix\_spawnattr\_getsigmask()* shall return zero and store the value  
 29413 of the *spawn-sigmask* attribute of *attr* into the object referenced by the *sigmask* parameter;  
 29414 otherwise, an error number shall be returned to indicate the error.

29415 Upon successful completion, *posix\_spawnattr\_setsigmask()* shall return zero; otherwise, an error  
 29416 number shall be returned to indicate the error.

29417 **ERRORS**

29418 These functions may fail if:

29419 [EINVAL] The value specified by *attr* is invalid.

29420 The *posix\_spawnattr\_setsigmask()* function may fail if:

29421 [EINVAL] The value of the attribute being set is not valid.

29422 **EXAMPLES**

29423 None.

29424 **APPLICATION USAGE**

29425 These functions are part of the Spawn option and need not be provided on all implementations.

29426 **RATIONALE**

29427 None.

29428 **FUTURE DIRECTIONS**

29429 None.

29430 **SEE ALSO**

29431 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*, *posix\_spawnattr\_getsigdefault()*,  
 29432 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*,  
 29433 *posix\_spawnattr\_getschedpolicy()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*,  
 29434 *posix\_spawnattr\_setpgroup()*, *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setschedpolicy()*,  
 29435 *posix\_spawnnp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<signal.h>`, `<spawn.h>`

29436 **CHANGE HISTORY**

29437 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

29438 **NAME**29439 `posix_spawnattr_init` — initialize spawn attributes object (**REALTIME**)29440 **SYNOPSIS**29441 SPN `#include <spawn.h>`29442 `int posix_spawnattr_init(posix_spawnattr_t *attr);`

29443

29444 **DESCRIPTION**29445 Refer to `posix_spawnattr_destroy()`.

29446 **NAME**

29447        `posix_spawnattr_setflags` — set spawn-flags attribute of spawn attributes object (**REALTIME**)

29448 **SYNOPSIS**

29449 SPN    `#include <spawn.h>`

29450        `int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);`

29451

29452 **DESCRIPTION**

29453        Refer to `posix_spawnattr_getflags()`.



29454 **NAME**

29455 posix\_spawnattr\_setpgroup — set spawn-pgroup attribute of spawn attributes object  
29456 (**REALTIME**)

29457 **SYNOPSIS**

29458 SPN `#include <spawn.h>`

29459 `int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);`

29460

29461 **DESCRIPTION**

29462 Refer to *posix\_spawnattr\_getpgroup()*.

29463 **NAME**

29464 posix\_spawnattr\_setschedparam — set spawn-schedparam attribute of spawn attributes object  
29465 (**REALTIME**)

29466 **SYNOPSIS**

29467 SPN PS #include <sched.h>

29468 #include <spawn.h>

```
29469 int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
29470 const struct sched_param *restrict schedparam);
```

29471

29472 **DESCRIPTION**

29473 Refer to *posix\_spawnattr\_getschedparam()*.

29474 **NAME**

29475 `posix_spawnattr_setschedpolicy` — set spawn-schedpolicy attribute of spawn attributes object  
29476 **(REALTIME)**

29477 **SYNOPSIS**

29478 SPN PS `#include <sched.h>`

29479 `#include <spawn.h>`

```
29480 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
29481 int schedpolicy);
```

29482

29483 **DESCRIPTION**

29484 Refer to `posix_spawnattr_getschedpolicy()`.

29485 **NAME**

29486 posix\_spawnattr\_setsigdefault — set spawn-sigdefault attribute of spawn attributes object  
29487 (**REALTIME**)

29488 **SYNOPSIS**

29489 SPN #include <signal.h>

29490 #include <spawn.h>

29491 int posix\_spawnattr\_setsigdefault(posix\_spawnattr\_t \*restrict attr,  
29492 const sigset\_t \*restrict sigdefault);

29493

29494 **DESCRIPTION**

29495 Refer to *posix\_spawnattr\_getsigdefault()*.

29496 **NAME**

29497 `posix_spawnattr_setsigmask` — set spawn-sigmask attribute of spawn attributes object  
29498 **(REALTIME)**

29499 **SYNOPSIS**

29500 SPN `#include <signal.h>`

29501 `#include <spawn.h>`

29502 `int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,`  
29503 `const sigset_t *restrict sigmask);`

29504

29505 **DESCRIPTION**

29506 Refer to `posix_spawnattr_getsigmask()`.

29507 **NAME**

29508        posix\_spawnnp — spawn a process (**REALTIME**)

29509 **SYNOPSIS**

29510 SPN    #include <spawn.h>

```
29511 int posix_spawnnp(pid_t *restrict pid, const char *restrict file,
29512 const posix_spawn_file_actions_t *file_actions,
29513 const posix_spawnattr_t *restrict attrp,
29514 char *const argv[restrict], char *const envp[restrict]);
29515
```

29516 **DESCRIPTION**

29517        Refer to *posix\_spawn()*.

29518 **NAME**

29519        posix\_trace\_attr\_destroy, posix\_trace\_attr\_init — trace stream attributes object destroy and  
29520        initialization

29521 **SYNOPSIS**

29522 TRC     #include <trace.h>

29523        int posix\_trace\_attr\_destroy(trace\_attr\_t \*attr);

29524        int posix\_trace\_attr\_init(trace\_attr\_t \*attr);

29525

29526 **DESCRIPTION**

29527        The *posix\_trace\_attr\_destroy()* function is used to destroy an initialized trace attributes object.  
29528        The results of using the attributes object after it has been destroyed are unspecified. A destroyed  
29529        trace attributes object can be reinitialized using *posix\_trace\_attr\_init()*.

29530        The *posix\_trace\_attr\_init()* function initializes a trace attributes object *attr* with the default value  
29531        for all of the individual attributes used by a given implementation. The read-only *generation-*  
29532        *version* and *clock-resolution* attributes of the newly initialized trace attributes object shall be set to  
29533        their appropriate values (Section 2.11.1.2 (on page 578)).

29534        The effect of initializing an already-initialized trace attributes object is unspecified.

29535        Implementations may add extensions to the trace attributes object structure as permitted in the  
29536        Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 2, Conformance.

29537        The resulting attributes object (possibly modified by setting individual attributes values), when  
29538        used by *posix\_trace\_create()*, defines the attributes of the trace stream created. A single attributes  
29539        object can be used in multiple calls to *posix\_trace\_create()*. After one or more trace streams have  
29540        been created using an attributes object, any function affecting that attributes object, including  
29541        destruction, does not affect any trace stream previously created. An initialized attributes object  
29542        also serves to receive the attributes of an existing trace stream or trace log when calling the  
29543        *posix\_trace\_get\_attr()* function.

29544 **RETURN VALUE**

29545        Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
29546        return the corresponding error number.

29547 **ERRORS**

29548        The *posix\_trace\_attr\_destroy()* function may fail if:

29549        [EINVAL]        The value of *attr* is invalid.

29550        The *posix\_trace\_attr\_init()* function shall fail if:

29551        [ENOMEM]       Insufficient memory exists to initialize the trace attributes object.

29552 **EXAMPLES**

29553        None.

29554 **APPLICATION USAGE**

29555        None.

29556 **RATIONALE**

29557        None.

29558 **FUTURE DIRECTIONS**

29559        None.

29560 **SEE ALSO**

29561            *posix\_trace\_create()*, *posix\_trace\_get\_attr()*, *uname()*, the Base Definitions volume of  
29562            IEEE Std. 1003.1-200x, <trace.h>

29563 **CHANGE HISTORY**

29564            First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.



29565 **NAME**

29566 `posix_trace_attr_getclockres`, `posix_trace_attr_getcreatetime`, `posix_trace_attr_getgenversion`,  
 29567 `posix_trace_attr_getname`, `posix_trace_attr_setname` — retrieve and set information about a  
 29568 trace stream

29569 **SYNOPSIS**

```
29570 TRC #include <time.h>
29571 #include <trace.h>

29572 int posix_trace_attr_getclockres(const trace_attr_t *attr,
29573 struct timespec *resolution);
29574 int posix_trace_attr_getcreatetime(const trace_attr_t *attr,
29575 struct timespec *createtime);

29576 #include <trace.h>

29577 int posix_trace_attr_getgenversion(const trace_attr_t *attr,
29578 char *genversion);
29579 int posix_trace_attr_getname(const trace_attr_t *attr,
29580 char *tracename);
29581 int posix_trace_attr_setname(trace_attr_t *attr,
29582 const char *tracename);
29583
```

29584 **DESCRIPTION**

29585 The `posix_trace_attr_getclockres()` function shall copy the clock resolution of the clock used to  
 29586 generate timestamps from the *clock-resolution* attribute of the attributes object pointed to by the  
 29587 *attr* argument into the structure pointed to by the *resolution* argument.

29588 The `posix_trace_attr_getcreatetime()` function shall copy the trace stream creation time from the  
 29589 *creation-time* attribute of the attributes object pointed to by the *attr* argument into the structure  
 29590 pointed to by the *createtime* argument. The *creation-time* attribute shall represent the time of  
 29591 creation of the trace stream.

29592 The `posix_trace_attr_getgenversion()` function shall copy the string containing version information  
 29593 from the *generation-version* attribute of the attributes object pointed to by the *attr* argument into  
 29594 the string pointed to by the *genversion* argument. The *genversion* argument shall be the address of  
 29595 a character array which can store at least {TRACE\_NAME\_MAX} characters.

29596 The `posix_trace_attr_getname()` function shall copy the string containing the trace name from the  
 29597 *trace-name* attribute of the attributes object pointed to by the *attr* argument into the string  
 29598 pointed to by the *tracename* argument. The *tracename* argument shall be the address of a character  
 29599 array which can store at least {TRACE\_NAME\_MAX} characters.

29600 The `posix_trace_attr_setname()` function shall set the name in the *trace-name* attribute of the  
 29601 attributes object pointed to by the *attr* argument, using the trace name string supplied by the  
 29602 *tracename* argument. If the supplied string contains more than {TRACE\_NAME\_MAX}  
 29603 characters, the name copied into the *trace-name* attribute may be truncated to one less than the  
 29604 length of {TRACE\_NAME\_MAX} characters. The default value is a null string.

29605 **RETURN VALUE**

29606 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 29607 return the corresponding error number.

29608 If successful, the `posix_trace_attr_getclockres()` function stores the *clock-resolution* attribute value  
 29609 in the object pointed to by *resolution*. Otherwise, the content of this object is unspecified.

29610 If successful, the *posix\_trace\_attr\_getcreatetime()* function stores the trace stream creation time in  
 29611 the object pointed to by *createtime*. Otherwise, the content of this object is unspecified.

29612 If successful, the *posix\_trace\_attr\_getgenversion()* function stores the trace version information in  
 29613 the string pointed to by *genversion*. Otherwise, the content of this string is unspecified.

29614 If successful, the *posix\_trace\_attr\_getname()* function stores the trace name in the string pointed  
 29615 to by *tracename*. Otherwise, the content of this string is unspecified.

29616 **ERRORS**

29617 The *posix\_trace\_attr\_getclockres()*, *posix\_trace\_attr\_getcreatetime()*, *posix\_trace\_attr\_getgenversion()*,  
 29618 and *posix\_trace\_attr\_getname()* functions may fail if:

29619 [EINVAL] The value specified by one of the arguments is invalid.

29620 **EXAMPLES**

29621 None.

29622 **APPLICATION USAGE**

29623 None.

29624 **RATIONALE**

29625 None.

29626 **FUTURE DIRECTIONS**

29627 None.

29628 **SEE ALSO**

29629 *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_get\_attr()*, *uname()*, the Base Definitions  
 29630 volume of IEEE Std. 1003.1-200x, <time.h>, <trace.h>

29631 **CHANGE HISTORY**

29632 First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

## 29633 NAME

29634 posix\_trace\_attr\_getinherited, posix\_trace\_attr\_getlogfullpolicy,  
 29635 posix\_trace\_attr\_getstreamfullpolicy, posix\_trace\_attr\_setinherited,  
 29636 posix\_trace\_attr\_setlogfullpolicy, posix\_trace\_attr\_setstreamfullpolicy — retrieve and set the  
 29637 behavior of a trace stream

## 29638 SYNOPSIS

```
29639 TRC #include <trace.h>
29640 TRC TRI int posix_trace_attr_getinherited(const trace_attr_t *attr,
29641 int *inheritancepolicy);
29642 TRC TRL int posix_trace_attr_getlogfullpolicy(const trace_attr_t *attr,
29643 int *logpolicy);
29644 TRC int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *attr,
29645 int *streampolicy);
29646 TRC TRI int posix_trace_attr_setinherited(trace_attr_t *attr,
29647 int inheritancepolicy);
29648 TRC TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
29649 int logpolicy);
29650 TRC int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
29651 int streampolicy);
29652
```

## 29653 DESCRIPTION

29654 TRI The *posix\_trace\_attr\_getinherited()* and *posix\_trace\_attr\_setinherited()* functions, respectively, get  
 29655 and set the inheritance policy stored in the *inheritance* attribute for traced processes across the  
 29656 *fork()* and *spawn()* operations. The *inheritance* attribute of the attributes object pointed to by the  
 29657 *attr* argument shall be set to one of the following values defined by manifest constants in the  
 29658 **<trace.h>** header:

## 29659 POSIX\_TRACE\_CLOSE\_FOR\_CHILD

29660 After a *fork()* or *spawn()* operation, the child shall not be traced, and tracing of the parent  
 29661 shall continue.

## 29662 POSIX\_TRACE\_INHERITED

29663 After a *fork()* or *spawn()* operation, if the parent is being traced, its child shall be  
 29664 concurrently traced using the same trace stream.

29665 The default value for the *inheritance* attribute is **POSIX\_TRACE\_CLOSE\_FOR\_CHILD**.

29666 TRL The *posix\_trace\_attr\_getlogfullpolicy()* and *posix\_trace\_attr\_setlogfullpolicy()* functions,  
 29667 respectively, get and set the trace log full policy stored in the *log-full-policy* attribute of the  
 29668 attributes object pointed to by the *attr* argument.

29669 The *log-full-policy* attribute shall be set to one of the following values defined by manifest  
 29670 constants in the **<trace.h>** header:

## 29671 POSIX\_TRACE\_LOOP

29672 The trace log shall loop until the associated trace stream is stopped. This policy means that  
 29673 when the trace log gets full, the file system shall reuse the resources allocated to the oldest  
 29674 trace events that were recorded. In this way, the trace log will always contain the most  
 29675 recent trace events flushed.

## 29676 POSIX\_TRACE\_UNTIL\_FULL

29677 The trace stream shall be flushed to the trace log until the trace log is full. This condition can  
 29678 be deduced from the *posix\_log\_full\_status* member status (see the *posix\_trace\_status\_info()*  
 29679 function). The last recorded trace event shall be the **POSIX\_TRACE\_STOP** trace event.

29680 POSIX\_TRACE\_APPEND  
 29681 The associated trace stream shall be flushed to the trace log without log size limitation. If  
 29682 the application specifies POSIX\_TRACE\_APPEND, the implementation shall ignore the  
 29683 *log-max-size* attribute.

29684 The default value for the *log-full-policy* attribute is POSIX\_TRACE\_LOOP.

29685 The *posix\_trace\_attr\_getstreamfullpolicy()* and *posix\_trace\_attr\_setstreamfullpolicy()* functions,  
 29686 respectively, get and set the trace stream full policy stored in the *stream-full-policy* attribute of the  
 29687 attributes object pointed to by the *attr* argument.

29688 The *stream-full-policy* attribute shall be set to one of the following values defined by manifest  
 29689 constants in the <trace.h> header:

29690 POSIX\_TRACE\_LOOP  
 29691 The trace stream shall loop until explicitly stopped by the *posix\_trace\_stop()* function. This  
 29692 policy means that when the trace stream is full, the trace system shall reuse the resources  
 29693 allocated to the oldest trace events recorded. In this way, the trace stream will always  
 29694 contain the most recent trace events recorded.

29695 POSIX\_TRACE\_UNTIL\_FULL  
 29696 The trace stream will run until the trace stream resources are exhausted. Then the trace  
 29697 stream will stop. This condition can be deduced from *posix\_stream\_status* and  
 29698 *posix\_stream\_full\_status* statuses (see the *posix\_trace\_status\_info()* function). When this trace  
 29699 stream is read, a POSIX\_TRACE\_STOP trace event shall be reported after reporting the last  
 29700 recorded trace event. The trace system shall reuse the resources allocated to any trace  
 29701 events already reported—see the *posix\_trace\_getnext\_event()*, *posix\_trace\_trygetnext\_event()*,  
 29702 and *posix\_trace\_timedgetnext\_event()* functions—or already flushed for an active trace stream  
 29703 with log if the Trace Log option is supported; see the *posix\_trace\_flush()* function. The trace  
 29704 system shall restart the trace stream when it is empty and may restart it sooner. A  
 29705 POSIX\_TRACE\_START trace event shall be reported before reporting the next recorded  
 29706 trace event.

29707 POSIX\_TRACE\_FLUSH  
 29708 If the Trace Log option is supported, this policy is identical to the  
 29709 POSIX\_TRACE\_UNTIL\_FULL trace stream full policy except that the trace stream shall be  
 29710 flushed regularly as if *posix\_trace\_flush()* had been explicitly called. Defining this policy for  
 29711 an active trace stream without log shall be invalid.

29712 The default value for the *stream-full-policy* attribute shall be POSIX\_TRACE\_LOOP for an active  
 29713 trace stream without log.

29714 If the Trace Log option is supported, the default value for the *stream-full-policy* attribute shall be  
 29715 POSIX\_TRACE\_FLUSH for an active trace stream with log.

29716 **RETURN VALUE**  
 29717 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 29718 return the corresponding error number.

29719 TRI If successful, the *posix\_trace\_attr\_getinherited()* function stores the *inheritance* attribute value in  
 29720 the object pointed to by *inheritancepolicy*. Otherwise, the content of this object is undefined.

29721 TRL If successful, the *posix\_trace\_attr\_getlogfullpolicy()* function stores the *log-full-policy* attribute  
 29722 value in the object pointed to by *logpolicy*. Otherwise, the content of this object is undefined.

29723 If successful, the *posix\_trace\_attr\_getstreamfullpolicy()* function stores the *stream-full-policy*  
 29724 attribute value in the object pointed to by *streampolicy*. Otherwise, the content of this object is  
 29725 undefined.

29726 **ERRORS**

29727 These functions may fail if:

29728 [EINVAL] The value specified by at least one of the arguments is invalid.

29729 **EXAMPLES**

29730 None.

29731 **APPLICATION USAGE**

29732 None.

29733 **RATIONALE**

29734 None.

29735 **FUTURE DIRECTIONS**

29736 None.

29737 **SEE ALSO**

29738 *fork()*, *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_flush()*, *posix\_trace\_get\_attr()*,  
29739 *posix\_trace\_getnext\_event()*, *posix\_trace\_start()*, **posix\_trace\_status\_info Structure**,  
29740 *posix\_trace\_timedgetnext\_event()*, <REFERENCE UNDEFINED>(spawn), the Base Definitions  
29741 volume of IEEE Std. 1003.1-200x, <trace.h>

29742 **CHANGE HISTORY**

29743 First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

29744 **NAME**

29745 posix\_trace\_attr\_getlogsize, posix\_trace\_attr\_getmaxdatasize,  
 29746 posix\_trace\_attr\_getmaxsystemeventsize, posix\_trace\_attr\_getmaxusereventsize,  
 29747 posix\_trace\_attr\_getstreamsize, posix\_trace\_attr\_setlogsize, posix\_trace\_attr\_setmaxdatasize,  
 29748 posix\_trace\_attr\_setstreamsize — retrieve and set trace stream size attributes

29749 **SYNOPSIS**

```

29750 TRC #include <sys/types.h>
29751 #include <trace.h>

29752 TRC TRL int posix_trace_attr_getlogsize(const trace_attr_t *attr,
29753 size_t *logsize);
29754 TRC int posix_trace_attr_getmaxdatasize(const trace_attr_t *attr,
29755 size_t *maxdatasize);
29756 int posix_trace_attr_getmaxsystemeventsize(const trace_attr_t *attr,
29757 size_t *eventsize);
29758 int posix_trace_attr_getmaxusereventsize(const trace_attr_t *attr,
29759 size_t data_len, size_t *eventsize);
29760 int posix_trace_attr_getstreamsize(const trace_attr_t *attr,
29761 size_t *streamsize);
29762 TRC TRL int posix_trace_attr_setlogsize(trace_attr_t *attr,
29763 size_t logsize);
29764 TRC int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
29765 size_t maxdatasize);
29766 int posix_trace_attr_setstreamsize(trace_attr_t *attr,
29767 size_t streamsize);
29768

```

29769 **DESCRIPTION**

29770 TRL The *posix\_trace\_attr\_getlogsize()* function shall copy the log size, in bytes, from the *log-max-size*  
 29771 attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by  
 29772 the *logsize* argument. This log size is the maximum total of bytes that shall be allocated for  
 29773 system and user trace events in the trace log. The default value for the *log-max-size* attribute is  
 29774 implementation-defined.

29775 The *posix\_trace\_attr\_setlogsize()* function shall set the maximum allowed size, in bytes, in the  
 29776 *log-max-size* attribute of the attributes object pointed to by the *attr* argument, using the size value  
 29777 supplied by the *logsize* argument.

29778 The trace log size shall be used if the *log-full-policy* attribute is set to *POSIX\_TRACE\_LOOP* or  
 29779 *POSIX\_TRACE\_UNTIL\_FULL*. If the *log-full-policy* attribute is set to *POSIX\_TRACE\_APPEND*,  
 29780 the implementation shall ignore the *log-max-size* attribute.

29781 The *posix\_trace\_attr\_getmaxdatasize()* function shall copy the maximum user trace event data  
 29782 size, in bytes, from the *max-data-size* attribute of the attributes object pointed to by the *attr*  
 29783 argument into the variable pointed to by the *maxdatasize* argument. The default value for the  
 29784 *max-data-size* attribute is implementation-defined.

29785 The *posix\_trace\_attr\_getmaxsystemeventsize()* function calculates the maximum memory size, in  
 29786 bytes, required to store a single system trace event. This value is calculated for the trace stream  
 29787 attributes object pointed to by the *attr* argument and is returned in the variable pointed to by the  
 29788 *eventsize* argument.

29789 The values returned as the maximum memory sizes of the user and system trace events shall be  
 29790 such that if the sum of the maximum memory sizes of a set of the trace events that may be  
 29791 recorded in a trace stream is less than or equal to the *stream-min-size* attribute of that trace

29792 stream, the system provides the necessary resources for recording all those trace events, without  
29793 loss.

29794 The *posix\_trace\_attr\_getmaxusersize()* function calculates the maximum memory size, in  
29795 bytes, required to store a single user trace event generated by a call to *posix\_trace\_event()* with a  
29796 *data\_len* parameter equal to the *data\_len* value specified in this call. This value is calculated for  
29797 the trace stream attributes object pointed to by the *attr* argument and is returned in the variable  
29798 pointed to by the *eventsz* argument.

29799 The *posix\_trace\_attr\_getstreamsize()* function shall copy the stream size, in bytes, from the  
29800 *stream-min-size* attribute of the attributes object pointed to by the *attr* argument into the variable  
29801 pointed to by the *streamsize* argument.

29802 This stream size is the current total memory size reserved for system and user trace events in the  
29803 trace stream. The default value for the *stream-min-size* attribute is implementation-defined. The  
29804 stream size refers to memory used to store trace event records. Other stream data (for example,  
29805 trace attribute values) shall not be included in this size.

29806 The *posix\_trace\_attr\_setmaxdatasize()* function shall set the maximum allowed size, in bytes, in  
29807 the *max-data-size* attribute of the attributes object pointed to by the *attr* argument, using the size  
29808 value supplied by the *maxdatasize* argument. This maximum size is the maximum allowed size  
29809 for the user data argument which may be passed to *posix\_trace\_event()*. The implementation  
29810 shall be allowed to truncate data passed to *trace\_user\_event* which is longer than *maxdatasize*.

29811 The *posix\_trace\_attr\_setstreamsize()* function shall set the minimum allowed size, in bytes, in the  
29812 *stream-min-size* attribute of the attributes object pointed to by the *attr* argument, using the size  
29813 value supplied by the *streamsize* argument.

#### 29814 RETURN VALUE

29815 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
29816 return the corresponding error number.

29817 TRL The *posix\_trace\_attr\_getlogsize()* function stores the maximum trace log allowed size in the object  
29818 pointed to by *logsize*, if successful.

29819 The *posix\_trace\_attr\_getmaxdatasize()* function stores the maximum trace event record memory  
29820 size in the object pointed to by *maxdatasize*, if successful.

29821 The *posix\_trace\_attr\_getmaxsystemeventsz()* function stores the maximum memory size to store  
29822 a single system trace event in the object pointed to by *eventsz*, if successful.

29823 The *posix\_trace\_attr\_getmaxusersize()* function stores the maximum memory size to store a  
29824 single user trace event in the object pointed to by *eventsz*, if successful.

29825 The *posix\_trace\_attr\_getstreamsize()* function stores the maximum trace stream allowed size in  
29826 the object pointed to by *streamsize*, if successful.

#### 29827 ERRORS

29828 These functions may fail if:

29829 [EINVAL] The value specified by one of the arguments is invalid.

29830 **EXAMPLES**

29831           None.

29832 **APPLICATION USAGE**

29833           None.

29834 **RATIONALE**

29835           None.

29836 **FUTURE DIRECTIONS**

29837           None.

29838 **SEE ALSO**

29839           *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_event()*, *posix\_trace\_get\_attr()*, the Base

29840           Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>, <trace.h>

29841 **CHANGE HISTORY**

29842           First released in Issue 6. Derived from the IEEE Std. 1003.1q-2000.



29843 **NAME**

29844        posix\_trace\_attr\_init — trace stream attributes object initialization

29845 **SYNOPSIS**

29846 TRC     #include &lt;trace.h&gt;

29847        int posix\_trace\_attr\_init(trace\_attr\_t \*attr);

29848

29849 **DESCRIPTION**29850        Refer to *posix\_trace\_attr\_destroy()*.

29851 **NAME**

29852        posix\_trace\_clear — clear trace stream and trace log

29853 **SYNOPSIS**

29854 TRC     #include <sys/types.h>

29855        #include <trace.h>

29856        int posix\_trace\_clear(trace\_id\_t trid);

29857

29858 **DESCRIPTION**

29859        The *posix\_trace\_clear()* function shall reinitialize the trace stream identified by the argument *trid* as if it were returning from the *posix\_trace\_create()* function, except that the same allocated resources are reused, the mapping of trace event type identifiers to trace event names is unchanged, and the trace stream status remains unchanged (that is, if it was running, it remains running and if it was suspended, it remains suspended).

29864        All trace events in the trace stream recorded before the call to *posix\_trace\_clear()* are lost. The *posix\_stream\_full\_status* status shall be set to POSIX\_TRACE\_NOT\_FULL. There is no guarantee that all trace events that occurred during the *posix\_trace\_clear()* call are recorded; the behavior with respect to trace points that may occur during this call, is unspecified.

29868 TRL     If the Trace Log option is supported and the trace stream has been created with a log, the *posix\_trace\_clear()* function shall reinitialize the trace stream with the same behavior as if the trace stream was created without the log, plus it shall reinitialize the trace log associated with the trace stream identified by the argument *trid* as if it were returning from the *posix\_trace\_create\_withlog()* function, except that the same allocated resources, for the trace log, may be reused and the associated trace stream status remains unchanged. The first trace event recorded in the trace log after the call to *posix\_trace\_clear()* shall be the same as the first trace event recorded in the active trace stream after the call to *posix\_trace\_clear()*. The *posix\_log\_full\_status* status shall be set to POSIX\_TRACE\_NOT\_FULL. There is no guarantee that all trace events that occurred during the *posix\_trace\_clear()* call are recorded in the trace log; the behavior with respect to trace points that may occur during this call is unspecified. If the log full policy is POSIX\_TRACE\_APPEND, the effect of a call to this function is unspecified for the trace log associated with the trace stream identified by the *trid* argument.

29881 **RETURN VALUE**

29882        Upon successful completion, the *posix\_trace\_clear()* function shall return a value of zero.  
29883        Otherwise, it shall return the corresponding error number.

29884 **ERRORS**

29885        The *posix\_trace\_clear()* function may fail if:

29886        [EINVAL]        The value of the *trid* argument does not correspond to an active trace stream.

29887 **EXAMPLES**

29888        None.

29889 **APPLICATION USAGE**

29890        None.

29891 **RATIONALE**

29892        None.

29893 **FUTURE DIRECTIONS**

29894        None.

29895 **SEE ALSO**

29896 *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_flush()*, *posix\_trace\_get\_attr()*, the Base  
29897 Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>, <trace.h>

29898 **CHANGE HISTORY**

29899 First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

29900 **NAME**

29901 posix\_trace\_close, posix\_trace\_open, posix\_trace\_rewind — trace log management

29902 **SYNOPSIS**

29903 TRC TRL #include <trace.h>

```
29904 int posix_trace_close(trace_id_t trid);
29905 int posix_trace_open(int file_desc, trace_id_t *trid);
29906 int posix_trace_rewind(trace_id_t trid);
29907
```

29908 **DESCRIPTION**

29909 The *posix\_trace\_close()* function shall deallocate the trace log identifier indicated by *trid*, and all  
 29910 of its associated resources. If there is no valid trace log pointed to by the *trid*, this function shall  
 29911 fail.

29912 The *posix\_trace\_open()* function allocates the necessary resources and establishes the connection  
 29913 between a trace log identified by the *file\_desc* argument and a trace stream identifier identified by  
 29914 the object pointed to by the *trid* argument. The *file\_desc* argument should be a valid open file  
 29915 descriptor that corresponds to a trace log. The *file\_desc* argument shall be open for reading. The  
 29916 current trace event timestamp, which specifies the timestamp of the trace event that will be read  
 29917 by the next call to *posix\_trace\_getnext\_event()*, shall be set to the timestamp of the oldest trace  
 29918 event recorded in the trace log identified by *trid*.

29919 The *posix\_trace\_open()* function returns a trace stream identifier in the variable pointed to by the  
 29920 *trid* argument, that may only be used by the following functions:

|       |                                               |  |                                    |  |
|-------|-----------------------------------------------|--|------------------------------------|--|
| 29921 | <i>posix_trace_close()</i>                    |  | <i>posix_trace_get_attr()</i>      |  |
| 29922 | <i>posix_trace_eventid_equal()</i>            |  | <i>posix_trace_get_status()</i>    |  |
| 29923 | <i>posix_trace_eventid_get_name()</i>         |  | <i>posix_trace_getnext_event()</i> |  |
| 29924 | <i>posix_trace_eventtypelist_getnext_id()</i> |  | <i>posix_trace_rewind()</i>        |  |
| 29925 | <i>posix_trace_eventtypelist_rewind()</i>     |  |                                    |  |

29926 In particular, notice that the operations normally used by a trace controller process, such as  
 29927 *posix\_trace\_start()*, *posix\_trace\_stop()*, or *posix\_trace\_shutdown()*, cannot be invoked using the  
 29928 trace stream identifier returned by the *posix\_trace\_open()* function.

29929 The *posix\_trace\_rewind()* function shall reset the current trace event timestamp, which specifies  
 29930 the timestamp of the trace event that will be read by the next call to *posix\_trace\_getnext\_event()*,  
 29931 to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.

29932 **RETURN VALUE**

29933 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 29934 return the corresponding error number.

29935 If successful, the *posix\_trace\_open()* function stores the trace stream identifier value in the object  
 29936 pointed to by *trid*.

29937 **ERRORS**

29938 The *posix\_trace\_open()* function may fail if:

- 29939 [EINTR] The operation was interrupted by a signal and thus no trace log was opened.
- 29940 [EINVAL] The object pointed to by *file\_desc* does not correspond to a valid trace log.

29941 The *posix\_trace\_close()* and *posix\_trace\_rewind()* functions may fail if:

- 29942 [EINVAL] The object pointed to by *trid* does not correspond to a valid trace log.

29943 **EXAMPLES**

29944 None.

29945 **APPLICATION USAGE**

29946 None.

29947 **RATIONALE**

29948 None.

29949 **FUTURE DIRECTIONS**

29950 None.

29951 **SEE ALSO**

29952 *posix\_trace\_get\_attr()*, *posix\_trace\_get\_filter()*, *posix\_trace\_getnext\_event()*, the Base Definitions  
29953 volume of IEEE Std. 1003.1-200x, <trace.h>

29954 **CHANGE HISTORY**

29955 First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

29956 **NAME**

29957 posix\_trace\_create, posix\_trace\_create\_withlog, posix\_trace\_flush, posix\_trace\_shutdown —  
 29958 trace stream initialization, flush, and shutdown from a process

29959 **SYNOPSIS**

```

29960 TRC #include <sys/types.h>
29961 #include <trace.h>

29962 int posix_trace_create(pid_t pid, const trace_attr_t *attr,
29963 trace_id_t *trid);
29964 TRC TRL int posix_trace_create_withlog(pid_t pid, const trace_attr_t *attr,
29965 int file_desc, trace_id_t *trid);
29966 int posix_trace_flush(trace_id_t trid);
29967 int posix_trace_shutdown(trace_id_t trid);
29968

```

29969 **DESCRIPTION**

29970 The *posix\_trace\_create()* function creates an active trace stream. It allocates all the resources  
 29971 needed by the trace stream being created for tracing the process specified by *pid* in accordance  
 29972 with the *attr* argument. The *attr* argument represents the initial attributes of the trace stream and  
 29973 shall have been initialized by the function *posix\_trace\_attr\_init()* prior the *posix\_trace\_create()*  
 29974 call. If the argument *attr* is NULL, the default attributes shall be used. The *attr* attributes object  
 29975 shall be manipulated through a set of functions described in the *posix\_trace\_attr* family of  
 29976 functions. If the attributes of the object pointed to by *attr* are modified later, the attributes of the  
 29977 trace stream shall not be affected. The *creation-time* attribute of the newly created trace stream  
 29978 shall be set to the value of the system clock, if the Timers option is not supported, or to the value  
 29979 of the CLOCK\_REALTIME clock, if the Timers option is supported.

29980 The *pid* argument represents the target process to be traced. If the process executing this  
 29981 function does not have appropriate privileges to trace the process identified by *pid*, an error shall  
 29982 be returned. If the *pid* argument is zero, the calling process shall be traced.

29983 The *posix\_trace\_create()* function stores the trace stream identifier of the new trace stream in the  
 29984 object pointed to by the *trid* argument. This trace stream identifier shall be used in subsequent  
 29985 calls to control tracing. The *trid* argument may only be used by the following functions:

|       |                                               |  |                                         |  |
|-------|-----------------------------------------------|--|-----------------------------------------|--|
| 29986 | <i>posix_trace_clear()</i>                    |  | <i>posix_trace_getnext_event()</i>      |  |
| 29987 | <i>posix_trace_eventid_equal()</i>            |  | <i>posix_trace_shutdown()</i>           |  |
| 29988 | <i>posix_trace_eventid_get_name()</i>         |  | <i>posix_trace_start()</i>              |  |
| 29989 | <i>posix_trace_eventtypelist_getnext_id()</i> |  | <i>posix_trace_stop()</i>               |  |
| 29990 | <i>posix_trace_eventtypelist_rewind()</i>     |  | <i>posix_trace_timedgetnext_event()</i> |  |
| 29991 | <i>posix_trace_get_attr()</i>                 |  | <i>posix_trace_trid_eventid_open()</i>  |  |
| 29992 | <i>posix_trace_get_status()</i>               |  | <i>posix_trace_trygetnext_event()</i>   |  |

29993 TEF If the Trace Event Filter option is supported, the following additional functions may use the *trid*  
 29994 argument:

|       |                                 |  |                                 |  |
|-------|---------------------------------|--|---------------------------------|--|
| 29995 | <i>posix_trace_get_filter()</i> |  | <i>posix_trace_spt_filter()</i> |  |
|-------|---------------------------------|--|---------------------------------|--|

29996

29997 In particular, notice that the operations normally used by a trace analyser process, such as  
 29998 *posix\_trace\_rewind()* or *posix\_trace\_close()*, cannot be invoked using the trace stream identifier  
 29999 returned by the *posix\_trace\_create()* function.

- 30000 TEF A trace stream shall be created in a suspended state. If the Trace Event Filter option is supported, its trace event type filter shall be empty.
- 30001
- 30002 The *posix\_trace\_create()* function may be called multiple times from the same or different processes, with the system-wide limit indicated by the runtime invariant value {TRACE\_SYS\_MAX}, which has the minimum value {\_POSIX\_TRACE\_SYS\_MAX}.
- 30003
- 30004
- 30005 The trace stream identifier returned by the *posix\_trace\_create()* function in the argument pointed to by *trid* is valid only in the process that made the function call. If it is used from another process, that is a child process, in functions defined in IEEE Std. 1003.1-200x, these functions shall return with the error [EINVAL].
- 30006
- 30007
- 30008
- 30009 TRL The *posix\_trace\_create\_withlog()* function creates a trace stream as in the *posix\_trace\_create()* function and behaves the same way, plus it associates a trace log with this trace stream. The *file\_desc* argument shall be the file descriptor designating the trace log destination. The function shall fail if this file descriptor refers to a file with a file type that is not compatible with the log policy associated with the trace log. The list of the appropriate file types that are compatible with each log policy shall be implementation-defined.
- 30010
- 30011
- 30012
- 30013
- 30014
- 30015 The *posix\_trace\_create\_withlog()* function returns in the parameter pointed to by *trid* the trace stream identifier, which uniquely identifies the newly created trace stream, and shall be used in subsequent calls to control tracing. The *trid* argument may only be used by the following functions:
- 30016
- 30017
- 30018
- |       |                                               |                                         |  |
|-------|-----------------------------------------------|-----------------------------------------|--|
| 30019 | <i>posix_trace_clear()</i>                    | <i>posix_trace_getnext_event()</i>      |  |
| 30020 | <i>posix_trace_eventid_equal()</i>            | <i>posix_trace_shutdown()</i>           |  |
| 30021 | <i>posix_trace_eventid_get_name()</i>         | <i>posix_trace_start()</i>              |  |
| 30022 | <i>posix_trace_eventtypelist_getnext_id()</i> | <i>posix_trace_stop()</i>               |  |
| 30023 | <i>posix_trace_eventtypelist_rewind()</i>     | <i>posix_trace_timedgetnext_event()</i> |  |
| 30024 | <i>posix_trace_flush()</i>                    | <i>posix_trace_trid_eventid_open()</i>  |  |
| 30025 | <i>posix_trace_get_attr()</i>                 | <i>posix_trace_trygetnext_event()</i>   |  |
| 30026 | <i>posix_trace_get_status()</i>               |                                         |  |
- 30027
- 30028 TRL TEF If the Trace Event Filter option is supported, the following additional functions may use the *trid* argument:
- 30029
- |       |                                 |                                 |  |
|-------|---------------------------------|---------------------------------|--|
| 30030 | <i>posix_trace_get_filter()</i> | <i>posix_trace_spt_filter()</i> |  |
|-------|---------------------------------|---------------------------------|--|
- 30031 In particular, notice that the operations normally used by a trace analyser process, such as *posix\_trace\_rewind()* or *posix\_trace\_close()*, cannot be invoked using the trace stream identifier returned by the *posix\_trace\_create\_withlog()* function.
- 30032
- 30033
- 30034 The *posix\_trace\_flush()* function initiates a flush operation which copies the contents of the trace stream identified by the argument *trid* into the trace log associated with the trace stream at the creation time. If no trace log has been associated with the trace stream pointed to by *trid*, this function shall return an error. The termination of the flush operation can be polled by the *posix\_trace\_get\_status()* function. During the flush operation, it shall be possible to trace new trace events up to the point when the trace stream becomes full. After flushing is completed, the space used by the flushed trace events shall be available for tracing new trace events.
- 30035
- 30036
- 30037
- 30038
- 30039
- 30040
- 30041 If flushing the trace stream causes the resulting trace log to become full, the trace log full policy shall be applied. If the trace *log-full-policy* attribute is set, the following occurs:
- 30042

30043 POSIX\_TRACE\_UNTIL\_FULL  
 30044 The trace events that have not yet been flushed are discarded.

30045 POSIX\_TRACE\_LOOP  
 30046 The trace events that have not yet been flushed are written to the beginning of the trace log,  
 30047 overwriting previous trace events stored there.

30048 POSIX\_TRACE\_APPEND  
 30049 The trace events that had not yet been flushed shall be appended to the trace log.

30050 For an active trace stream with log, when the *posix\_trace\_shutdown()* function is called, all trace  
 30051 events that have not yet been flushed to the trace log shall be flushed, as in the  
 30052 *posix\_trace\_flush()* function, and the trace log shall be closed.

30053 When a trace log is closed, all the information that may be retrieved later from the trace log  
 30054 through the trace interface, shall have been written to the trace log. This information includes  
 30055 the trace attributes, the list of trace event types (with the mapping between trace event names  
 30056 and trace event type identifiers), and the trace status.

30057 In addition, some unspecified information shall be written to the trace log to allow detection of a  
 30058 valid trace log during the *posix\_trace\_open()* operation.

30059 The *posix\_trace\_shutdown()* function shall stop the tracing of trace events in the trace stream  
 30060 identified by *trid*, as if *posix\_trace\_stop()* had been invoked. The *posix\_trace\_shutdown()* function  
 30061 shall free all the resources associated with the trace stream.

30062 The *posix\_trace\_shutdown()* function shall not return until all the resources associated with the  
 30063 trace stream have been freed. When the *posix\_trace\_shutdown()* function returns, the *trid*  
 30064 argument becomes an invalid trace stream identifier. A call to this function shall unconditionally  
 30065 deallocate the resources regardless of whether all trace events have been retrieved by the  
 30066 analyzer process. Any thread blocked on one of the *trace\_getnext\_event()* functions (which  
 30067 specified this *trid*) before this call is unblocked with the error [EINVAL].

30068 If the process exits, invokes an *exec()* call, or is terminated, the trace streams that the process had  
 30069 created and that have not yet been shut down, shall be automatically shut down as if an explicit  
 30070 call were made to the *posix\_trace\_shutdown()* function.

30071 The *posix\_trace\_shutdown()* function shall not return until all trace events have been flushed.

30072 **RETURN VALUE**

30073 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 30074 return the corresponding error number.

30075 TRL The *posix\_trace\_create()* and *posix\_trace\_create\_withlog()* functions store the trace stream identifier  
 30076 value in the object pointed to by *trid*, if successful.

30077 **ERRORS**

30078 TRL The *posix\_trace\_create()* and *posix\_trace\_create\_withlog()* functions shall fail if:

30079 [EAGAIN] No more trace streams can be started now. {TRACE\_SYS\_MAX} has been  
 30080 exceeded.

30081 [EINTR] The operation was interrupted by a signal. No trace stream was created.

30082 [EINVAL] One or more of the trace parameters specified by the *attr* parameter is invalid.

30083 [ENOMEM] The implementation does not currently have sufficient memory to create the  
 30084 trace stream with the specified parameters.

30085 [EPERM] The caller does not have appropriate privilege to trace the process specified by  
 30086 *pid*.



|       |          |                                                                                                                                                  |
|-------|----------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| 30087 | [ESRCH]  | The <i>pid</i> argument does not refer to an existing process.                                                                                   |
| 30088 | TRL      | The <i>posix_trace_create_withlog()</i> function shall fail if:                                                                                  |
| 30089 | [EBADF]  | The <i>file_desc</i> argument is not a valid file descriptor open for writing.                                                                   |
| 30090 | [EINVAL] | The <i>file_desc</i> argument refers to a file with a file type that does not support the log policy associated with the trace log.              |
| 30091 |          |                                                                                                                                                  |
| 30092 | [ENOSPC] | No space left on device. The device corresponding to the argument <i>file_desc</i> does not contain the space required to create this trace log. |
| 30093 |          |                                                                                                                                                  |
| 30094 |          |                                                                                                                                                  |
| 30095 | TRL      | The <i>posix_trace_flush()</i> and <i>posix_trace_shutdown()</i> functions shall fail if:                                                        |
| 30096 | [EINVAL] | The value of the <i>trid</i> argument does not correspond to an active trace stream with log.                                                    |
| 30097 |          |                                                                                                                                                  |
| 30098 | [EFBIG]  | The trace log file has attempted to exceed an implementation-defined maximum file size.                                                          |
| 30099 |          |                                                                                                                                                  |
| 30100 | [ENOSPC] | No space left on device.                                                                                                                         |
| 30101 |          |                                                                                                                                                  |
| 30102 |          | <b>EXAMPLES</b>                                                                                                                                  |
| 30103 |          | None.                                                                                                                                            |
| 30104 |          | <b>APPLICATION USAGE</b>                                                                                                                         |
| 30105 |          | None.                                                                                                                                            |
| 30106 |          | <b>RATIONALE</b>                                                                                                                                 |
| 30107 |          | None.                                                                                                                                            |
| 30108 |          | <b>FUTURE DIRECTIONS</b>                                                                                                                         |
| 30109 |          | None.                                                                                                                                            |
| 30110 |          | <b>SEE ALSO</b>                                                                                                                                  |
| 30111 |          | <i>clock_getres()</i> , <i>exec</i> , <i>posix_trace_attr_init()</i> , <i>posix_trace_clear()</i> , <i>posix_trace_close()</i> ,                 |
| 30112 |          | <i>posix_trace_eventid_equal()</i> , <i>posix_trace_eventtypelist_getnext_id()</i> , <i>posix_trace_flush()</i> ,                                |
| 30113 |          | <i>posix_trace_get_attr()</i> , <i>posix_trace_get_filter()</i> , <i>posix_trace_get_status()</i> , <i>posix_trace_getnext_event()</i> ,         |
| 30114 |          | <i>posix_trace_open()</i> , <i>posix_trace_rewind()</i> , <i>posix_trace_set_filter()</i> , <i>posix_trace_shutdown()</i> ,                      |
| 30115 |          | <i>posix_trace_start()</i> , <i>posix_trace_timedgetnext_event()</i> , <i>posix_trace_trid_eventid_open()</i> ,                                  |
| 30116 |          | <i>posix_trace_start()</i> , <i>time()</i> , the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>,                                |
| 30117 |          | <trace.h>                                                                                                                                        |
| 30118 |          | <b>CHANGE HISTORY</b>                                                                                                                            |
| 30119 |          | First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.                                                                                  |

30120 **NAME**

30121 posix\_trace\_event, posix\_trace\_eventid\_open — trace functions for instrumenting application  
 30122 code

30123 **SYNOPSIS**

```
30124 TRC #include <sys/types.h>
30125 #include <trace.h>

30126 void posix_trace_event(trace_event_id_t event_id, const void *data_ptr,
30127 size_t data_len);
30128 int posix_trace_eventid_open(const char *event_name,
30129 trace_event_id_t *event_id);
30130
```

30131 **DESCRIPTION**

30132 The *posix\_trace\_event()* function records the *event\_id* and the user data pointed to by *data\_ptr* in  
 30133 the trace stream into which the calling process is being traced and in which *event\_id* is not  
 30134 filtered out. If the total size of the user trace event data represented by *data\_len* is not greater  
 30135 than the declared maximum size for user trace event data, then the *truncation-status* attribute of  
 30136 the trace event recorded is POSIX\_TRACE\_NOT\_TRUNCATED. Otherwise, the user trace event  
 30137 data is truncated to this declared maximum size and the *truncation-status* attribute of the trace  
 30138 event recorded is POSIX\_TRACE\_TRUNCATED\_RECORD.

30139 If there is no trace stream created for the process or if the created trace stream is not running or if  
 30140 the trace event specified by *event\_id* is filtered out in the trace stream, the *posix\_trace\_event()*  
 30141 function has no effect.

30142 The *posix\_trace\_eventid\_open()* function is used to associate a user trace event name with a trace  
 30143 event type identifier for the calling process. The trace event name is the string pointed to by the  
 30144 argument *event\_name*. It shall have a maximum of {TRACE\_EVENT\_NAME\_MAX} characters  
 30145 (which has the minimum value {\_POSIX\_TRACE\_EVENT\_NAME\_MAX}). The number of user  
 30146 trace event type identifiers that can be defined for any given process is limited by the maximum  
 30147 value {TRACE\_USER\_EVENT\_MAX}, which has the minimum value  
 30148 {POSIX\_TRACE\_USER\_EVENT\_MAX}.

30149 If the Trace Inheritance option is not supported, the *posix\_trace\_trid\_eventid\_open()* function shall  
 30150 associate the user trace event name pointed to by the *event\_name* argument with a trace event  
 30151 type identifier that is unique for the traced process, and is returned in the variable pointed to by  
 30152 the *event* argument. If the user trace event name has already been mapped for the traced process,  
 30153 then the previously assigned trace event type identifier shall be returned. If the per-process user  
 30154 trace event name limit represented by {TRACE\_USER\_EVENT\_MAX} has been reached, the  
 30155 pre-defined POSIX\_TRACE\_UNNAMED\_USEREVENT (see Table 2-10 (on page 582)) user trace  
 30156 event shall be returned.

30157 TRI If the Trace Inherit option is supported, the *posix\_trace\_trid\_eventid\_open()* function shall  
 30158 associate the user trace event name pointed to by the *event\_name* argument with a trace event  
 30159 type identifier that is unique for all the processes being traced in this same trace stream, and is  
 30160 returned in the variable pointed to by the *event* argument. If the user trace event name has  
 30161 already been mapped for the traced processes, then the previously assigned trace event type  
 30162 identifier shall be returned. If the per-process user trace event name limit represented by  
 30163 {TRACE\_USER\_EVENT\_MAX} has been reached, the pre-defined  
 30164 POSIX\_TRACE\_UNNAMED\_USEREVENT (Table 2-10 (on page 582)) user trace event shall be  
 30165 returned.

30166 **Note:** The above procedure, together with the fact that multiple processes can only be  
 30167 traced into the same trace stream by inheritance, ensure that all the processes that are

30168                   traced into a trace stream have the same mapping of trace event names to trace event  
30169                   type identifiers.

30170

30171                   If there is no trace stream created, the *posix\_trace\_eventid\_open()* function shall store this  
30172                   information for future trace streams created for this process.

30173 **RETURN VALUE**

30174                   No return value is defined for the *posix\_trace\_event()* function.

30175                   Upon successful completion, the *posix\_trace\_eventid\_open()* function shall return a value of zero.  
30176                   Otherwise, it shall return the corresponding error number. The *posix\_trace\_eventid\_open()*  
30177                   function stores the trace event type identifier value in the object pointed to by *event\_id*, if  
30178                   successful.

30179 **ERRORS**

30180                   The *posix\_trace\_eventid\_open()* function may fail if:

30181                   [ENAMETOOLONG]

30182                                           The size of the name pointed to by *event\_name* argument was longer than the  
30183                                           implementation-defined value {TRACE\_EVENT\_NAME\_MAX}.

30184 **EXAMPLES**

30185                   None.

30186 **APPLICATION USAGE**

30187                   None.

30188 **RATIONALE**

30189                   None.

30190 **FUTURE DIRECTIONS**

30191                   None.

30192 **SEE ALSO**

30193                   *posix\_trace\_start()*, *posix\_trace\_trid\_eventid\_open()*, the Base Definitions volume of  
30194                   IEEE Std. 1003.1-200x, <sys/types.h>, <trace.h>

30195 **CHANGE HISTORY**

30196                   First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

30197 **NAME**

30198 posix\_trace\_eventid\_equal, posix\_trace\_eventid\_get\_name, posix\_trace\_trid\_eventid\_open —  
 30199 manipulate trace event type identifier

30200 **SYNOPSIS**

```
30201 TRC #include <trace.h>
30202
30202 int posix_trace_eventid_equal(trace_id_t trid, trace_eventid_t event1,
30203 trace_eventid_t event2);
30204 int posix_trace_eventid_get_name(trace_id_t trid, trace_eventid_t event,
30205 char *event_name);
30206 TRC TEF int posix_trace_trid_eventid_open(trace_id_t trid,
30207 const char *event_name, trace_eventid_t *event);
30208
```

30209 **DESCRIPTION**

30210 The *posix\_trace\_eventid\_equal()* function compares the trace event type identifiers *event1* and  
 30211 *event2* from the same trace stream or the same trace log identified by the *trid* argument. If the  
 30212 trace event type identifiers *event1* and *event2* are from different trace streams, the return value  
 30213 shall be unspecified.

30214 The *posix\_trace\_eventid\_get\_name()* function returns in the argument pointed to by *event\_name*,  
 30215 the trace event name associated with the trace event type identifier identified by the argument  
 30216 *event*, for the trace stream or for the trace log identified by the *trid* argument. The name of the  
 30217 trace event shall have a maximum of {TRACE\_EVENT\_NAME\_MAX} characters (which has the  
 30218 minimum value {\_POSIX\_TRACE\_EVENT\_NAME\_MAX}). Successive calls to this function  
 30219 with the same trace event type identifier and the same trace stream identifier shall return the  
 30220 same event name.

30221 TEF The *posix\_trace\_trid\_eventid\_open()* function is used to associate a user trace event name with a  
 30222 trace event type identifier for a given trace stream. The trace stream is identified by the *trid*  
 30223 argument, and it shall be an active trace stream. The trace event name is the string pointed to by  
 30224 the argument *event\_name*. It shall have a maximum of {TRACE\_EVENT\_NAME\_MAX}  
 30225 characters (which has the minimum value {\_POSIX\_TRACE\_EVENT\_NAME\_MAX}). The  
 30226 number of user trace event type identifiers that can be defined for any given process is limited  
 30227 by the maximum value {TRACE\_USER\_EVENT\_MAX}, which has the minimum value  
 30228 {\_POSIX\_TRACE\_USER\_EVENT\_MAX}.

30229 If the Trace Inheritance option is not supported, the *posix\_trace\_trid\_eventid\_open()* function shall  
 30230 associate the user trace event name pointed to by the *event\_name* argument with a trace event  
 30231 type identifier that is unique for the process being traced in the trace stream identified by the *trid*  
 30232 argument, and is returned in the variable pointed to by the *event* argument. If the user trace  
 30233 event name has already been mapped for the traced process, then the previously assigned trace  
 30234 event type identifier shall be returned. If the per-process user trace event name limit represented  
 30235 by {TRACE\_USER\_EVENT\_MAX} has been reached, the pre-defined  
 30236 POSIX\_TRACE\_UNNAMED\_USEREVENT (see Table 2-10 (on page 582)) user trace event shall  
 30237 be returned.

30238 TEF TRI If the Trace Inheritance option is supported, the *posix\_trace\_trid\_eventid\_open()* function shall  
 30239 associate the user trace event name pointed to by the *event\_name* argument with a trace event  
 30240 type identifier that is unique for all the processes being traced in the trace stream identified by  
 30241 the *trid* argument, and is returned in the variable pointed to by the *event* argument. If the user  
 30242 trace event name has already been mapped for the traced processes, then the previously  
 30243 assigned trace event type identifier shall be returned. If the per-process user trace event name  
 30244 limit represented by {TRACE\_USER\_EVENT\_MAX} has been reached, the pre-defined  
 30245 POSIX\_TRACE\_UNNAMED\_USEREVENT (see Table 2-10 (on page 582)) user trace event shall

30246 be returned.

### 30247 RETURN VALUE

30248 TEF Upon successful completion, the `posix_trace_eventid_get_name()` and  
 30249 `posix_trace_trid_eventid_open()` functions shall return a value of zero. Otherwise, they shall return  
 30250 the corresponding error number.

30251 The `posix_trace_eventid_equal()` function shall return a non-zero value if *event1* and *event2* are  
 30252 equal; otherwise, a value of zero shall be returned. No errors are defined. If either *event1* or  
 30253 *event2* are not valid trace event type identifiers for the trace stream specified by *trid* or if the *trid*  
 30254 is invalid, the behavior shall be unspecified.

30255 The `posix_trace_eventid_get_name()` function stores the trace event name value in the object  
 30256 pointed to by *event\_name*, if successful.

30257 TEF The `posix_trace_trid_eventid_open()` function stores the trace event type identifier value in the  
 30258 object pointed to by *event*, if successful.

### 30259 ERRORS

30260 TEF The `posix_trace_eventid_get_name()` and `posix_trace_trid_eventid_open()` functions may fail if:

30261 [EINVAL] The *trid* argument was not a valid trace type identifier.

30262 TEF The `posix_trace_trid_eventid_open()` function may fail if:

30263 [ENAMETOOLONG]

30264 The size of the name pointed to by *event\_name* argument was longer than the  
 30265 implementation-defined value {TRACE\_EVENT\_NAME\_MAX}.  
 30266

30267 The `posix_trace_eventid_get_name()` function may fail if:

30268 [EINVAL] The trace event type identifier *event* was not associated with any name.

### 30269 EXAMPLES

30270 None.

### 30271 APPLICATION USAGE

30272 None.

### 30273 RATIONALE

30274 None.

### 30275 FUTURE DIRECTIONS

30276 None.

### 30277 SEE ALSO

30278 `posix_trace_event()`, <REFERENCE UNDEFINED>(posix\_trace\_eventid),  
 30279 `posix_trace_getnext_event()`, the Base Definitions volume of IEEE Std. 1003.1-200x, <trace.h>

### 30280 CHANGE HISTORY

30281 First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

## 30282 NAME

30283 `posix_trace_eventset_add`, `posix_trace_eventset_del`, `posix_trace_eventset_empty`,  
 30284 `posix_trace_eventset_fill`, `posix_trace_eventset_ismember` — manipulate trace event type sets

## 30285 SYNOPSIS

```
30286 TRC TEF #include <trace.h>

30287 int posix_trace_eventset_add(trace_event_id_t event_id,
30288 trace_event_set_t *set);
30289 int posix_trace_eventset_del(trace_event_id_t event_id,
30290 trace_event_set_t *set);
30291 int posix_trace_eventset_empty(trace_event_set_t *set);
30292 int posix_trace_eventset_fill(trace_event_set_t *set, int what);
30293 int posix_trace_eventset_ismember(trace_event_id_t event_id,
30294 const trace_event_set_t *set, int *ismember);
30295
```

## 30296 DESCRIPTION

30297 These primitives manipulate sets of trace event types. They operate on data objects addressable  
 30298 by the application, not on the current trace event filter of any trace stream.

30299 The `posix_trace_eventset_add()` and `posix_trace_eventset_del()` functions, respectively, add or  
 30300 delete the individual trace event type specified by the value of the argument `event_id` to or from  
 30301 the trace event type set pointed to by the argument `set`. Adding a trace event type already in the  
 30302 set or deleting a trace event type not in the set shall not be considered an error.

30303 The `posix_trace_eventset_empty()` function initializes the trace event type set pointed to by the `set`  
 30304 argument such that all trace event types defined, both system and user, shall be excluded from  
 30305 the set.

30306 The `posix_trace_eventset_fill()` function initializes the trace event type set pointed to by the  
 30307 argument `set`, such that the set of trace event types defined by the argument `what` shall be  
 30308 included in the set. The value of the argument `what` shall consist of one of the following values,  
 30309 as defined in the `<trace.h>` header:

30310 **POSIX\_TRACE\_WOPID\_EVENTS**

30311 All the process-independent implementation-defined system trace event types are included  
 30312 in the set.

30313 **POSIX\_TRACE\_SYSTEM\_EVENTS** All the implementation-defined system trace event types are  
 30314 included in the set, as are those defined in IEEE Std. 1003.1-200x.

30315 **POSIX\_TRACE\_ALL\_EVENTS** All trace event types defined, both system and user, are included  
 30316 in the set.

30317 Applications shall call either `posix_trace_eventset_empty()` or `posix_trace_eventset_fill()` at least  
 30318 once for each object of type `trace_event_set_t` prior to any other use of that object. If such an  
 30319 object is not initialized in this way, but is nonetheless supplied as an argument to any of the  
 30320 `posix_trace_eventset_add()`, `posix_trace_eventset_del()`, or `posix_trace_eventset_ismember()` functions,  
 30321 the results are undefined.

30322 The `posix_trace_eventset_ismember()` function tests whether the trace event type specified by the  
 30323 value of the argument `event_id` is a member of the set pointed to by the argument `set`. The value  
 30324 returned in the object pointed to by `ismember` argument is zero if the trace event type identifier is  
 30325 not a member of the set and a value different from zero if it is a member of the set.

**30326 RETURN VALUE**

30327       Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
30328       return the corresponding error number.

**30329 ERRORS**

30330       These functions may fail if:

30331       [EINVAL]       The value of one of the arguments is invalid.

**30332 EXAMPLES**

30333       None.

**30334 APPLICATION USAGE**

30335       None.

**30336 RATIONALE**

30337       None.

**30338 FUTURE DIRECTIONS**

30339       None.

**30340 SEE ALSO**

30341       *posix\_trace\_set\_filter()*, *posix\_trace\_trid\_eventid\_open()*, the Base Definitions volume of  
30342       IEEE Std. 1003.1-200x, <trace.h>

**30343 CHANGE HISTORY**

30344       First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

30345 **NAME**

30346 posix\_trace\_eventtypelist\_getnext\_id, posix\_trace\_eventtypelist\_rewind — iterate over a  
30347 mapping of trace event types

30348 **SYNOPSIS**

```
30349 TRC #include <trace.h>
30350 int posix_trace_eventtypelist_getnext_id(trace_id_t trid,
30351 trace_eventid_t *event, int *unavailable);
30352 int posix_trace_eventtypelist_rewind(trace_id_t trid);
30353
```

30354 **DESCRIPTION**

30355 The first time *posix\_trace\_eventtypelist\_getnext\_id()* is called, the function shall return in the  
30356 variable pointed to by *event* the first trace event type identifier of the list of trace events of the  
30357 trace stream identified by the *trid* argument. Successive calls to  
30358 *posix\_trace\_eventtypelist\_getnext\_id()* return in the variable pointed to by *event* the next trace  
30359 event type identifier in that same list. Each time a trace event type identifier is successfully  
30360 written into the variable pointed to by the *event* argument, the variable pointed to by the  
30361 *unavailable* argument shall be set to zero. When no more trace event type identifiers are  
30362 available, and so none is returned, the variable pointed to by the *unavailable* argument shall be  
30363 set to a value different from zero.

30364 The *posix\_trace\_eventtypelist\_rewind()* function shall reset the next trace event type identifier to  
30365 be read to the first trace event type identifier from the list of trace events used in the trace stream  
30366 identified by *trid*.

30367 **RETURN VALUE**

30368 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
30369 return the corresponding error number.

30370 The *posix\_trace\_eventtypelist\_getnext\_id()* function stores the trace event type identifier value in  
30371 the object pointed to by *event*, if successful.

30372 **ERRORS**

30373 These functions may fail if:

30374 [EINVAL] The *trid* argument was not a valid trace stream identifier.

30375 **EXAMPLES**

30376 None.

30377 **APPLICATION USAGE**

30378 None.

30379 **RATIONALE**

30380 None.

30381 **FUTURE DIRECTIONS**

30382 None.

30383 **SEE ALSO**

30384 *posix\_trace\_event()*, <REFERENCE UNDEFINED>(posix\_trace\_eventid),  
30385 *posix\_trace\_getnext\_event()*, *posix\_trace\_trid\_eventid\_open()*, the Base Definitions volume of  
30386 IEEE Std. 1003.1-200x, <trace.h>



30387 **CHANGE HISTORY**

30388 First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

30389 **NAME**

30390        posix\_trace\_flush — trace stream flush from a process

30391 **SYNOPSIS**

30392 TRC     #include <sys/types.h>

30393        #include <trace.h>

30394        int posix\_trace\_flush(trace\_id\_t trid);

30395

30396 **DESCRIPTION**

30397        Refer to *posix\_trace\_create()*.

30398 **NAME**

30399        posix\_trace\_get\_attr, posix\_trace\_get\_status — retrieve the trace attributes or trace statuses

30400 **SYNOPSIS**

30401 TRC     #include &lt;trace.h&gt;

30402        int posix\_trace\_get\_attr(trace\_id\_t trid, trace\_attr\_t \*attr);

30403        int posix\_trace\_get\_status(trace\_id\_t trid,

30404            struct posix\_trace\_status\_info \*statusinfo);

30405

30406 **DESCRIPTION**30407        The *posix\_trace\_get\_attr()* function shall copy the attributes of the active trace stream identified  
30408 TRL     by *trid* into the object pointed to by the *attr* argument. If the Trace Log option is supported, *trid*  
30409     may represent a pre-recorded trace log.30410        The *posix\_trace\_get\_status()* function returns, in the structure pointed to by the *statusinfo*  
30411     argument, the current trace status for the trace stream identified by the *trid* argument. These  
30412     status values returned in the structure pointed to by *statusinfo* shall have been appropriately  
30413 TRL     read to ensure that the returned values are consistent. If the Trace Log option is supported and  
30414     the *trid* argument refers to a pre-recorded trace stream, the status shall be the status of the  
30415     completed trace stream.30416        Each time the *posix\_trace\_get\_status()* function is used, the overrun status of the trace stream  
30417 TRL     shall be reset to POSIX\_TRACE\_NO\_OVERRUN immediately after the call completes. If the  
30418     Trace Log option is supported, the *posix\_trace\_get\_status()* function shall behave the same as  
30419     when the option is not supported except for the following differences:30420        

- If the *trid* argument refers to a trace stream with log, each time the *posix\_trace\_get\_status()*  
30421     function is used, the log overrun status of the trace stream shall be reset to  
30422     POSIX\_TRACE\_NO\_OVERRUN and the *flush\_error* status shall be reset to zero immediately  
30423     after the call completes.

30424        

- If the *trid* argument refers to a pre-recorded trace stream, the status returned shall be the  
30425     status of the completed trace stream and the status values of the trace stream shall not be  
30426     reset.

30427

30428 **RETURN VALUE**30429        Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
30430     return the corresponding error number.30431        The *posix\_trace\_get\_attr()* function stores the trace attributes in the object pointed to by *attr*, if  
30432     successful.30433        The *posix\_trace\_get\_status()* function stores the trace status in the object pointed to by *statusinfo*,  
30434     if successful.30435 **ERRORS**

30436        These functions may fail if:

30437        [EINVAL]        The trace stream argument *trid* does not correspond to a valid active trace  
30438     stream or a valid trace log.

30439 **EXAMPLES**

30440           None.

30441 **APPLICATION USAGE**

30442           None.

30443 **RATIONALE**

30444           None.

30445 **FUTURE DIRECTIONS**

30446           None.

30447 **SEE ALSO**

30448           *posix\_trace\_attr\_destroy()*, *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_open()*, the Base

30449           Definitions volume of IEEE Std. 1003.1-200x, <**trace.h**>

30450 **CHANGE HISTORY**

30451           First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

30452 **NAME**

30453 posix\_trace\_get\_filter, posix\_trace\_set\_filter — retrieve and set filter of an initialized trace  
 30454 stream

30455 **SYNOPSIS**

30456 TRC TEF #include <trace.h>

30457 int posix\_trace\_get\_filter(trace\_id\_t *trid*, trace\_event\_set\_t \**set*);

30458 int posix\_trace\_set\_filter(trace\_id\_t *trid*,

30459 const trace\_event\_set\_t \**set*, int *how*);

30460

30461 **DESCRIPTION**

30462 The *posix\_trace\_get\_filter()* function shall be used to retrieve, into the argument pointed to by *set*,  
 30463 the actual trace event filter from the trace stream specified by *trid*.

30464 The *posix\_trace\_set\_filter()* function shall be used to change the set of filtered trace event types  
 30465 after a trace stream identified by the *trid* argument is created. This function may be called prior  
 30466 to starting the trace stream, or while the trace stream is active. By default, if no call is made to  
 30467 *posix\_trace\_set\_filter()*, all trace events shall be recorded (that is, none of the trace event types are  
 30468 filtered out).

30469 If this function is called while the trace is in progress, a special system trace event,  
 30470 POSIX\_TRACE\_FILTER, shall be recorded in the trace indicating both the old and the new sets  
 30471 of filtered trace event types (see Table 2-7 (on page 580) and Table 2-9 (on page 581)).

30472 If the *posix\_trace\_set\_filter()* function is interrupted by a signal, an error is returned and the filter  
 30473 is not changed. In this case, the state of the trace stream shall not be changed.

30474 The value of the argument *how* indicates the manner in which the set is to be changed and shall  
 30475 have one of the following values, as defined in the <trace.h> header:

30476 POSIX\_TRACE\_SET\_EVENTSET

30477 The resulting set of trace event types to be filtered shall be the trace event type set pointed  
 30478 to by the argument *set*.

30479 POSIX\_TRACE\_ADD\_EVENTSET

30480 The resulting set of trace event types to be filtered shall be the union of the current set and  
 30481 the trace event type set pointed to by the argument *set*.

30482 POSIX\_TRACE\_SUB\_EVENTSET

30483 The resulting set of trace event types to be filtered shall be all trace event types in the  
 30484 current set that are not in the set pointed to by the argument *set*; that is, remove each  
 30485 element of the specified set from the current filter.

30486 **RETURN VALUE**

30487 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 30488 return the corresponding error number.

30489 The *posix\_trace\_get\_filter()* function stores the set of filtered trace event types in *set*, if successful.

30490 **ERRORS**

30491 These functions may fail if:

30492 [EINVAL] The value of the *trid* argument does not correspond to an active trace stream  
 30493 or the value of the argument pointed to by *set* is invalid.

30494 [EINTR] The operation was interrupted by a signal.

30495 **EXAMPLES**

30496           None.

30497 **APPLICATION USAGE**

30498           None.

30499 **RATIONALE**

30500           None.

30501 **FUTURE DIRECTIONS**

30502           None.

30503 **SEE ALSO**

30504           *posix\_trace\_eventset\_add()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**trace.h**>

30505 **CHANGE HISTORY**

30506           First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

30507 **NAME**

30508        posix\_trace\_get\_status — retrieve the trace statuses

30509 **SYNOPSIS**

30510 TRC     #include &lt;trace.h&gt;

30511        int posix\_trace\_get\_status(trace\_id\_t *trid*,  
30512            struct posix\_trace\_status\_info \**statusinfo*);

30513

30514 **DESCRIPTION**30515        Refer to *posix\_trace\_get\_attr()*.

30516 **NAME**

30517 posix\_trace\_getnext\_event, posix\_trace\_timedgetnext\_event, posix\_trace\_trygetnext\_event —  
 30518 retrieve a trace event

30519 **SYNOPSIS**

```

30520 TRC #include <sys/types.h>
30521 #include <trace.h>

30522 int posix_trace_getnext_event(trace_id_t trid,
30523 struct posix_trace_event_info *event, void *data,
30524 size_t num_bytes, size_t *data_len, int *unavailable);
30525 TRC TMO int posix_trace_timedgetnext_event(trace_id_t trid,
30526 struct posix_trace_event_info *event, void *data,
30527 size_t num_bytes, size_t *data_len, int *unavailable,
30528 const struct timespec *abs_timeout);
30529 int posix_trace_trygetnext_event(trace_id_t trid,
30530 struct posix_trace_event_info *event, void *data,
30531 size_t num_bytes, size_t *data_len, int *unavailable);
30532

```

30533 **DESCRIPTION**

30534 The *posix\_trace\_getnext\_event()* function shall be used to report a recorded trace event either  
 30535 TRL from an active trace stream without log or a pre-recorded trace stream identified by the *trid*  
 30536 argument. The *posix\_trace\_trygetnext\_event()* function shall be used to report a recorded trace  
 30537 event from an active trace stream without log identified by the *trid* argument.

30538 The trace event information associated with the recorded trace event shall be copied by the  
 30539 function into the structure pointed to by the argument *event* and the data associated with the  
 30540 trace event shall be copied into the buffer pointed to by the *data* argument.

30541 The *posix\_trace\_getnext\_event()* function shall block if the *trid* argument identifies an active trace  
 30542 stream and there is currently no trace event ready to be retrieved. When returning, if a recorded  
 30543 trace event was reported, the variable pointed to by the *unavailable* argument shall be set to zero.  
 30544 Otherwise, the variable pointed to by the *unavailable* argument shall be set to a value different  
 30545 from zero.

30546 TMO The *posix\_trace\_timedgetnext\_event()* function attempts to get another trace event from an active  
 30547 trace stream without log, as in the *posix\_trace\_getnext\_event()* function. However, if no trace  
 30548 event is available from the trace stream, this wait shall be terminated when the timeout specified  
 30549 by the argument *abs\_timeout* expires, and the function shall return the error [ETIMEDOUT].

30550 The timeout expires when the absolute time specified by *abs\_timeout* passes, as measured by the  
 30551 clock upon which timeouts are based (that is, when the value of that clock equals or exceeds  
 30552 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already passed at the time of the  
 30553 call.

30554 TMO TMR If the Timers option is supported, the timeout is based on the CLOCK\_REALTIME clock; if the  
 30555 Timers option is not supported, the timeout is based on the system clock as returned by the  
 30556 *time()* function. The resolution of the timeout is the resolution of the clock on which it is based.  
 30557 The **timespec** data type is defined as a structure in the header **<time.h>**.

30558 Under no circumstance will the function fail with a timeout if a trace event is immediately  
 30559 available from the trace stream. The validity of the *abs\_timeout* argument need not be checked if  
 30560 a trace event is immediately available from the trace stream.

30561 TMO TMR The behavior of this function for a pre-recorded trace stream is unspecified.



30562 TRL The *posix\_trace\_trygetnext\_event()* function shall not block. This function shall return an error if  
 30563 the *trid* argument identifies a pre-recorded trace stream. If a recorded trace event was reported,  
 30564 the variable pointed to by the *unavailable* argument shall be set to zero. Otherwise, if no trace  
 30565 event was reported, the variable pointed to by the *unavailable* argument shall be set to a value  
 30566 different from zero.

30567 The argument *num\_bytes* shall be the size of the buffer pointed to by the *data* argument. The  
 30568 argument *data\_len* reports to the application the length in bytes of the data record just  
 30569 transferred. If *num\_bytes* is greater than or equal to the size of the data associated with the trace  
 30570 event pointed to by the *event* argument, all the recorded data shall be transferred. In this case, the  
 30571 *truncation-status* member of the trace event structure shall be either  
 30572 POSIX\_TRACE\_NOT\_TRUNCATED, if the trace event data was recorded without truncation  
 30573 while tracing, or POSIX\_TRACE\_TRUNCATED\_RECORD, if the trace event data was truncated  
 30574 when it was recorded. If the *num\_bytes* argument is less than the length of recorded trace event  
 30575 data, the data transferred shall be truncated to a length of *num\_bytes*, the value stored in the  
 30576 variable pointed to by *data\_len* shall be equal to *num\_bytes*, and the *truncation-status* member of  
 30577 the *event* structure argument shall be set to POSIX\_TRACE\_TRUNCATED\_READ (see the  
 30578 *posix\_trace\_event\_info()* function).

30579 The report of a trace event shall be sequential starting from the oldest recorded trace event. Trace  
 30580 events shall be reported in the order in which they were generated, up to an implementation-  
 30581 defined time resolution that causes the ordering of trace events occurring very close to each  
 30582 other to be unknown. Once reported, a trace event cannot be reported again from an active trace  
 30583 stream. Once a trace event is reported from an active trace stream without log, the trace stream  
 30584 shall make the resources associated with that trace event available to record future generated  
 30585 trace events.

#### 30586 RETURN VALUE

30587 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 30588 return the corresponding error number.

30589 If successful, these functions store:

- 30590 • The recorded trace event in the object pointed to by *event*
- 30591 • The trace event information associated with the recorded trace event in the object pointed to  
 30592 by *data*
- 30593 • The length of this trace event information in the object pointed to by *data\_len*
- 30594 • The value of zero in the object pointed to by *unavailable*

#### 30595 ERRORS

30596 These functions may fail if:

30597 [EINVAL] The trace stream identifier argument *trid* is invalid.

30598 The *posix\_trace\_getnext\_event()* and *posix\_trace\_timedgetnext\_event()* functions may fail if:

30599 [EINTR] The operation was interrupted by a signal, and so the call had no effect.

30600 The *posix\_trace\_trygetnext\_event()* function may fail if:

30601 [EINVAL] The trace stream identifier argument *trid* does not correspond to an active  
 30602 trace stream.

30603 TMO The *posix\_trace\_timedgetnext\_event()* function may fail if:

30604 [EINVAL] There is no trace event immediately available from the trace stream, and the  
 30605 *timeout* argument is invalid.

30606 [ETIMEDOUT] No trace event was available from the trace stream before the specified  
30607 timeout *timeout* expired.  
30608

30609 **EXAMPLES**

30610 None.

30611 **APPLICATION USAGE**

30612 None.

30613 **RATIONALE**

30614 None.

30615 **FUTURE DIRECTIONS**

30616 None.

30617 **SEE ALSO**

30618 *posix\_trace\_create()*, **posix\_trace\_event\_info Structure**, *posix\_trace\_open()*, the Base Definitions  
30619 volume of IEEE Std. 1003.1-200x, <sys/types.h>, <trace.h>

30620 **CHANGE HISTORY**

30621 First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

30622 **NAME**

30623        posix\_trace\_open — trace log management

30624 **SYNOPSIS**

30625 TCT TRL #include &lt;trace.h&gt;

30626        int posix\_trace\_open(int *file\_desc*, trace\_id\_t \*trid);

30627

30628 **DESCRIPTION**30629        Refer to *posix\_trace\_close()*.

30630 **NAME**

30631        posix\_trace\_rewind — trace log management

30632 **SYNOPSIS**

30633 TCT TRL #include <trace.h>

30634        int posix\_trace\_rewind(trace\_id\_t *trid*);

30635

30636 **DESCRIPTION**

30637        Refer to *posix\_trace\_close()*.

30638 **NAME**

30639        posix\_trace\_set\_filter — set filter of an initialized trace stream

30640 **SYNOPSIS**

30641 TRC TEF #include &lt;trace.h&gt;

30642        int posix\_trace\_set\_filter(trace\_id\_t *trid*,  
30643            const trace\_event\_set\_t \**set*, int *how*);

30644

30645 **DESCRIPTION**30646        Refer to *posix\_trace\_get\_filter()*.

30647 **NAME**

30648        posix\_trace\_shutdown — trace stream shutdown from a process

30649 **SYNOPSIS**

30650 TRC     #include <sys/types.h>

30651        #include <trace.h>

30652        int posix\_trace\_shutdown(trace\_id\_t *trid*);

30653

30654 **DESCRIPTION**

30655        Refer to *posix\_trace\_create()*.

30656 **NAME**

30657            posix\_trace\_start, posix\_trace\_stop — trace start and stop

30658 **SYNOPSIS**

30659 TRC        #include &lt;trace.h&gt;

30660            int posix\_trace\_start(trace\_id\_t trid);

30661            int posix\_trace\_stop (trace\_id\_t trid);

30662

30663 **DESCRIPTION**30664            The *posix\_trace\_start()* and *posix\_trace\_stop()* functions, respectively, start and stop the trace stream identified by the argument *trid*.30666            The effect of calling the *posix\_trace\_start()* function shall be recorded in the trace stream as the POSIX\_TRACE\_START system trace event and the status of the trace stream shall become POSIX\_TRACE\_RUNNING. If the trace stream is in progress when this function is called, the POSIX\_TRACE\_START system trace event shall not be recorded and the trace stream shall continue to run. If the trace stream is full, the POSIX\_TRACE\_START system trace event shall not be recorded and the status of the trace stream shall not be changed.30672            The effect of calling the *posix\_trace\_stop()* function shall be recorded in the trace stream as the POSIX\_TRACE\_STOP system trace event and the status of the trace stream shall become POSIX\_TRACE\_SUSPENDED. If the trace stream is suspended when this function is called, the POSIX\_TRACE\_STOP system trace event shall not be recorded and the trace stream shall remain suspended. If the trace stream is full, the POSIX\_TRACE\_STOP system trace event shall not be recorded and the status of the trace stream shall not be changed.30678 **RETURN VALUE**

30679            Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

30681 **ERRORS**

30682            These functions may fail if:

30683            [EINVAL]            The value of the argument *trid* does not correspond to an active trace stream and thus no trace stream was started or stopped.

30685            [EINTR]            The operation was interrupted by a signal and thus the trace stream was not necessarily started or stopped.

30687 **EXAMPLES**

30688            None.

30689 **APPLICATION USAGE**

30690            None.

30691 **RATIONALE**

30692            None.

30693 **FUTURE DIRECTIONS**

30694            None.

30695 **SEE ALSO**30696            *posix\_trace\_create()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <trace.h>

30697 **CHANGE HISTORY**

30698 First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.



30699 **NAME**30700 `posix_trace_timedgetnext_event`, — retrieve a trace event30701 **SYNOPSIS**30702 TRC TMO `#include <sys/types.h>`30703 `#include <trace.h>`

```
30704 int posix_trace_timedgetnext_event(trace_id_t trid,
30705 struct posix_trace_event_info *event, void *data,
30706 size_t num_bytes, size_t *data_len, int *unavailable,
30707 const struct timespec *abs_timeout);
30708
```

30709 **DESCRIPTION**30710 Refer to `posix_trace_getnext_event()`.

30711 **NAME**

30712        posix\_trace\_trid\_eventid\_open — manipulate trace event type identifier

30713 **SYNOPSIS**

30714 TRC TEF #include <trace.h>

```
30715 int posix_trace_trid_eventid_open(trace_id_t trid,
30716 const char *event_name, trace_eventid_t *event);
```

30717

30718 **DESCRIPTION**

30719        Refer to *posix\_trace\_eventid\_equal()*.

30720 **NAME**

30721        posix\_trace\_trygetnext\_event — retrieve a trace event

30722 **SYNOPSIS**

30723 TRC TMO #include &lt;sys/types.h&gt;

30724        #include &lt;trace.h&gt;

```
30725 int posix_trace_trygetnext_event(trace_id_t trid,
30726 struct posix_trace_event_info *event, void *data,
30727 size_t num_bytes, size_t *data_len, int *unavailable);
```

30728

30729 **DESCRIPTION**30730        Refer to *posix\_trace\_getnext\_event()*.

30731 **NAME**

30732 `posix_typed_mem_get_info` — query typed memory information

30733 **SYNOPSIS**

30734 **TYM** `#include <sys/mman.h>`

```
30735 int posix_typed_mem_get_info(int fildes,
30736 struct posix_typed_mem_info *info);
```

30737

30738 **DESCRIPTION**

30739 The `posix_typed_mem_get_info()` function returns, in the `posix_tmi_length` field of the  
 30740 **posix\_typed\_mem\_info** structure pointed to by `info`, the maximum length which may be  
 30741 successfully allocated by the typed memory object designated by `fildes`. This maximum length  
 30742 shall take into account the flag `POSIX_TYPED_MEM_ALLOCATE` or  
 30743 `POSIX_TYPED_MEM_ALLOCATE_CONTIG` specified when the typed memory object  
 30744 represented by `fildes` was opened. The maximum length is dynamic; therefore, the value returned  
 30745 is valid only while the current mapping of the corresponding typed memory pool remains  
 30746 unchanged.

30747 If `fildes` represents a typed memory object opened with neither the  
 30748 `POSIX_TYPED_MEM_ALLOCATE` flag nor the `POSIX_TYPED_MEM_ALLOCATE_CONTIG`  
 30749 flag specified, the returned value of `info->posix_tmi_length` is unspecified.

30750 The `posix_typed_mem_get_info()` function may return additional implementation-defined  
 30751 information in other fields of the **posix\_typed\_mem\_info** structure pointed to by `info`.

30752 If the memory object specified by `fildes` is not a typed memory object, then the behavior of this  
 30753 function is undefined.

30754 **RETURN VALUE**

30755 Upon successful completion, the `posix_typed_mem_get_info()` function shall return zero;  
 30756 otherwise, the corresponding error status value shall be returned.

30757 **ERRORS**

30758 The `posix_typed_mem_get_info()` function shall fail if:

30759 [EBADF] The `fildes` argument is not a valid open file descriptor.

30760 [ENODEV] The `fildes` argument is not connected to a memory object supported by this  
 30761 function.

30762 This function shall not return an error code of [EINTR].

30763 **EXAMPLES**

30764 None.

30765 **APPLICATION USAGE**

30766 None.

30767 **RATIONALE**

30768 An application that needs to allocate a block of typed memory with length dependent upon the  
 30769 amount of memory currently available must either query the typed memory object to obtain the  
 30770 amount available, or repeatedly invoke `mmap()` attempting to guess an appropriate length.  
 30771 While the latter method is existing practice with `malloc()`, it is awkward and imprecise. The  
 30772 `posix_typed_mem_get_info()` function allows an application to immediately determine available  
 30773 memory. This is particularly important for typed memory objects that may in some cases be  
 30774 scarce resources. Note that when a typed memory pool is a shared resource, some form of  
 30775 mutual exclusion or synchronization may be required while typed memory is being queried and

30776 allocated to prevent race conditions.

30777 The existing *fstat()* function is not suitable for this purpose. We realize that implementations  
30778 may wish to provide other attributes of typed memory objects (for example, alignment  
30779 requirements, page size, and so on). The *fstat()* function returns a structure which is not  
30780 extensible and, furthermore, contains substantial information that is inappropriate for typed  
30781 memory objects.

30782 **FUTURE DIRECTIONS**

30783 None.

30784 **SEE ALSO**

30785 *fstat()*, *mmap()*, *posix\_typed\_mem\_open()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |

30786 <**sys/mman.h**>

30787 **CHANGE HISTORY**

30788 First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.

30789 **NAME**

30790 posix\_typed\_mem\_open — open a typed memory object

30791 **SYNOPSIS**

30792 TYM #include <sys/mman.h>

30793 int posix\_typed\_mem\_open(const char \*name, int oflag, int tflag);

30794

30795 **DESCRIPTION**

30796 The *posix\_typed\_mem\_open()* function establishes a connection between the typed memory object  
 30797 specified by the string pointed to by *name* and a file descriptor. It creates an open file description  
 30798 that refers to the typed memory object and a file descriptor that refers to that open file  
 30799 description. The file descriptor is used by other functions to refer to that typed memory object. It  
 30800 is unspecified whether the name appears in the file system and is visible to other functions that  
 30801 take path names as arguments. The *name* argument shall conform to the construction rules for a  
 30802 path name. If *name* begins with the slash character, then processes calling  
 30803 *posix\_typed\_mem\_open()* with the same value of *name* shall refer to the same typed memory  
 30804 object. If *name* does not begin with the slash character, the effect is implementation-defined. The  
 30805 interpretation of slash characters other than the leading slash character in *name* is  
 30806 implementation-defined.

30807 Each typed memory object supported in a system is identified by a name which specifies not  
 30808 only its associated typed memory pool, but also the path or port by which it is accessed. That is,  
 30809 the same typed memory pool accessed via several different ports has several different  
 30810 corresponding names. The binding between names and typed memory objects is established in  
 30811 an implementation-defined manner. Unlike shared memory objects, there is ordinarily no way  
 30812 for a program to create a typed memory object.

30813 The value of *tflag* determines how the typed memory object behaves when subsequently  
 30814 mapped by calls to *mmap()*. At most, one of the following flags defined in <sys/mman.h> may  
 30815 be specified:

30816 POSIX\_TYPED\_MEM\_ALLOCATE

30817 Allocate on *mmap()*.

30818 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG

30819 Allocate contiguously on *mmap()*.

30820 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE

30821 Map on *mmap()*, without affecting allocatability.

30822 If *tflag* has the flag POSIX\_TYPED\_MEM\_ALLOCATE specified, any subsequent call to *mmap()*  
 30823 using the returned file descriptor shall result in allocation and mapping of typed memory from  
 30824 the specified typed memory pool. The allocated memory may be a contiguous previously  
 30825 unallocated area of the typed memory pool or several non-contiguous previously unallocated  
 30826 areas (mapped to a contiguous portion of the process address space). If *tflag* has the flag  
 30827 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG specified, any subsequent call to *mmap()* using the  
 30828 returned file descriptor shall result in allocation and mapping of a single contiguous previously  
 30829 unallocated area of the typed memory pool (also mapped to a contiguous portion of the process  
 30830 address space). If *tflag* has none of the flags POSIX\_TYPED\_MEM\_ALLOCATE or  
 30831 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG specified, any subsequent call to *mmap()* using the  
 30832 returned file descriptor shall map an application-chosen area from the specified typed memory  
 30833 pool such that this mapped area becomes unavailable for allocation until unmapped by all  
 30834 processes. If *tflag* has the flag POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE specified, any  
 30835 subsequent call to *mmap()* using the returned file descriptor shall map an application-chosen  
 30836 area from the specified typed memory pool without an effect on the availability of that area for

30837 allocation; that is, mapping such an object leaves each byte of the mapped area unallocated if it  
 30838 was unallocated prior to the mapping or allocated if it was allocated prior to the mapping. The  
 30839 appropriate privilege to specify the POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag is  
 30840 implementation-defined.

30841 If successful, *posix\_typed\_mem\_open()* returns a file descriptor for the typed memory object that  
 30842 is the lowest numbered file descriptor not currently open for that process. The open file  
 30843 description is new, and therefore the file descriptor does not share it with any other processes. It  
 30844 is unspecified whether the file offset is set. The FD\_CLOEXEC file descriptor flag associated  
 30845 with the new file descriptor shall be cleared.

30846 The behavior of *msync()*, *ftruncate()*, and all file operations other than *mmap()*,  
 30847 *posix\_mem\_offset()*, *posix\_typed\_mem\_get\_info()*, *fstat()*, *dup()*, *dup2()*, and *close()*, is unspecified  
 30848 when passed a file descriptor connected to a typed memory object by this function.

30849 The file status flags of the open file description shall be set according to the value of *oflag*.  
 30850 Applications shall specify exactly one of the three access mode values described below and  
 30851 defined in the header `<fcntl.h>`, as the value of *oflag*.

30852 O\_RDONLY      Open for read access only.  
 30853 O\_WRONLY      Open for write access only.  
 30854 O\_RDWR        Open for read or write access.

#### 30855 RETURN VALUE

30856 Upon successful completion, the *posix\_typed\_mem\_open()* function shall return a non-negative  
 30857 integer representing the lowest numbered unused file descriptor. Otherwise, it shall return `-1`  
 30858 and set *errno* to indicate the error.

#### 30859 ERRORS

30860 The *posix\_typed\_mem\_open()* function shall fail if:

30861 [EACCES]      The typed memory object exists and the permissions specified by *oflag* are  
 30862 denied.

30863 [EINTR]        The *posix\_typed\_mem\_open()* operation was interrupted by a signal.

30864 [EINVAL]       The flags specified in *tflag* are invalid (more than one of  
 30865 POSIX\_TYPED\_MEM\_ALLOCATE,  
 30866 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG, or  
 30867 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE is specified).

30868 [EMFILE]       Too many file descriptors are currently in use by this process.

30869 [ENAMETOOLONG]   The length of the *name* argument exceeds {PATH\_MAX} or a path name  
 30870 component is longer than {NAME\_MAX}.  
 30871

30872 [ENFILE]       Too many file descriptors are currently open in the system.

30873 [ENOENT]       The named typed memory object does not exist.

30874 [EPERM]        The caller lacks the appropriate privilege to specify the flag  
 30875 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE in argument *tflag*.

30876 **EXAMPLES**

30877           None.

30878 **APPLICATION USAGE**

30879           None.

30880 **RATIONALE**

30881           None.

30882 **FUTURE DIRECTIONS**

30883           None.

30884 **SEE ALSO**

30885           *close()*, *dup()*, *exec*, *fcntl()*, *fstat()*, *fruncate()*, *mmap()*, *msync()*, *posix\_mem\_offset()*,  
30886           *posix\_typed\_mem\_get\_info()*, *umask()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |  
30887           <fcntl.h,> <sys/mman.h>

30888 **CHANGE HISTORY**

30889           First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.



30890 **NAME**

30891 pow, powf, powl — power function

30892 **SYNOPSIS**

30893 #include &lt;math.h&gt;

30894 double pow(double x, double y);

30895 float powf(float x, float y);

30896 long double powl(long double x, long double y);

30897 **DESCRIPTION**

30898 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 30899 conflict between the requirements described here and the ISO C standard is unintentional. This  
 30900 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

30901 These functions shall compute the value of  $x$  raised to the power  $y$ ,  $x^y$ . If  $x$  is negative, the  
 30902 application shall ensure that  $y$  is an integer value.

30903 An application wishing to check for error situations should set *errno* to 0 before calling *pow()*. If  
 30904 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

30905 **RETURN VALUE**30906 Upon successful completion, these functions shall return the value of  $x$  raised to the power  $y$ .30907 If  $x$  is 0 and  $y$  is 0, 1.0 shall be returned.

30908 **XSI** If  $y$  is NaN, or  $y$  is non-zero and  $x$  is NaN, NaN shall be returned and *errno* may be set to  
 30909 [EDOM]. If  $y$  is 0.0 and  $x$  is NaN, either 1.0 shall be returned, or NaN shall be returned and *errno*  
 30910 may be set to [EDOM].

30911 **XSI** If  $x$  is 0.0 and  $y$  is negative,  $-\text{HUGE\_VAL}$  shall be returned and *errno* may be set to [EDOM] or  
 30912 [ERANGE].

30913 If the correct value would cause overflow,  $\pm\text{HUGE\_VAL}$  shall be returned, and *errno* set to  
 30914 [ERANGE].

30915 If the correct value would cause underflow, 0 is returned and *errno* may be set to [ERANGE].30916 **ERRORS**

30917 These functions shall fail if:

30918 [EDOM] The value of  $x$  is negative and  $y$  is non-integral.

30919 [ERANGE] The value to be returned would have caused overflow.

30920 These functions may fail if:

30921 **XSI** [EDOM] The value of  $x$  is 0.0 and  $y$  is negative, or  $y$  is NaN.

30922 [ERANGE] The correct value would cause underflow.

30923 **XSI** No other errors shall occur.

30924 **EXAMPLES**

30925 None.

30926 **APPLICATION USAGE**

30927 None.

30928 **RATIONALE**

30929 None.

30930 **FUTURE DIRECTIONS**

30931 None.

30932 **SEE ALSO**30933 *exp()*, *isnan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>30934 **CHANGE HISTORY**

30935 First released in Issue 1. Derived from Issue 1 of the SVID.

30936 **Issue 4**30937 References to *matherr()* are removed.30938 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the  
30939 ISO C standard and to rationalize error handling in the mathematics functions.

30940 The return value specified for [EDOM] is marked as an extension.

30941 **Issue 5**30942 The DESCRIPTION is updated to indicate how an application should check for an error. This  
30943 text was previously published in the APPLICATION USAGE section.30944 **Issue 6**

30945 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

30946 The *powf()* and *powl()* functions are added for alignment with the ISO/IEC 9899: 1999 standard.

30947 **NAME**

30948        pread — read from a file

30949 **SYNOPSIS**

30950 xSI        #include &lt;unistd.h&gt;

30951        ssize\_t pread(int *fd*, void \**buf*, size\_t *nbyte*, off\_t *offset*);

30952

30953 **DESCRIPTION**30954        Refer to *read()*.

30955 **NAME**

30956       printf — print formatted output

30957 **SYNOPSIS**

30958       #include <stdio.h>

30959       int printf(const char \*restrict *format*, ...);

30960 **DESCRIPTION**

30961       Refer to *fprintf()*.

## 30962 NAME

30963 pselect, select — synchronous I/O multiplexing

## 30964 SYNOPSIS

```

30965 #include <sys/select.h>

30966 int pselect(int nfds, fd_set *readfds, fd_set *writefds,
30967 fd_set *errorfds, const struct timespec *timeout,
30968 const sigset_t *sigmask);
30969 int select(int nfds, fd_set *restrict readfds,
30970 fd_set *restrict writefds, fd_set *restrict errorfds,
30971 struct timeval *restrict timeout);
30972 void FD_CLR(int fd, fd_set *fdset);
30973 int FD_ISSET(int fd, fd_set *fdset);
30974 void FD_SET(int fd, fd_set *fdset);
30975 void FD_ZERO(fd_set *fdset);

```

## 30976 DESCRIPTION

30977 The *pselect()* function shall examine the file descriptor sets whose addresses are passed in the  
 30978 *readfds*, *writefds*, and *errorfds* parameters to see if some of their descriptors are ready for reading,  
 30979 are ready for writing, or have an exceptional condition pending, respectively.

30980 The *select()* function shall be equivalent to the *pselect()* function, except as follows:

- 30981 • For the *select()* function, the timeout period is given in seconds and microseconds in an  
 30982 argument of type **struct timeval**, whereas for the *pselect()* function the timeout period is  
 30983 given in seconds and nanoseconds in an argument of type **struct timespec**.
- 30984 • The *select()* function has no *sigmask* argument; it shall behave as *pselect()* does when *sigmask*  
 30985 is a null pointer.
- 30986 • Upon successful completion, the *select()* function may modify the object pointed to by the  
 30987 *timeout* argument.

30988 The *pselect()* and *select()* functions shall support regular files, terminal and pseudo-terminal  
 30989 XSR devices, **STREAMS**-based files, FIFOs, pipes, and sockets. The behavior of *pselect()* and *select()*  
 30990 on file descriptors that refer to other types of file is unspecified.

30991 The *nfds* argument specifies the range of descriptors to be tested. The first *nfds* descriptors shall  
 30992 be checked in each set; that is, the descriptors from zero through *nfds*-1 in the descriptor sets  
 30993 shall be examined.

30994 If the *readfds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 30995 specifies the file descriptors to be checked for being ready to read, and on output indicates  
 30996 which file descriptors are ready to read.

30997 If the *writefds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 30998 specifies the file descriptors to be checked for being ready to write, and on output indicates  
 30999 which file descriptors are ready to write.

31000 If the *errorfds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 31001 specifies the file descriptors to be checked for error conditions pending, and on output indicates  
 31002 which file descriptors have error conditions pending.

31003 Upon successful completion, the *pselect()* function shall modify the objects pointed to by the  
 31004 *readfds*, *writefds*, and *errorfds* arguments to indicate which file descriptors are ready for reading,  
 31005 ready for writing, or have an error condition pending, respectively, and shall return the total  
 31006 number of ready descriptors in all the output sets. For each file descriptor less than *nfds*, the  
 31007 corresponding bit shall be set on successful completion if it was set on input and the associated

31008 condition is true for that file descriptor.

31009 If none of the selected descriptors are ready for the requested operation, the *pselect()* function  
31010 may block until at least one of the requested operations becomes ready. The parameter *timeout*  
31011 controls how long the *pselect()* function may take to complete. If the *timeout* parameter is not a  
31012 null pointer, it specifies a maximum interval to wait for the selection to complete. If the specified  
31013 time interval expires without any requested operation becoming ready, the function shall return.  
31014 If the *timeout* parameter is a null pointer, then the call to *pselect()* shall block indefinitely until at  
31015 least one descriptor meets the specified criteria. To effect a poll, the *timeout* parameter should not  
31016 be a null pointer, and should point to a zero-valued **timespec** structure.

31017 The use of a timeout does not affect any pending timers set up by *alarm()*, *ualarm()*, or  
31018 *setitimer()*.

31019 Implementations may place limitations on the maximum timeout interval supported. All  
31020 implementations shall support a maximum timeout interval of at least 31 days. If the *timeout*  
31021 argument specifies a timeout interval greater than the implementation-defined maximum value,  
31022 the maximum value shall be used as the actual timeout value. Implementations may also place  
31023 limitations on the granularity of timeout intervals. If the requested timeout interval requires a  
31024 finer granularity than the implementation supports, the actual timeout interval shall be rounded  
31025 up to the next supported value.

31026 If *sigmask* is not a null pointer, then the *pselect()* function shall replace the signal mask of the  
31027 process by the set of signals pointed to by *sigmask* before examining the descriptors, and shall  
31028 restore the signal mask of the process before returning.

31029 A descriptor shall be considered ready for reading when a call to an input function with  
31030 O\_NONBLOCK clear would not block, whether or not the function would transfer data  
31031 successfully. (The function might return data, an end-of-file indication, or an error other than  
31032 one indicating that it is blocked, and in each of these cases the descriptor shall be considered  
31033 ready for reading.)

31034 A descriptor shall be considered ready for writing when a call to an output function with  
31035 O\_NONBLOCK clear would not block, whether or not the function would transfer data  
31036 successfully.

31037 If a socket has a pending error, it shall be considered to have an exceptional condition pending.  
31038 Otherwise, what constitutes an exceptional condition is file type-specific. For a file descriptor for  
31039 use with a socket, it is protocol-specific except as noted below. For other file types it is  
31040 implementation-defined. If the operation is meaningless for a particular file type, *pselect()* shall  
31041 indicate that the descriptor is ready for read or write operations, and shall indicate that the  
31042 descriptor has no exceptional condition pending.

31043 If a descriptor refers to a socket, the implied input function is the *recvmsg()* function with  
31044 parameters requesting normal and ancillary data, such that the presence of either type shall  
31045 cause the socket to be marked as readable. The presence of out-of-band data will be checked if  
31046 the socket option SO\_OOBINLINE has been enabled, as out-of-band data is enqueued with  
31047 normal data. If the socket is currently listening, then it will be marked as readable if an incoming  
31048 connection request has been received, and a call to the *accept()* function is guaranteed to  
31049 complete without blocking.

31050 If a descriptor refers to a socket, the implied output function is the *sendmsg()* function supplying  
31051 an amount of normal data equal to the current value of the SO\_SNDLOWAT option for the  
31052 socket. If a non-blocking call to the *connect()* function has been made for a socket, and the  
31053 connection attempt has either succeeded or failed leaving a pending error, the socket shall be  
31054 marked as writable.

31055 A socket shall be considered to have an exceptional condition pending if a receive operation  
 31056 with `O_NONBLOCK` clear for the open file description and with the `MSG_OOB` flag set would  
 31057 return out-of-band data without blocking. (It is protocol-specific whether the `MSG_OOB` flag  
 31058 would be used to read out-of-band data.) A socket shall also be considered to have an  
 31059 exceptional condition pending if an out-of-band data mark is present in the receive queue. Other  
 31060 circumstances under which a socket may be considered to have an exceptional condition  
 31061 pending are protocol-specific and implementation-defined.

31062 If the `readfds`, `writfds`, and `errorfds` arguments are all null pointers and the `timeout` argument is not  
 31063 a null pointer, `select()` blocks for the time specified, or until interrupted by a signal. If the `readfds`,  
 31064 `writfds`, and `errorfds` arguments are all null pointers and the `timeout` argument is a null pointer,  
 31065 `pselect()` blocks until interrupted by a signal.

31066 File descriptors associated with regular files shall always select true for ready to read, ready to  
 31067 write, and error conditions.

31068 On failure, the objects pointed to by the `readfds`, `writfds`, and `errorfds` arguments are not modified.  
 31069 If the timeout interval expires without the specified condition being true for any of the specified  
 31070 file descriptors, the objects pointed to by the `readfds`, `writfds`, and `errorfds` arguments have all bits  
 31071 set to 0.

31072 File descriptor masks of type `fd_set` can be initialized and tested with `FD_CLR()`, `FD_ISSET()`,  
 31073 `FD_SET()`, and `FD_ZERO()`. It is unspecified whether each of these is a macro or a function. If a  
 31074 macro definition is suppressed in order to access an actual function, or a program defines an  
 31075 external identifier with any of these names, the behavior is undefined.

31076 The macro `FD_CLR(fd, fdsetp)` shall remove the file descriptor `fd` from the set pointed to by `fdsetp`.  
 31077 If `fd` is not a member of this set, there shall be no effect on the set, nor will an error be returned.

31078 The macro `FD_ISSET(fd, fdsetp)` shall evaluate to non-zero if the file descriptor `fd` is a member of  
 31079 the set pointed to by `fdsetp`, and shall evaluate to zero otherwise.

31080 The macro `FD_SET(fd, fdsetp)` shall add the file descriptor `fd` to the set pointed to by `fdsetp`. If the  
 31081 file descriptor `fd` is already in this set, there shall be no effect on the set, nor will an error be  
 31082 returned.

31083 The macro `FD_ZERO(fdsetp)` shall initialize the descriptor set pointed to by `fdsetp` to the null set.  
 31084 No error is returned if the set is not empty at the time `FD_ZERO()` is invoked.

31085 The behavior of these macros is undefined if the `fd` argument is less than 0 or greater than or  
 31086 equal to `FD_SETSIZE`, or if `fd` is not a valid file descriptor, or if any of the arguments are  
 31087 expressions with side effects.

#### 31088 RETURN VALUE

31089 Upon successful completion, the `pselect()` and `select()` functions shall return the total number of  
 31090 bits set in the bit masks. Otherwise, `-1` shall be returned, and shall set `errno` to indicate the error.

31091 `FD_CLR()`, `FD_SET()`, and `FD_ZERO()` return no value. `FD_ISSET()` returns a non-zero value if  
 31092 the bit for the file descriptor `fd` is set in the file descriptor set pointed to by `fdset`, and 0 otherwise.

#### 31093 ERRORS

31094 Under the following conditions, `pselect()` and `select()` shall fail and set `errno` to:

31095 [EBADF] One or more of the file descriptor sets specified a file descriptor that is not a  
 31096 valid open file descriptor.

31097 [EINTR] The function was interrupted before any of the selected events occurred and  
 31098 before the timeout interval expired.

- 31099 XSI If SA\_RESTART has been set for the interrupting signal, it is implementation-  
31100 defined whether the function restarts or returns with [EINTR].
- 31101 [EINVAL] An invalid timeout interval was specified.
- 31102 [EINVAL] The *nfds* argument is less than 0 or greater than FD\_SETSIZE.
- 31103 XSR [EINVAL] One of the specified file descriptors refers to a STREAM or multiplexer that is  
31104 linked (directly or indirectly) downstream from a multiplexer.
- 31105 **EXAMPLES**
- 31106 None.
- 31107 **APPLICATION USAGE**
- 31108 None.
- 31109 **RATIONALE**
- 31110 In previous versions of the Single UNIX Specification, the *select()* function was defined in the  
31111 `<sys/time.h>` header. This is now changed to `<sys/select.h>`. The rationale for this change was  
31112 as follows: the introduction of the *pselect()* function included the `<sys/select.h>` header and the  
31113 `<sys/select.h>` header defines all the related definitions for the *pselect()* and *select()* functions.  
31114 Backwards-compatibility to existing XSI implementations is handled by allowing `<sys/time.h>`  
31115 to include `<sys/select.h>`.
- 31116 **FUTURE DIRECTIONS**
- 31117 None.
- 31118 **SEE ALSO**
- 31119 *accept()*, *alarm()*, *connect()*, *fcntl()*, *poll()*, *read()*, *recvmsg()*, *sendmsg()*, *setitimer()*, *ualarm()*,  
31120 *write()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<sys/select.h>`, `<sys/time.h>`
- 31121 **CHANGE HISTORY**
- 31122 First released in Issue 4, Version 2.
- 31123 **Issue 5**
- 31124 Moved from X/OPEN UNIX extension to BASE.
- 31125 In the ERRORS section, the text has been changed to indicate that [EINVAL] is returned when  
31126 *nfds* is less than 0 or greater than FD\_SETSIZE. It previously stated less than 0, or greater than or  
31127 equal to FD\_SETSIZE.
- 31128 Text about *timeout* is moved from the APPLICATION USAGE section to the DESCRIPTION.
- 31129 **Issue 6**
- 31130 The Open Group corrigenda item U026/6 has been applied, changing the occurrences of *readfs*  
31131 and *writefs* in the *select()* DESCRIPTION to be *readfds* and *writefds*.
- 31132 Text referring to sockets is added to the DESCRIPTION.
- 31133 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
31134 marked as part of the XSI STREAMS Option Group.
- 31135 The following new requirements on POSIX implementations derive from alignment with the  
31136 Single UNIX Specification:
- 31137 • These functions are now mandatory.
- 31138 The *pselect()* function is added for alignment with IEEE Std. 1003.1g-2000 and additional detail  
31139 related to sockets semantics is added to the DESCRIPTION.
- 31140 The *select()* function now requires inclusion of `<sys/select.h>`.



31141 The **restrict** keyword is added to the *select()* prototype for alignment with the  
31142 ISO/IEC 9899:1999 standard.

31143 **NAME**

31144 pthread\_atfork — register fork handlers

31145 **SYNOPSIS**

31146 THR #include &lt;pthread.h&gt;

31147 #include &lt;sys/types.h&gt;

31148 #include &lt;unistd.h&gt;

```
31149 int pthread_atfork(void (*prepare)(void), void (*parent)(void),
31150 void (*child)(void));
```

31151

31152 **DESCRIPTION**

31153 The *pthread\_atfork()* function shall declare fork handlers to be called before and after *fork()*, in  
 31154 the context of the thread that called *fork()*. The *prepare* fork handler shall be called before *fork()*  
 31155 processing commences. The *parent* fork handle shall be called after *fork()* processing completes  
 31156 in the parent process. The *child* fork handler shall be called after *fork()* processing completes in  
 31157 the child process. If no handling is desired at one or more of these three points, the  
 31158 corresponding fork handler address(es) may be set to NULL.

31159 The order of calls to *pthread\_atfork()* is significant. The *parent* and *child* fork handlers shall be  
 31160 called in the order in which they were established by calls to *pthread\_atfork()*. The *prepare* fork  
 31161 handlers shall be called in the opposite order.

31162 **RETURN VALUE**

31163 Upon successful completion, *pthread\_atfork()* shall return a value of zero; otherwise, an error  
 31164 number shall be returned to indicate the error.

31165 **ERRORS**31166 The *pthread\_atfork()* function shall fail if:

31167 [ENOMEM] Insufficient table space exists to record the fork handler addresses.

31168 The *pthread\_atfork()* function shall not return an error code of [EINTR].31169 **EXAMPLES**

31170 None.

31171 **APPLICATION USAGE**

31172 None.

31173 **RATIONALE**

31174 There are at least two serious problems with the semantics of *fork()* in a multi-threaded  
 31175 program. One problem has to do with state (for example, memory) covered by mutexes.  
 31176 Consider the case where one thread has a mutex locked and the state covered by that mutex is  
 31177 inconsistent while another thread calls *fork()*. In the child, the mutex is in the locked state  
 31178 (locked by a nonexistent thread and thus can never be unlocked). Having the child simply  
 31179 reinitialize the mutex is unsatisfactory since this approach does not resolve the question about  
 31180 how to correct or otherwise deal with the inconsistent state in the child.

31181 It is suggested that programs that use *fork()* call an *exec* function very soon afterwards in the  
 31182 child process, thus resetting all states. In the meantime, only a short list of async-signal-safe  
 31183 library routines are promised to be available.

31184 Unfortunately, this solution does not address the needs of multi-threaded libraries. Application  
 31185 programs may not be aware that a multi-threaded library is in use, and they feel free to call any  
 31186 number of library routines between the *fork()* and *exec* calls, just as they always have. Indeed,  
 31187 they may be extant single-threaded programs and cannot, therefore, be expected to obey new  
 31188 restrictions imposed by the threads library.

31189 On the other hand, the multi-threaded library needs a way to protect its internal state during  
31190 *fork()* in case it is re-entered later in the child process. The problem arises especially in multi-  
31191 threaded I/O libraries, which are almost sure to be invoked between the *fork()* and *exec* calls to  
31192 effect I/O redirection. The solution may require locking mutex variables during *fork()*, or it may  
31193 entail simply resetting the state in the child after the *fork()* processing completes.

31194 The *pthread\_atfork()* function provides multi-threaded libraries with a means to protect  
31195 themselves from innocent application programs that call *fork()*, and it provides multi-threaded  
31196 application programs with a standard mechanism for protecting themselves from *fork()* calls in  
31197 a library routine or the application itself.

31198 The expected usage is that the *prepare* handler acquires all mutex locks and the other two fork  
31199 handlers release them.

31200 For example, an application can supply a *prepare* routine that acquires the necessary mutexes the  
31201 library maintains and supply *child* and *parent* routines that release those mutexes, thus ensuring  
31202 that the child gets a consistent snapshot of the state of the library (and that no mutexes are left  
31203 stranded). Alternatively, some libraries might be able to supply just a *child* routine that re-  
31204 initializes the mutexes in the library and all associated states to some known value (for example,  
31205 what it was when the image was originally executed).

31206 When *fork()* is called, only the calling thread is duplicated in the child process. Synchronization  
31207 variables remain in the same state in the child as they were in the parent at the time *fork()* was  
31208 called. Thus, for example, mutex locks may be held by threads that no longer exist in the child  
31209 process, and any associated states may be inconsistent. The parent process may avoid this by  
31210 explicit code that acquires and releases locks critical to the child via *pthread\_atfork()*. In addition,  
31211 any critical threads need to be recreated and re-initialized to the proper state in the child (also  
31212 via *pthread\_atfork()*).

31213 A higher-level package may acquire locks on its own data structures before invoking lower-level  
31214 packages. Under this scenario, the order specified for fork handler calls allows a simple rule of  
31215 initialization for avoiding package deadlock: a package initializes all packages on which it  
31216 depends before it calls the *pthread\_atfork()* function for itself.

#### 31217 **FUTURE DIRECTIONS**

31218 None.

#### 31219 **SEE ALSO**

31220 *atexit()*, *fork()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**sys/types.h**>

#### 31221 **CHANGE HISTORY**

31222 First released in Issue 5. Derived from the POSIX Threads Extension.

31223 IEEE PASC Interpretation 1003.1c #4 is included.

#### 31224 **Issue 6**

31225 The *pthread\_atfork()* function is marked as part of the Threads option.

31226 The <**pthread.h**> header is added to the SYNOPSIS.

31227 **NAME**

31228 pthread\_attr\_destroy, pthread\_attr\_init — destroy and initialize threads attributes object

31229 **SYNOPSIS**

31230 THR #include <pthread.h>

31231 int pthread\_attr\_destroy(pthread\_attr\_t \*attr);

31232 int pthread\_attr\_init(pthread\_attr\_t \*attr);

31233

31234 **DESCRIPTION**

31235 The *pthread\_attr\_destroy()* function is used to destroy a thread attributes object. An  
31236 implementation may cause *pthread\_attr\_destroy()* to set *attr* to an implementation-defined  
31237 invalid value. The behavior of using the attribute after it has been destroyed is undefined.

31238 The *pthread\_attr\_init()* function initializes a thread attributes object *attr* with the default value  
31239 for all of the individual attributes used by a given implementation.

31240 The resulting attributes object (possibly modified by setting individual attribute values), when  
31241 used by *pthread\_create()* defines the attributes of the thread created. A single attributes object can  
31242 be used in multiple simultaneous calls to *pthread\_create()*. The effect of initializing an already  
31243 initialized thread attributes object is undefined.

31244 **RETURN VALUE**

31245 Upon successful completion, *pthread\_attr\_destroy()* and *pthread\_attr\_init()* shall return a value of  
31246 0; otherwise, an error number shall be returned to indicate the error.

31247 **ERRORS**

31248 The *pthread\_attr\_init()* function shall fail if:

31249 [ENOMEM] Insufficient memory exists to initialize the thread attributes object.

31250 These functions shall not return an error code of [EINTR].

31251 **EXAMPLES**

31252 None.

31253 **APPLICATION USAGE**

31254 None.

31255 **RATIONALE**

31256 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to  
31257 support probable future standardization in these areas without requiring that the function itself  
31258 be changed.

31259 Attributes objects provide clean isolation of the configurable aspects of threads. For example,  
31260 “stack size” is an important attribute of a thread, but it cannot be expressed portably. When  
31261 porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects  
31262 can help by allowing the changes to be isolated in a single place, rather than being spread across  
31263 every instance of thread creation.

31264 Attributes objects can be used to set up “classes” of threads with similar attributes; for example,  
31265 “threads with large stacks and high priority” or “threads with minimal stacks”. These classes  
31266 can be defined in a single place and then referenced wherever threads need to be created.  
31267 Changes to “class” decisions become straightforward, and detailed analysis of each  
31268 *pthread\_create()* call is not required.

31269 The attributes objects are defined as opaque types as an aid to extensibility. If these objects had  
31270 been specified as structures, adding new attributes would force recompilation of all multi-  
31271 threaded programs when the attributes objects are extended; this might not be possible if

31272 different program components were supplied by different vendors.

31273 Additionally, opaque attributes objects present opportunities for improving performance.  
31274 Argument validity can be checked once when attributes are set, rather than each time a thread is  
31275 created. Implementations often need to cache kernel objects that are expensive to create.  
31276 Opaque attributes objects provide an efficient mechanism to detect when cached objects become  
31277 invalid due to attribute changes.

31278 Because assignment is not necessarily defined on a given opaque type, implementation-defined  
31279 default values cannot be defined in a portable way. The solution to this problem is to allow  
31280 attributes objects to be initialized dynamically by attributes object initialization functions, so  
31281 that default values can be supplied automatically by the implementation.

31282 The following proposal was provided as a suggested alternative to the supplied attributes:

- 31283 1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to  
31284 the initialization routines (*pthread\_create()*, *pthread\_mutex\_init()*, *pthread\_cond\_init()*). The  
31285 parameter containing the flags should be an opaque type for extensibility. If no flags are  
31286 set in the parameter, then the objects are created with default characteristics. An  
31287 implementation may specify implementation-defined flag values and associated behavior.
- 31288 2. If further specialization of mutexes and condition variables is necessary, implementations  
31289 may specify additional procedures that operate on the **pthread\_mutex\_t** and  
31290 **pthread\_cond\_t** objects (instead of on attributes objects).

31291 The difficulties with this solution are:

- 31292 1. A bitmask is not opaque if bits have to be set into bitvector attributes objects using  
31293 explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**,  
31294 application programmers need to know the location of each bit. If bits are set or read by  
31295 encapsulation (that is, get or set functions), then the bitmask is merely an implementation  
31296 of attributes objects as currently defined and should not be exposed to the programmer.
- 31297 2. Many attributes are not Boolean or very small integral values. For example, scheduling  
31298 policy may be placed in 3-bit or 4-bit, but priority requires 5-bit or more, thereby taking up  
31299 at least 8-bit out of a possible 16-bit on machines with 16-bit integers. Because of this, the  
31300 bitmask can only reasonably control whether particular attributes are set or not, and it  
31301 cannot serve as the repository of the value itself. The value needs to be specified as a  
31302 function parameter (which is non-extensible), or by setting a structure field (which is non-  
31303 opaque), or by get and set functions (making the bitmask a redundant addition to the  
31304 attributes objects).

31305 Stack size is defined as an optional attribute because the very notion of a stack is inherently  
31306 machine-dependent. Some implementations may not be able to change the size of the stack, for  
31307 example, and others may not need to because stack pages may be discontinuous and can be  
31308 allocated and released on demand.

31309 The attribute mechanism has been designed in large measure for extensibility. Future extensions  
31310 to the attribute mechanism or to any attributes object defined in this volume of  
31311 IEEE Std. 1003.1-200x has to be done with care so as not to affect binary-compatibility.

31312 Attributes objects, even if allocated by means of dynamic allocation functions such as *malloc()*,  
31313 may have their size fixed at compile time. This means, for example, a *pthread\_create()* in an  
31314 implementation with extensions to the **pthread\_attr\_t** cannot look beyond the area that the  
31315 binary application assumes is valid. This suggests that implementations should maintain a size  
31316 field in the attributes object, as well as possibly version information, if extensions in different  
31317 directions (possibly by different vendors) are to be accommodated.

31318 **FUTURE DIRECTIONS**

31319 None.

31320 **SEE ALSO**

31321 *pthread\_attr\_getstackaddr()*, *pthread\_attr\_getstacksize()*, *pthread\_attr\_getdetachstate()*,  
31322 *pthread\_create()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

31323 **CHANGE HISTORY**

31324 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31325 **Issue 6**

31326 The *pthread\_attr\_destroy()* and *pthread\_attr\_init()* functions marked as part of the Threads  
31327 option.

31328 IEEE PASC Interpretation 1003.1 #107 is applied, noting that the effect of initializing an already  
31329 initialized thread attributes object is undefined.

31330 **NAME**

31331 pthread\_attr\_getdetachstate, pthread\_attr\_setdetachstate — get or set detachstate attribute

31332 **SYNOPSIS**

```
31333 THR #include <pthread.h>
31334
31334 int pthread_attr_getdetachstate(const pthread_attr_t *attr,
31335 int *detachstate);
31336 int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
31337
```

31338 **DESCRIPTION**

31339 The *detachstate* attribute controls whether the thread is created in a detached state. If the thread  
 31340 is created detached, then use of the ID of the newly created thread by the *pthread\_detach()* or  
 31341 *pthread\_join()* function is an error.

31342 The *pthread\_attr\_getdetachstate()* and *pthread\_attr\_setdetachstate()* functions, respectively, get and  
 31343 set the *detachstate* attribute in the *attr* object.

31344 For *pthread\_attr\_getdetachstate()*, *detachstate* shall be set to either  
 31345 PTHREAD\_CREATE\_DETACHED or PTHREAD\_CREATE\_JOINABLE.

31346 For *pthread\_attr\_setdetachstate()*, the application shall set *detachstate* to either  
 31347 PTHREAD\_CREATE\_DETACHED or PTHREAD\_CREATE\_JOINABLE.

31348 A value of PTHREAD\_CREATE\_DETACHED causes all threads created with *attr* to be in the  
 31349 detached state, whereas using a value of PTHREAD\_CREATE\_JOINABLE shall cause all threads  
 31350 created with *attr* to be in the joinable state. The default value of the *detachstate* attribute is  
 31351 PTHREAD\_CREATE\_JOINABLE.

31352 **RETURN VALUE**

31353 Upon successful completion, *pthread\_attr\_getdetachstate()* and *pthread\_attr\_setdetachstate()* shall  
 31354 return a value of 0; otherwise, an error number shall be returned to indicate the error.

31355 The *pthread\_attr\_getdetachstate()* function stores the value of the *detachstate* attribute in *detachstate*  
 31356 if successful.

31357 **ERRORS**

31358 The *pthread\_attr\_setdetachstate()* function shall fail if:

31359 [EINVAL] The value of *detachstate* was not valid

31360 These functions shall not return an error code of [EINTR].

31361 **EXAMPLES**

31362 None.

31363 **APPLICATION USAGE**

31364 None.

31365 **RATIONALE**

31366 None.

31367 **FUTURE DIRECTIONS**

31368 None.

31369 **SEE ALSO**

31370 *pthread\_attr\_destroy()*, *pthread\_attr\_getstackaddr()*, *pthread\_attr\_getstacksize()*, *pthread\_create()*, the  
 31371 Base Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

31372 **CHANGE HISTORY**

31373 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31374 **Issue 6**

31375 The *pthread\_attr\_setdetachstate()* and *pthread\_attr\_getdetachstate()* functions are marked as part of  
31376 the Threads option.

31377 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.



31378 **NAME**

31379 pthread\_attr\_getguardsize, pthread\_attr\_setguardsize — get or set the thread guardsize  
 31380 attribute

31381 **SYNOPSIS**

```
31382 xSI #include <pthread.h>
31383
31383 int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
31384 size_t *restrict guardsize);
31385 int pthread_attr_setguardsize(pthread_attr_t *attr,
31386 size_t guardsize);
31387
```

31388 **DESCRIPTION**

31389 The *guardsize* attribute controls the size of the guard area for the created thread's stack. The  
 31390 *guardsize* attribute provides protection against overflow of the stack pointer. If a thread's stack is  
 31391 created with guard protection, the implementation allocates extra memory at the overflow end  
 31392 of the stack as a buffer against stack overflow of the stack pointer. If an application overflows  
 31393 into this buffer an error results (possibly in a SIGSEGV signal being delivered to the thread).

31394 The *pthread\_attr\_getguardsize()* function gets the *guardsize* attribute in the *attr* object. This  
 31395 attribute is returned in the *guardsize* parameter.

31396 The *pthread\_attr\_setguardsize()* function sets the *guardsize* attribute in the *attr* object. The new  
 31397 value of this attribute is obtained from the *guardsize* parameter. If *guardsize* is zero, a guard area  
 31398 shall not be provided for threads created with *attr*. If *guardsize* is greater than zero, a guard area  
 31399 of at least size *guardsize* bytes is provided for each thread created with *attr*.

31400 A conforming implementation is permitted to round up the value contained in *guardsize* to a  
 31401 multiple of the configurable system variable {PAGESIZE} (see <sys/mman.h>). If an  
 31402 implementation rounds up the value of *guardsize* to a multiple of {PAGESIZE}, a call to  
 31403 *pthread\_attr\_getguardsize()* specifying *attr* shall store in the *guardsize* parameter the guard size  
 31404 specified by the previous *pthread\_attr\_setguardsize()* function call.

31405 The default value of the *guardsize* attribute is {PAGESIZE} bytes. The actual value of {PAGESIZE}  
 31406 is implementation-defined and may not be the same on all implementations.

31407 If the *stackaddr* attribute has been set (that is, the caller is allocating and managing its own thread  
 31408 stacks), the *guardsize* attribute is ignored and no protection shall be provided by the  
 31409 implementation. It is the responsibility of the application to manage stack overflow along with  
 31410 stack allocation and management in this case.

31411 **RETURN VALUE**

31412 If successful, the *pthread\_attr\_getguardsize()* and *pthread\_attr\_setguardsize()* functions shall return  
 31413 zero; otherwise, an error number shall be returned to indicate the error.

31414 **ERRORS**

31415 The *pthread\_attr\_getguardsize()* and *pthread\_attr\_setguardsize()* functions shall fail if:

31416 [EINVAL] The attribute *attr* is invalid.

31417 [EINVAL] The parameter *guardsize* is invalid.

31418 These functions shall not return an error code of [EINTR].

31419 **EXAMPLES**

31420 None.

31421 **APPLICATION USAGE**

31422 None.

31423 **RATIONALE**31424 The *guardsize* attribute is provided to the application for two reasons:

- 31425 1. Overflow protection can potentially result in wasted system resources. An application  
31426 that creates a large number of threads, and which knows its threads never overflow their  
31427 stack, can save system resources by turning off guard areas.
- 31428 2. When threads allocate large data structures on the stack, large guard areas may be needed  
31429 to detect stack overflow.

31430 **FUTURE DIRECTIONS**

31431 None.

31432 **SEE ALSO**

31433 The Base Definitions volume of IEEE Std. 1003.1-200x, &lt;pthread.h&gt;, &lt;sys/mman.h&gt;

31434 **CHANGE HISTORY**

31435 First released in Issue 5.

31436 **Issue 6**31437 In the ERRORS section, a third [EINVAL] error condition is removed as it is covered by the  
31438 second error condition.31439 The **restrict** keyword is added to the *pthread\_attr\_getguardsize()* prototype for alignment with the  
31440 ISO/IEC 9899:1999 standard.

31441 **NAME**

31442 pthread\_attr\_getinheritsched, pthread\_attr\_setinheritsched — get and set inheritsched attribute  
 31443 (**REALTIME THREADS**)

31444 **SYNOPSIS**

31445 TPS #include <pthread.h>

```
31446 int pthread_attr_getinheritsched(const pthread_attr_t *restrict attr,
31447 int *restrict inheritsched);
```

```
31448 int pthread_attr_setinheritsched(pthread_attr_t *attr,
31449 int inheritsched);
```

31450

31451 **DESCRIPTION**

31452 The *pthread\_attr\_getinheritsched()*, and *pthread\_attr\_setinheritsched()* functions, respectively, get  
 31453 and set the *inheritsched* attribute in the *attr* argument.

31454 When the attributes objects are used by *pthread\_create()*, the *inheritsched* attribute determines  
 31455 how the other scheduling attributes of the created thread shall be set:

31456 **PTHREAD\_INHERIT\_SCHED**

31457 Specifies that the scheduling policy and associated attributes shall be inherited from the  
 31458 creating thread, and the scheduling attributes in this *attr* argument shall be ignored.

31459 **PTHREAD\_EXPLICIT\_SCHED**

31460 Specifies that the scheduling policy and associated attributes shall be set to the  
 31461 corresponding values from this attributes object.

31462 The symbols **PTHREAD\_INHERIT\_SCHED** and **PTHREAD\_EXPLICIT\_SCHED** are defined in  
 31463 the header <**pthread.h**>.

31464 **RETURN VALUE**

31465 If successful, the *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions shall  
 31466 return zero; otherwise, an error number shall be returned to indicate the error.

31467 **ERRORS**

31468 The *pthread\_attr\_setinheritsched()* function may fail if:

31469 [EINVAL] The value of *inheritsched* is not valid.

31470 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

31471 These functions shall not return an error code of [EINTR].

31472 **EXAMPLES**

31473 None.

31474 **APPLICATION USAGE**

31475 After these attributes have been set, a thread can be created with the specified attributes using  
 31476 *pthread\_create()*. Using these routines does not affect the current running thread.

31477 **RATIONALE**

31478 None.

31479 **FUTURE DIRECTIONS**

31480 None.

31481 **SEE ALSO**

31482 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getschedpolicy()*,  
31483 *pthread\_attr\_getschedparam()*, *pthread\_create()*, the Base Definitions volume of  
31484 IEEE Std. 1003.1-200x, <pthread.h>, <sched.h>

31485 **CHANGE HISTORY**

31486 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31487 Marked as part of the Realtime Threads Feature Group.

31488 **Issue 6**

31489 The *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions are marked as part  
31490 of the Thread Execution Scheduling option.

31491 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
31492 implementation does not support the Thread Execution Scheduling option.

31493 The **restrict** keyword is added to the *pthread\_attr\_getinheritsched()* prototype for alignment with  
31494 the ISO/IEC 9899:1999 standard.

31495 **NAME**

31496 pthread\_attr\_getschedparam, pthread\_attr\_setschedparam — get and set schedparam attribute

31497 **SYNOPSIS**

31498 THR #include <pthread.h>

```
31499 int pthread_attr_getschedparam(const pthread_attr_t *restrict attr,
31500 struct sched_param *restrict param);
```

```
31501 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
31502 const struct sched_param *restrict param);
```

31503

31504 **DESCRIPTION**

31505 The *pthread\_attr\_getschedparam()*, and *pthread\_attr\_setschedparam()* functions, respectively, get  
31506 and set the scheduling parameter attributes in the *attr* argument. The contents of the *param*  
31507 structure are defined in <sched.h>. For the SCHED\_FIFO and SCHED\_RR policies, the only  
31508 required member of *param* is *sched\_priority*.

31509 TSP For the SCHED\_SPORADIC policy, the required members of the *param* structure are  
31510 *sched\_priority*, *sched\_ss\_low\_priority*, *sched\_ss\_repl\_period*, *sched\_ss\_init\_budget*, and  
31511 *sched\_ss\_max\_repl*. The specified *sched\_ss\_repl\_period* must be greater than or equal to the  
31512 specified *sched\_ss\_init\_budget* for the function to succeed; if it is not, then the function shall fail.  
31513 The value of *sched\_ss\_max\_repl* shall be within the inclusive range [1,{SS\_REPL\_MAX}] for the  
31514 function to succeed; if not, the function shall fail.

31515 **RETURN VALUE**

31516 If successful, the *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions shall  
31517 return zero; otherwise, an error number shall be returned to indicate the error.

31518 **ERRORS**

31519 The *pthread\_attr\_setschedparam()* function may fail if:

31520 [EINVAL] The value of *param* is not valid.

31521 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

31522 These functions shall not return an error code of [EINTR].

31523 **EXAMPLES**

31524 None.

31525 **APPLICATION USAGE**

31526 After these attributes have been set, a thread can be created with the specified attributes using  
31527 *pthread\_create()*. Using these routines does not affect the current running thread.

31528 **RATIONALE**

31529 None.

31530 **FUTURE DIRECTIONS**

31531 None.

31532 **SEE ALSO**

31533 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getinheritsched()*,  
31534 *pthread\_attr\_getschedpolicy()*, *pthread\_create()*, the Base Definitions volume of  
31535 IEEE Std. 1003.1-200x, <pthread.h>, <sched.h>

## 31536 CHANGE HISTORY

31537 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

## 31538 Issue 6

31539 The *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions are marked as part  
31540 of the Threads option.

31541 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std. 1003.1d-1999.

31542 The **restrict** keyword is added to the *pthread\_attr\_getschedparam()* and  
31543 *pthread\_attr\_setschedparam()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

31544 **NAME**

31545 pthread\_attr\_getschedpolicy, pthread\_attr\_setschedpolicy — get and set schedpolicy attribute  
 31546 (**REALTIME THREADS**)

31547 **SYNOPSIS**

```
31548 TPS #include <pthread.h>
31549
31549 int pthread_attr_getschedpolicy(const pthread_attr_t *restrict attr,
31550 int *restrict policy);
31551 int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
31552
```

31553 **DESCRIPTION**

31554 The *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions, respectively, get and  
 31555 set the *schedpolicy* attribute in the *attr* argument.

31556 The supported values of *policy* shall include *SCHED\_FIFO*, *SCHED\_RR*, and *SCHED\_OTHER*,  
 31557 which are defined by the header *<sched.h>*. When threads executing with the scheduling policy  
 31558 *SCHED\_FIFO*, *SCHED\_RR*, or *SCHED\_SPORADIC* are waiting on a mutex, they acquire the  
 31559 mutex in priority order when the mutex is unlocked.

31560 **RETURN VALUE**

31561 If successful, the *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions shall  
 31562 return zero; otherwise, an error number shall be returned to indicate the error.

31563 **ERRORS**

31564 The *pthread\_attr\_setschedpolicy()* function may fail if:

31565 [EINVAL] The value of *policy* is not valid.  
 31566 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.  
 31567 These functions shall not return an error code of [EINTR].

31568 **EXAMPLES**

31569 None.

31570 **APPLICATION USAGE**

31571 After these attributes have been set, a thread can be created with the specified attributes using  
 31572 *pthread\_create()*. Using these routines does not affect the current running thread.

31573 **RATIONALE**

31574 None.

31575 **FUTURE DIRECTIONS**

31576 None.

31577 **SEE ALSO**

31578 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getinheritsched()*,  
 31579 *pthread\_attr\_getschedparam()*, *pthread\_create()*, the Base Definitions volume of  
 31580 IEEE Std. 1003.1-200x, *<pthread.h>*, *<sched.h>*

31581 **CHANGE HISTORY**

31582 First released in Issue 5. Included for alignment with the POSIX Threads Extension.  
 31583 Marked as part of the Realtime Threads Feature Group.

31584 **Issue 6**

31585 The *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions are marked as part of  
31586 the Thread Execution Scheduling option.

31587 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
31588 implementation does not support the Thread Execution Scheduling option.

31589 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std. 1003.1d-1999.

31590 The **restrict** keyword is added to the *pthread\_attr\_getschedpolicy()* prototype for alignment with  
31591 the ISO/IEC 9899:1999 standard.



31592 **NAME**

31593 pthread\_attr\_getscope, pthread\_attr\_setscope — get and set contention scope attribute  
 31594 (**REALTIME THREADS**)

31595 **SYNOPSIS**

```
31596 TPS #include <pthread.h>
31597
31597 int pthread_attr_getscope(const pthread_attr_t *restrict attr,
31598 int *restrict contention_scope);
31599 int pthread_attr_setscope(pthread_attr_t *attr, int contention_scope);
31600
```

31601 **DESCRIPTION**

31602 The *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions, respectively, are used to get  
 31603 and set the *contention\_scope* attribute in the *attr* object.

31604 The *contention\_scope* attribute may have the values `PTHREAD_SCOPE_SYSTEM`, signifying  
 31605 system scheduling contention scope, or `PTHREAD_SCOPE_PROCESS`, signifying process  
 31606 scheduling contention scope. The symbols `PTHREAD_SCOPE_SYSTEM` and  
 31607 `PTHREAD_SCOPE_PROCESS` are defined by the header `<pthread.h>`.

31608 **RETURN VALUE**

31609 If successful, the *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions shall return zero;  
 31610 otherwise, an error number shall be returned to indicate the error.

31611 **ERRORS**

31612 The *pthread\_attr\_setscope()* function may fail if:

- 31613 [EINVAL] The value of *contention\_scope* is not valid.
- 31614 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.
- 31615 These functions shall not return an error code of [EINTR].

31616 **EXAMPLES**

31617 None.

31618 **APPLICATION USAGE**

31619 After these attributes have been set, a thread can be created with the specified attributes using  
 31620 *pthread\_create()*. Using these routines does not affect the current running thread.

31621 **RATIONALE**

31622 None.

31623 **FUTURE DIRECTIONS**

31624 None.

31625 **SEE ALSO**

31626 *pthread\_attr\_destroy()*, *pthread\_attr\_getinheritsched()*, *pthread\_attr\_getschedpolicy()*,  
 31627 *pthread\_attr\_getschedparam()*, *pthread\_create()*, the Base Definitions volume of  
 31628 IEEE Std. 1003.1-200x, `<pthread.h>`, `<sched.h>`

31629 **CHANGE HISTORY**

- 31630 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
- 31631 Marked as part of the Realtime Threads Feature Group.

31632 **Issue 6**

31633 The *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions are marked as part of the  
31634 Thread Execution Scheduling option.

31635 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
31636 implementation does not support the Thread Execution Scheduling option.

31637 The **restrict** keyword is added to the *pthread\_attr\_getscope()* prototype for alignment with the  
31638 ISO/IEC 9899:1999 standard.

31639 **NAME**

31640 pthread\_attr\_getstack, pthread\_attr\_setstack — get and set stack attribute

31641 **SYNOPSIS**31642 XSI 

```
#include <pthread.h>
```

31643 

```
int pthread_attr_getstack(const pthread_attr_t *attr, void **stackaddr,
31644 size_t *stacksize);
```

31645 

```
int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
31646 size_t stacksize);
```

31647

31648 **DESCRIPTION**31649 The *pthread\_attr\_getstack()* and *pthread\_attr\_setstack()* functions, respectively, shall get and set  
31650 the thread creation stack attribute in the *attr* object.31651 The *stack* attribute specifies the area of storage to be used for the created thread's stack. The base  
31652 (lowest addressable byte) of the storage is *stackaddr*, and the size of the storage is *stacksize* bytes.  
31653 The *stacksize* shall be at least {PTHREAD\_STACK\_MIN}. The *stackaddr* shall be aligned  
31654 appropriately to be used as a stack; for example, *pthread\_attr\_setstack()* may fail with [EINVAL]  
31655 if (*stackaddr* & 0x7) is not 0. All pages within the stack described by *stackaddr* and *stacksize* shall be  
31656 both readable and writable by the thread.31657 **RETURN VALUE**31658 Upon successful completion, these functions shall return a value of 0; otherwise, an error  
31659 number shall be returned to indicate the error.31660 The *pthread\_attr\_getstack()* function shall store the stack attribute values in *stackaddr* and *stacksize*  
31661 if successful.31662 **ERRORS**31663 The *pthread\_attr\_setstack()* function shall fail if:31664 [EINVAL] The value of *stacksize* is less than {PTHREAD\_STACK\_MIN} or exceeds an  
31665 implementation-defined limit.31666 The *pthread\_attr\_setstack()* function may fail if:31667 [EINVAL] The value of *stackaddr* does not have proper alignment to be used as a stack, or  
31668 if (*stackaddr* + *stacksize*) lacks proper alignment.31669 [EACCES] The stack page(s) described by *stackaddr* and *stacksize* are not both readable  
31670 and writable by the thread.

31671 These functions shall not return an error code of [EINTR].

31672 **EXAMPLES**

31673 None.

31674 **APPLICATION USAGE**31675 These functions are appropriate for use by applications in an environment where the stack for a  
31676 thread must be placed in some particular region of memory.31677 While it might seem that an application could detect stack overflow by providing a protected  
31678 page outside the specified stack region, this cannot be done portably. Implementations are free  
31679 to place the thread's initial stack pointer anywhere within the specified region to accommodate  
31680 the machine's stack pointer behavior and allocation requirements. Furthermore, on some  
31681 architectures, such as the IA-64, "overflow" might mean that two separate stack pointers  
31682 allocated within the region will overlap somewhere in the middle of the region.

31683 **RATIONALE**

31684           None.

31685 **FUTURE DIRECTIONS**

31686           None.

31687 **SEE ALSO**

31688           *pthread\_attr\_init()*, *pthread\_attr\_setdetachstate()*, *pthread\_attr\_setstacksize()*, *pthread\_create()*, the

31689           Base Definitions volume of IEEE Std. 1003.1-200x, <limits.h>, <pthread.h>

31690 **CHANGE HISTORY**

31691           First released in Issue 6.

31692 **NAME**

31693 pthread\_attr\_getstackaddr, pthread\_attr\_setstackaddr — get and set stackaddr attribute

31694 **SYNOPSIS**

31695 THR TSA #include <pthread.h>

```
31696 int pthread_attr_getstackaddr(const pthread_attr_t *restrict attr,
31697 void **restrict stackaddr);
31698 int pthread_attr_setstackaddr(pthread_attr_t *attr, void *stackaddr);
31699
```

31700 **DESCRIPTION**

31701 The *pthread\_attr\_getstackaddr()* and *pthread\_attr\_setstackaddr()* functions, respectively, get and set  
31702 the thread creation *stackaddr* attribute in the *attr* object.

31703 The *stackaddr* attribute specifies the location of storage to be used for the created thread's stack.  
31704 The size of the storage is at least {PTHREAD\_STACK\_MIN}.

31705 **RETURN VALUE**

31706 Upon successful completion, *pthread\_attr\_getstackaddr()* and *pthread\_attr\_setstackaddr()* shall  
31707 return a value of 0; otherwise, an error number shall be returned to indicate the error.

31708 The *pthread\_attr\_getstackaddr()* function stores the *stackaddr* attribute value in *stackaddr* if  
31709 successful.

31710 **ERRORS**

31711 No errors are defined.

31712 These functions shall not return an error code of [EINTR].

31713 **EXAMPLES**

31714 None.

31715 **APPLICATION USAGE**

31716 None.

31717 **RATIONALE**

31718 None.

31719 **FUTURE DIRECTIONS**

31720 None.

31721 **SEE ALSO**

31722 *pthread\_attr\_destroy()*, *pthread\_attr\_getdetachstate()*, *pthread\_attr\_getstacksize()*, *pthread\_create()*,  
31723 the Base Definitions volume of IEEE Std. 1003.1-200x, <limits.h>, <pthread.h>

31724 **CHANGE HISTORY**

31725 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31726 **Issue 6**

31727 The *pthread\_attr\_getstackaddr()* and *pthread\_attr\_setstackaddr()* functions are marked as part of  
31728 the Threads and Thread Stack Address Attribute options.

31729 The **restrict** keyword is added to the *pthread\_attr\_getstackaddr()* prototype for alignment with the  
31730 ISO/IEC 9899:1999 standard.

31731 **NAME**

31732 pthread\_attr\_getstacksize, pthread\_attr\_setstacksize — get and set stacksize attribute

31733 **SYNOPSIS**

31734 THR TSA #include &lt;pthread.h&gt;

```

31735 int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
31736 size_t *restrict stacksize);
31737 int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
31738

```

31739 **DESCRIPTION**31740 The *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* functions, respectively, get and set  
31741 the thread creation *stacksize* attribute in the *attr* object.31742 The *stacksize* attribute defines the minimum stack size (in bytes) allocated for the created threads  
31743 stack.31744 **RETURN VALUE**31745 Upon successful completion, *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* shall  
31746 return a value of 0; otherwise, an error number shall be returned to indicate the error.31747 The *pthread\_attr\_getstacksize()* function stores the *stacksize* attribute value in *stacksize* if  
31748 successful.31749 **ERRORS**31750 The *pthread\_attr\_setstacksize()* function shall fail if:31751 [EINVAL] The value of *stacksize* is less than {PTHREAD\_STACK\_MIN} or exceeds a  
31752 system-imposed limit.

31753 These functions shall not return an error code of [EINTR].

31754 **EXAMPLES**

31755 None.

31756 **APPLICATION USAGE**

31757 None.

31758 **RATIONALE**

31759 None.

31760 **FUTURE DIRECTIONS**

31761 None.

31762 **SEE ALSO**31763 *pthread\_attr\_destroy()*, *pthread\_attr\_getstackaddr()*, *pthread\_attr\_getdetachstate()*, *pthread\_create()*,  
31764 the Base Definitions volume of IEEE Std. 1003.1-200x, <limits.h>, <pthread.h>31765 **CHANGE HISTORY**

31766 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31767 **Issue 6**31768 The *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* functions are marked as part of the  
31769 Threads and Thread Stack Address Attribute options.31770 The **restrict** keyword is added to the *pthread\_attr\_getstacksize()* prototype for alignment with the  
31771 ISO/IEC 9899:1999 standard.

31772 **NAME**

31773 pthread\_attr\_init — initialize threads attributes object

31774 **SYNOPSIS**

31775 THR #include &lt;pthread.h&gt;

31776 int pthread\_attr\_init(pthread\_attr\_t \*attr);

31777

31778 **DESCRIPTION**31779 Refer to *pthread\_attr\_destroy()*.

31780 **NAME**

31781 pthread\_attr\_setdetachstate — set detachstate attribute

31782 **SYNOPSIS**

31783 THR #include <pthread.h>

31784 int pthread\_attr\_setdetachstate(pthread\_attr\_t \*attr, int detachstate);

31785

31786 **DESCRIPTION**

31787 Refer to *pthread\_attr\_getdetachstate()*.



31788 **NAME**

31789 pthread\_attr\_setguardsize — set thread guardsize attribute

31790 **SYNOPSIS**

31791 xSI #include &lt;pthread.h&gt;

31792 int pthread\_attr\_setguardsize(pthread\_attr\_t \*attr,  
31793 size\_t guardsize);

31794

31795 **DESCRIPTION**31796 Refer to *pthread\_attr\_getguardsize()*.

31797 **NAME**

31798 pthread\_attr\_setinheritsched — set inheritsched attribute (**REALTIME THREADS**)

31799 **SYNOPSIS**

31800 TPS #include <pthread.h>

```
31801 int pthread_attr_setinheritsched(pthread_attr_t *attr,
31802 int inheritsched);
```

31803

31804 **DESCRIPTION**

31805 Refer to *pthread\_attr\_getinheritsched()*.

31806 **NAME**

31807 pthread\_attr\_setschedparam — set schedparam attribute

31808 **SYNOPSIS**

31809 THR #include &lt;pthread.h&gt;

31810 int pthread\_attr\_setschedparam(pthread\_attr\_t \*restrict attr,  
31811 const struct sched\_param \*restrict param);

31812

31813 **DESCRIPTION**31814 Refer to *pthread\_attr\_getschedparam()*.

31815 **NAME**

31816 pthread\_attr\_setschedpolicy — set schedpolicy attribute (**REALTIME THREADS**)

31817 **SYNOPSIS**

31818 TPS #include <pthread.h>

31819 int pthread\_attr\_setschedpolicy(pthread\_attr\_t \*attr, int policy);

31820

31821 **DESCRIPTION**

31822 Refer to *pthread\_attr\_getschedpolicy()*.

31823 **NAME**

31824 pthread\_attr\_setscope — set contentionscope attribute (**REALTIME THREADS**)

31825 **SYNOPSIS**

31826 TPS `#include <pthread.h>`

31827 `int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);`

31828

31829 **DESCRIPTION**

31830 Refer to *pthread\_attr\_getscope()*.

31831 **NAME**

31832 pthread\_attr\_setstack — set stack attribute

31833 **SYNOPSIS**

31834 XSI #include <pthread.h>

```
31835 int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
31836 size_t stacksize);
```

31837

31838 **DESCRIPTION**

31839 Refer to *pthread\_attr\_getstack()*.

31840 **NAME**

31841 pthread\_attr\_setstackaddr — set stackaddr attribute

31842 **SYNOPSIS**

31843 THR TSA #include &lt;pthread.h&gt;

31844 int pthread\_attr\_setstackaddr(pthread\_attr\_t \*attr, void \*stackaddr);

31845

31846 **DESCRIPTION**31847 Refer to *pthread\_attr\_getstackaddr()*.

31848 **NAME**

31849 pthread\_attr\_setstacksize — set stacksize attribute

31850 **SYNOPSIS**

31851 THR TSA #include <pthread.h>

31852 int pthread\_attr\_setstacksize(pthread\_attr\_t \*attr, size\_t stacksize);

31853

31854 **DESCRIPTION**

31855 Refer to *pthread\_attr\_getstacksize()*.



## 31856 NAME

31857 pthread\_barrier\_destroy, pthread\_barrier\_init — destroy and initialize a barrier object

## 31858 SYNOPSIS

31859 BAR #include &lt;pthread.h&gt;

```

31860 int pthread_barrier_destroy(pthread_barrier_t *barrier);
31861 int pthread_barrier_init(pthread_barrier_t *restrict barrier,
31862 const pthread_barrierattr_t *restrict attr, unsigned count);
31863

```

## 31864 DESCRIPTION

31865 The *pthread\_barrier\_destroy()* function destroys the barrier referenced by *barrier* and releases any  
 31866 resources used by the barrier. The effect of subsequent use of the barrier is undefined until the  
 31867 barrier is reinitialized by another call to *pthread\_barrier\_init()*. An implementation may use this  
 31868 function to set *barrier* to an invalid value. The results are undefined if *pthread\_barrier\_destroy()* is  
 31869 called when any thread is blocked on the barrier, or if this function is called with an uninitialized  
 31870 barrier.

31871 The *pthread\_barrier\_init()* function shall allocate any resources required to use the barrier  
 31872 referenced by *barrier* and initializes the barrier with attributes referenced by *attr*. If *attr* is NULL,  
 31873 the default barrier attributes are used; the effect is the same as passing the address of a default  
 31874 barrier attributes object. The results are undefined if *pthread\_barrier\_init()* is called when any  
 31875 thread is blocked on the barrier (that is, has not returned from the *pthread\_barrier\_wait()* call).  
 31876 The results are undefined if a barrier is used without first being initialized. The results are  
 31877 undefined if *pthread\_barrier\_init()* is called specifying an already initialized barrier.

31878 The *count* argument specifies the number of threads that must call *pthread\_barrier\_wait()* before  
 31879 any of them successfully return from the call. The value specified by *count* must be greater than  
 31880 zero.

31881 If the *pthread\_barrier\_init()* function fails, the barrier is not initialized and the contents of *barrier*  
 31882 are undefined.

31883 Only the object referenced by *barrier* may be used for performing synchronization. The result of  
 31884 referring to copies of that object in calls to *pthread\_barrier\_destroy()* or *pthread\_barrier\_wait()* is  
 31885 undefined.

## 31886 RETURN VALUE

31887 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 31888 be returned to indicate the error.

## 31889 ERRORS

31890 The *pthread\_barrier\_destroy()* function may fail if:

31891 [EBUSY] The implementation has detected an attempt to destroy a barrier while it is in  
 31892 use (for example, while being used in a *pthread\_barrier\_wait()* call) by another  
 31893 thread.

31894 [EINVAL] The value specified by *barrier* is invalid.

31895 The *pthread\_barrier\_init()* function shall fail if:

31896 [EAGAIN] The system lacks the necessary resources to initialize another barrier.

31897 [EINVAL] The value specified by *count* is equal to zero.

31898 [ENOMEM] Insufficient memory exists to initialize the barrier.

- 31899 The *pthread\_barrier\_init()* function may fail if:
- 31900 [EBUSY] The implementation has detected an attempt to reinitialize a barrier while it is  
31901 in use (for example, while being used in a *pthread\_barrier\_wait()* call) by  
31902 another thread.
- 31903 [EINVAL] The value specified by *attr* is invalid.
- 31904 These functions shall not return an error code of [EINTR].
- 31905 **EXAMPLES**
- 31906 None.
- 31907 **APPLICATION USAGE**
- 31908 The *pthread\_barrier\_destroy()* and *pthread\_barrier\_init()* functions are part of the Barriers option  
31909 and need not be provided on all implementations.
- 31910 **RATIONALE**
- 31911 None.
- 31912 **FUTURE DIRECTIONS**
- 31913 None.
- 31914 **SEE ALSO**
- 31915 *pthread\_barrier\_wait()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>
- 31916 **CHANGE HISTORY**
- 31917 First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.

31918 **NAME**

31919 pthread\_barrier\_init — initialize a barrier object

31920 **SYNOPSIS**

31921 BAR #include &lt;pthread.h&gt;

31922 int pthread\_barrier\_init(pthread\_barrier\_t \*restrict *barrier*,  
31923 const pthread\_barrierattr\_t \*restrict *attr*, unsigned *count*);

31924

31925 **DESCRIPTION**31926 Refer to *pthread\_barrier\_destroy()*.

31927 **NAME**

31928 pthread\_barrier\_wait — synchronize at a barrier

31929 **SYNOPSIS**31930 **BAR** #include <pthread.h>

31931 int pthread\_barrier\_wait(pthread\_barrier\_t \*barrier);

31932

31933 **DESCRIPTION**

31934 The *pthread\_barrier\_wait()* function synchronizes participating threads at the barrier referenced  
31935 by *barrier*. The calling thread blocks (that is, does not return from the *pthread\_barrier\_wait()* call)  
31936 until the required number of threads have called *pthread\_barrier\_wait()* specifying the barrier.

31937 When the required number of threads have called *pthread\_barrier\_wait()* specifying the barrier,  
31938 the constant PTHREAD\_BARRIER\_SERIAL\_THREAD is returned to one unspecified thread  
31939 and zero is returned to each of the remaining threads. At this point, the barrier is reset to the  
31940 state it had as a result of the most recent *pthread\_barrier\_init()* function that referenced it.

31941 The constant PTHREAD\_BARRIER\_SERIAL\_THREAD is defined in <pthread.h> and its value  
31942 is distinct from any other value returned by *pthread\_barrier\_wait()*.

31943 The results are undefined if this function is called with an uninitialized barrier.

31944 If a signal is delivered to a thread blocked on a barrier, upon return from the signal handler the  
31945 thread shall resume waiting at the barrier if the barrier wait has not completed (that is, if the  
31946 required number of threads have not arrived at the barrier during the execution of the signal  
31947 handler); otherwise, the thread shall continue as normal from the completed barrier wait. Until  
31948 the thread in the signal handler returns from it, it is unspecified whether other threads may  
31949 proceed past the barrier once they have all reached it.

31950 A thread that has blocked on a barrier shall not prevent any unblocked thread that is eligible to  
31951 use the same processing resources from eventually making forward progress in its execution.  
31952 Eligibility for processing resources shall be determined by the scheduling policy.

31953 **RETURN VALUE**

31954 Upon successful completion, the *pthread\_barrier\_wait()* function shall return  
31955 PTHREAD\_BARRIER\_SERIAL\_THREAD for a single (arbitrary) thread synchronized at the  
31956 barrier and zero for each of the other threads. Otherwise, an error number shall be returned to  
31957 indicate the error.

31958 **ERRORS**

31959 The *pthread\_barrier\_wait()* function may fail if:

31960 [EINVAL] The value specified by *barrier* does not refer to an initialized barrier object.

31961 This function shall not return an error code of [EINTR].

31962 **EXAMPLES**

31963 None.

31964 **APPLICATION USAGE**

31965 Applications using this function may be subject to priority inversion, as discussed in the Base  
31966 Definitions volume of IEEE Std. 1003.1-200x, Section 3.287, Priority Inversion.

31967 The *pthread\_barrier\_wait()* function is part of the Barriers option and need not be provided on all  
31968 implementations.

31969 **RATIONALE**

31970 None.

31971 **FUTURE DIRECTIONS**

31972 None.

31973 **SEE ALSO**31974 *pthread\_barrier\_destroy()*, *pthread\_barrier\_init()*, the Base Definitions volume of |

31975 IEEE Std. 1003.1-200x, &lt;pthread.h&gt; |

31976 **CHANGE HISTORY**

31977 First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.

31978 In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

31979 **NAME**

31980 pthread\_barrierattr\_destroy, pthread\_barrierattr\_init — destroy and initialize barrier attributes  
31981 object

31982 **SYNOPSIS**

```
31983 BAR #include <pthread.h>
```

```
31984 int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);
```

```
31985 int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```

```
31986
```

31987 **DESCRIPTION**

31988 The *pthread\_barrierattr\_destroy()* function destroys a barrier attributes object. The effect of  
31989 subsequent use of the object is undefined until the object is reinitialized by another call to  
31990 *pthread\_barrierattr\_init()*. An implementation may cause *pthread\_barrierattr\_destroy()* to set the  
31991 object referenced by *attr* to an invalid value.

31992 The *pthread\_barrierattr\_init()* function initializes a barrier attributes object *attr* with the default  
31993 value for all of the attributes defined by the implementation.

31994 The results are undefined if *pthread\_barrierattr\_init()* is called specifying an already initialized  
31995 barrier attributes object.

31996 After a barrier attributes object has been used to initialize one or more barriers, any function  
31997 affecting the attributes object (including destruction) does not affect any previously initialized  
31998 barrier.

31999 **RETURN VALUE**

32000 If successful, the *pthread\_barrierattr\_destroy()* and *pthread\_barrierattr\_init()* functions shall return  
32001 zero; otherwise, an error number shall be returned to indicate the error.

32002 **ERRORS**

32003 The *pthread\_barrierattr\_destroy()* function may fail if:

32004 [EINVAL] The value specified by *attr* is invalid.

32005 The *pthread\_barrierattr\_init()* function shall fail if:

32006 [ENOMEM] Insufficient memory exists to initialize the barrier attributes object.

32007 These functions shall not return an error code of [EINTR].

32008 **EXAMPLES**

32009 None.

32010 **APPLICATION USAGE**

32011 The *pthread\_barrierattr\_destroy()* and *pthread\_barrierattr\_init()* functions are part of the Barriers  
32012 option and need not be provided on all implementations.

32013 **RATIONALE**

32014 None.

32015 **FUTURE DIRECTIONS**

32016 None.

32017 **SEE ALSO**

32018 *pthread\_barrierattr\_getshared()*, *pthread\_barrierattr\_setshared()*, the Base Definitions volume of  
32019 IEEE Std. 1003.1-200x, <pthread.h>.

32020 **CHANGE HISTORY**

32021 First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.

32022 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

32023 **NAME**

32024 pthread\_barrierattr\_getpshared, pthread\_barrierattr\_setpshared — get and set process-shared  
 32025 attribute of barrier attributes object

32026 **SYNOPSIS**

32027 BAR TSH #include <pthread.h>

```
32028 int pthread_barrierattr_getpshared(const pthread_barrierattr_t *restrict attr, |
32029 int *restrict pshared); |
32030 int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr, |
32031 int pshared); |
32032
```

32033 **DESCRIPTION**

32034 The process-shared attribute is set to PTHREAD\_PROCESS\_SHARED to permit a barrier to be  
 32035 operated upon by any thread that has access to the memory where the barrier is allocated. If the  
 32036 process-shared attribute is PTHREAD\_PROCESS\_PRIVATE, the barrier shall only be operated  
 32037 upon by threads created within the same process as the thread that initialized the barrier; if  
 32038 threads of different processes attempt to operate on such a barrier, the behavior is undefined.  
 32039 The default value of the attribute shall be PTHREAD\_PROCESS\_PRIVATE. Both constants  
 32040 PTHREAD\_PROCESS\_SHARED and PTHREAD\_PROCESS\_PRIVATE are defined in  
 32041 <pthread.h>.

32042 The *pthread\_barrierattr\_getpshared()* function obtains the value of the process-shared attribute  
 32043 from the attributes object referenced by *attr*. The *pthread\_barrierattr\_setpshared()* function is used  
 32044 to set the process-shared attribute in an initialized attributes object referenced by *attr*.

32045 Additional attributes, their default values, and the names of the associated functions to get and  
 32046 set those attribute values are implementation-defined.

32047 **RETURN VALUE**

32048 If successful, the *pthread\_barrierattr\_getpshared()* function shall return zero and store the value of  
 32049 the process-shared attribute of *attr* into the object referenced by the *pshared* parameter.  
 32050 Otherwise, an error number shall be returned to indicate the error.

32051 If successful, the *pthread\_barrierattr\_setpshared()* function shall return zero; otherwise, an error  
 32052 number shall be returned to indicate the error.

32053 **ERRORS**

32054 These functions may fail if:

32055 [EINVAL] The value specified by *attr* is invalid.

32056 The *pthread\_barrierattr\_setpshared()* function may fail if:

32057 [EINVAL] The new value specified for the process-shared attribute is not one of the legal  
 32058 values PTHREAD\_PROCESS\_SHARED or PTHREAD\_PROCESS\_PRIVATE.

32059 These functions shall not return an error code of [EINTR].



32060 **EXAMPLES**

32061 None.

32062 **APPLICATION USAGE**

32063 The *pthread\_barrierattr\_getpshared()* and *pthread\_barrierattr\_setpshared()* functions are part of the  
32064 Barriers option and need not be provided on all implementations.

32065 **RATIONALE**

32066 None.

32067 **FUTURE DIRECTIONS**

32068 None.

32069 **SEE ALSO**

32070 *pthread\_barrier\_init()*, *pthread\_barrierattr\_destroy()*, *pthread\_barrierattr\_init()*, the Base Definitions  
32071 volume of IEEE Std. 1003.1-200x, <pthread.h>

32072 **CHANGE HISTORY**

32073 First released in Issue 6. Derived from IEEE Std. 1003.1j-2000

32074 **NAME**

32075 pthread\_barrierattr\_init — initialize barrier attributes object

32076 **SYNOPSIS**

32077 BAR #include <pthread.h>

32078 int pthread\_barrierattr\_init(pthread\_barrierattr\_t \*attr);

32079

32080 **DESCRIPTION**

32081 Refer to *pthread\_barrierattr\_destroy()*.

32082 **NAME**

32083 pthread\_barrierattr\_setpshared — set process-shared attribute of barrier attributes object

32084 **SYNOPSIS**

32085 BAR TSH #include &lt;pthread.h&gt;

32086 int pthread\_barrierattr\_setpshared(pthread\_barrierattr\_t \*attr,  
32087 int pshared);

32088

32089 **DESCRIPTION**32090 Refer to *pthread\_barrierattr\_getpshared()*.

32091 **NAME**

32092 pthread\_cancel — cancel execution of a thread

32093 **SYNOPSIS**

32094 THR #include &lt;pthread.h&gt;

32095 int pthread\_cancel(pthread\_t thread);

32096

32097 **DESCRIPTION**

32098 The *pthread\_cancel()* function requests that *thread* be canceled. The target thread's cancelability state and type determines when the cancellation takes effect. When the cancellation is acted on, 32099 the cancellation cleanup handlers for *thread* are called. When the last cancellation cleanup handler 32100 returns, the thread-specific data destructor functions shall be called for *thread*. When the last 32101 destructor function returns, *thread* shall be terminated. 32102

32103 The cancellation processing in the target thread runs asynchronously with respect to the calling 32104 thread returning from *pthread\_cancel()*.

32105 **RETURN VALUE**

32106 If successful, the *pthread\_cancel()* function shall return zero; otherwise, an error number shall be 32107 returned to indicate the error.

32108 **ERRORS**32109 The *pthread\_cancel()* function may fail if:

32110 [ESRCH] No thread could be found corresponding to that specified by the given thread ID. | 32111

32112 The *pthread\_cancel()* function shall not return an error code of [EINTR]. |32113 **EXAMPLES**

32114 None.

32115 **APPLICATION USAGE**

32116 None.

32117 **RATIONALE**

32118 Two alternative functions were considered to sending the cancellation notification to a thread. 32119 One would be to define a new SIGCANCEL signal that had the cancellation semantics when 32120 delivered; the other was to define the new *pthread\_cancel()* function, which would trigger the 32121 cancellation semantics.

32122 The advantage of a new signal was that so much of the delivery criteria were identical to that 32123 used when trying to deliver a signal that making cancellation notification a signal was seen as 32124 consistent. Indeed, many implementations implement cancellation using a special signal. On the 32125 other hand, there would be no signal functions that could be used with this signal except 32126 *pthread\_kill()*, and the behavior of the delivered cancellation signal would be unlike any 32127 previously existing defined signal.

32128 The benefits of a special function include the recognition that this signal would be defined 32129 because of the similar delivery criteria and that this is the only common behavior between a 32130 cancellation request and a signal. In addition, the cancellation delivery mechanism does not have 32131 to be implemented as a signal. There are also strong, if not stronger, parallels with language 32132 exception mechanisms than with signals that are potentially obscured if the delivery mechanism 32133 is visibly closer to signals.

32134 In the end, it was considered that as there were so many exceptions to the use of the new signal 32135 with existing signals functions that it would be misleading. A special function has resolved this

32136 problem. This function was carefully defined so that an implementation wishing to provide the  
32137 cancelation functions on top of signals could do so. The special function also means that  
32138 implementations are not obliged to implement cancelation with signals.

32139 **FUTURE DIRECTIONS**

32140 None.

32141 **SEE ALSO**

32142 *pthread\_exit()*, *pthread\_cond\_wait()*, *pthread\_cond\_timedwait()*, *pthread\_join()*,  
32143 *pthread\_setcancelstate()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

32144 **CHANGE HISTORY**

32145 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32146 **Issue 6**

32147 The *pthread\_cancel()* function is marked as part of the Threads option.

32148 **NAME**

32149 pthread\_cleanup\_pop, pthread\_cleanup\_push — establish cancelation handlers

32150 **SYNOPSIS**

32151 THR #include &lt;pthread.h&gt;

32152 void pthread\_cleanup\_pop(int *execute*);32153 void pthread\_cleanup\_push(void (\**routine*)(void\*), void \**arg*);

32154

32155 **DESCRIPTION**32156 The *pthread\_cleanup\_pop()* function shall remove the routine at the top of the calling thread's  
32157 cancelation cleanup stack and optionally invoke it (if *execute* is non-zero).32158 The *pthread\_cleanup\_push()* function shall push the specified cancelation cleanup handler *routine*  
32159 onto the calling thread's cancelation cleanup stack. The cancelation cleanup handler shall be  
32160 popped from the cancelation cleanup stack and invoked with the argument *arg* when:

- 32161 • The thread exits (that is, calls *pthread\_exit()*).
- 32162 • The thread acts upon a cancelation request.
- 32163 • The thread calls *pthread\_cleanup\_pop()* with a non-zero *execute* argument.

32164 These functions may be implemented as macros. The application shall ensure that they appear  
32165 as statements, and in pairs within the same lexical scope (that is, the *pthread\_cleanup\_push()*  
32166 macro may be thought to expand to a token list whose first token is '{' with  
32167 *pthread\_cleanup\_pop()* expanding to a token list whose last token is the corresponding '}').32168 The effect of calling *longjmp()* or *siglongjmp()* is undefined if there have been any calls to  
32169 *pthread\_cleanup\_push()* or *pthread\_cleanup\_pop()* made without the matching call since the jump  
32170 buffer was filled. The effect of calling *longjmp()* or *siglongjmp()* from inside a cancelation  
32171 cleanup handler is also undefined unless the jump buffer was also filled in the cancelation  
32172 cleanup handler.32173 **RETURN VALUE**32174 The *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()* functions shall return no value.32175 **ERRORS**

32176 No errors are defined.

32177 These functions shall not return an error code of [EINTR].

32178 **EXAMPLES**32179 The following is an example using thread primitives to implement a cancelable, writers-priority  
32180 read-write lock:

```

32181 typedef struct {
32182 pthread_mutex_t lock;
32183 pthread_cond_t rcond,
32184 wcond;
32185 int lock_count; /* < 0 .. Held by writer. */
32186 /* > 0 .. Held by lock_count readers. */
32187 /* = 0 .. Held by nobody. */
32188 int waiting_writers; /* Count of waiting writers. */
32189 } rwlock;
32190
32191 void
32192 waiting_reader_cleanup(void *arg)
32193 {

```

```

32193 rwlock *l;
32194 l = (rwlock *) arg;
32195 pthread_mutex_unlock(&l->lock);
32196 }
32197 void
32198 lock_for_read(rwlock *l)
32199 {
32200 pthread_mutex_lock(&l->lock);
32201 pthread_cleanup_push(waiting_reader_cleanup, l);
32202 while ((l->lock_count < 0) && (l->waiting_writers != 0))
32203 pthread_cond_wait(&l->rcond, &l->lock);
32204 l->lock_count++;
32205 /*
32206 * Note the pthread_cleanup_pop executes
32207 * waiting_reader_cleanup.
32208 */
32209 pthread_cleanup_pop(1);
32210 }
32211 void
32212 release_read_lock(rwlock *l)
32213 {
32214 pthread_mutex_lock(&l->lock);
32215 if (--l->lock_count == 0)
32216 pthread_cond_signal(&l->wcond);
32217 pthread_mutex_unlock(l);
32218 }
32219 void
32220 waiting_writer_cleanup(void *arg)
32221 {
32222 rwlock *l;
32223 l = (rwlock *) arg;
32224 if ((--l->waiting_writers == 0) && (l->lock_count >= 0)) {
32225 /*
32226 * This only happens if we have been canceled.
32227 */
32228 pthread_cond_broadcast(&l->wcond);
32229 }
32230 pthread_mutex_unlock(&l->lock);
32231 }
32232 void
32233 lock_for_write(rwlock *l)
32234 {
32235 pthread_mutex_lock(&l->lock);
32236 l->waiting_writers++;
32237 pthread_cleanup_push(waiting_writer_cleanup, l);
32238 while (l->lock_count != 0)
32239 pthread_cond_wait(&l->wcond, &l->lock);
32240 l->lock_count = -1;
32241 /*

```

```
32242 * Note the pthread_cleanup_pop executes
32243 * waiting_writer_cleanup.
32244 */
32245 pthread_cleanup_pop(1);
32246 }

32247 void
32248 release_write_lock(rwlock *l)
32249 {
32250 pthread_mutex_lock(&l->lock);
32251 l->lock_count = 0;
32252 if (l->waiting_writers == 0)
32253 pthread_cond_broadcast(&l->rcond)
32254 else
32255 pthread_cond_signal(&l->wcond);
32256 pthread_mutex_unlock(&l->lock);
32257 }

32258 /*
32259 * This function is called to initialize the read/write lock.
32260 */
32261 void
32262 initialize_rwlock(rwlock *l)
32263 {
32264 pthread_mutex_init(&l->lock, pthread_mutexattr_default);
32265 pthread_cond_init(&l->wcond, pthread_condattr_default);
32266 pthread_cond_init(&l->rcond, pthread_condattr_default);
32267 l->lock_count = 0;
32268 l->waiting_writers = 0;
32269 }

32270 reader_thread()
32271 {
32272 lock_for_read(&lock);
32273 pthread_cleanup_push(release_read_lock, &lock);
32274 /*
32275 * Thread has read lock.
32276 */
32277 pthread_cleanup_pop(1);
32278 }

32279 writer_thread()
32280 {
32281 lock_for_write(&lock);
32282 pthread_cleanup_push(release_write_lock, &lock);
32283 /*
32284 * Thread has write lock.
32285 */
32286 pthread_cleanup_pop(1);
32287 }
```



32288 **APPLICATION USAGE**

32289 The two routines that push and pop cancellation cleanup handlers, *pthread\_cleanup\_push()* and  
 32290 *pthread\_cleanup\_pop()*, can be thought of as left and right parentheses. They always need to be  
 32291 matched.

32292 **RATIONALE**

32293 The restriction that the two routines that push and pop cancellation cleanup handlers,  
 32294 *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()*, have to appear in the same lexical scope  
 32295 allows for efficient macro or compiler implementations and efficient storage management. A  
 32296 sample implementation of these routines as macros might look like this:

```
32297 #define pthread_cleanup_push(rtn,arg) { \
32298 struct _pthread_handler_rec __cleanup_handler, **__head; \
32299 __cleanup_handler.rtn = rtn; \
32300 __cleanup_handler.arg = arg; \
32301 (void) pthread_getspecific(_pthread_handler_key, &__head); \
32302 __cleanup_handler.next = *__head; \
32303 *__head = &__cleanup_handler;
32304 #define pthread_cleanup_pop(ex) \
32305 *__head = __cleanup_handler.next; \
32306 if (ex) (*__cleanup_handler.rtn)(__cleanup_handler.arg); \
32307 }
```

32308 A more ambitious implementation of these routines might do even better by allowing the  
 32309 compiler to note that the cancellation cleanup handler is a constant and can be expanded inline.

32310 This volume of IEEE Std. 1003.1-200x currently leaves unspecified the effect of calling *longjmp()*  
 32311 from a signal handler executing in a POSIX System Interfaces function. If an implementation  
 32312 wants to allow this and give the programmer reasonable behavior, the *longjmp()* function has to  
 32313 call all cancellation cleanup handlers that have been pushed but not popped since the time  
 32314 *setjmp()* was called.

32315 Consider a multi-threaded function called by a thread that uses signals. If a signal were  
 32316 delivered to a signal handler during the operation of *qsort()* and that handler were to call  
 32317 *longjmp()* (which, in turn, did *not* call the cancellation cleanup handlers) the helper threads  
 32318 created by the *qsort()* function would not be canceled. Instead, they would continue to execute  
 32319 and write into the argument array even though the array might have been popped off of the  
 32320 stack.

32321 Note that the specified cleanup handling mechanism is especially tied to the C language and,  
 32322 while the requirement for a uniform mechanism for expressing cleanup is language-  
 32323 independent, the mechanism used in other languages may be quite different. In addition, this  
 32324 mechanism is really only necessary due to the lack of a real exception mechanism in the C  
 32325 language, which would be the ideal solution.

32326 There is no notion of a cancellation cleanup-safe function. If an application has no cancellation  
 32327 points in its signal handlers, blocks any signal whose handler may have cancellation points while  
 32328 calling async-unsafe functions, or disables cancellation while calling async-unsafe functions, all  
 32329 functions may be safely called from cancellation cleanup routines.

32330 **FUTURE DIRECTIONS**

32331 None.

32332 **SEE ALSO**

32333        *pthread\_cancel()*, *pthread\_setcancelstate()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
32334        <pthread.h>

32335 **CHANGE HISTORY**

32336        First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32337 **Issue 6**

32338        The *pthread\_cleanup\_pop()* and *pthread\_cleanup\_push()* functions are marked as part of the  
32339        Threads option.

32340        The APPLICATION USAGE section is added.

32341        The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

32342 **NAME**

32343 pthread\_cond\_broadcast, pthread\_cond\_signal — broadcast or signal a condition

32344 **SYNOPSIS**

32345 THR #include &lt;pthread.h&gt;

32346 int pthread\_cond\_broadcast(pthread\_cond\_t \*cond);

32347 int pthread\_cond\_signal(pthread\_cond\_t \*cond);

32348

32349 **DESCRIPTION**

32350 These functions are used to unblock threads blocked on a condition variable.

32351 The *pthread\_cond\_broadcast()* function shall unblock all threads currently blocked on the  
32352 specified condition variable *cond*.32353 The *pthread\_cond\_signal()* function shall unblock at least one of the threads that are blocked on  
32354 the specified condition variable *cond* (if any threads are blocked on *cond*).32355 If more than one thread is blocked on a condition variable, the scheduling policy determines the  
32356 order in which threads are unblocked. When each thread unblocked as a result of a  
32357 *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()* returns from its call to *pthread\_cond\_wait()* or  
32358 *pthread\_cond\_timedwait()*, the thread owns the mutex with which it called *pthread\_cond\_wait()* or  
32359 *pthread\_cond\_timedwait()*. The thread(s) that are unblocked shall contend for the mutex  
32360 according to the scheduling policy (if applicable), and as if each had called *pthread\_mutex\_lock()*.32361 The *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()* functions may be called by a thread  
32362 whether or not it currently owns the mutex that threads calling *pthread\_cond\_wait()* or  
32363 *pthread\_cond\_timedwait()* have associated with the condition variable during their waits;  
32364 however, if predictable scheduling behavior is required, then that mutex shall be locked by the  
32365 thread calling *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()*.32366 The *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* functions have no effect if there are no  
32367 threads currently blocked on *cond*.32368 **RETURN VALUE**32369 If successful, the *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* functions shall return zero;  
32370 otherwise, an error number shall be returned to indicate the error.32371 **ERRORS**32372 The *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* function may fail if:32373 [EINVAL] The value *cond* does not refer to an initialized condition variable.

32374 These functions shall not return an error code of [EINTR].

32375 **EXAMPLES**

32376 None.

32377 **APPLICATION USAGE**32378 The *pthread\_cond\_broadcast()* function is used whenever the shared-variable state has been  
32379 changed in a way that more than one thread can proceed with its task. Consider a single  
32380 producer/multiple consumer problem, where the producer can insert multiple items on a list  
32381 that is accessed one item at a time by the consumers. By calling the *pthread\_cond\_broadcast()*  
32382 function, the producer would notify all consumers that might be waiting, and thereby the  
32383 application would receive more throughput on a multiprocessor. In addition,  
32384 *pthread\_cond\_broadcast()* makes it easier to implement a read-write lock. The  
32385 *pthread\_cond\_broadcast()* function is needed in order to wake up all waiting readers when a  
32386 writer releases its lock. Finally, the two-phase commit algorithm can use this broadcast function

32387 to notify all clients of an impending transaction commit.

32388 It is not safe to use the *pthread\_cond\_signal()* function in a signal handler that is invoked  
 32389 asynchronously. Even if it were safe, there would still be a race between the test of the Boolean  
 32390 *pthread\_cond\_wait()* that could not be efficiently eliminated.

32391 Mutexes and condition variables are thus not suitable for releasing a waiting thread by signaling  
 32392 from code running in a signal handler.

### 32393 RATIONALE

#### 32394 Multiple Awakenings by Condition Signal

32395 On a multiprocessor, it may be impossible for an implementation of *pthread\_cond\_signal()* to  
 32396 avoid the unblocking of more than one thread blocked on a condition variable. For example,  
 32397 consider the following partial implementation of *pthread\_cond\_wait()* and *pthread\_cond\_signal()*,  
 32398 executed by two threads in the order given. One thread is trying to wait on the condition  
 32399 variable, another is concurrently executing *pthread\_cond\_signal()*, while a third thread is already  
 32400 waiting.

```

32401 pthread_cond_wait(mutex, cond):
32402 value = cond->value; /* 1 */
32403 pthread_mutex_unlock(mutex); /* 9 */
32404 pthread_mutex_lock(cond->mutex); /* 10 */
32405 if (value == cond->value) { /* 11 */
32406 me->next_cond = cond->waiter;
32407 cond->waiter = me;
32408 pthread_mutex_unlock(cond->mutex);
32409 unable_to_run(me);
32410 } else
32411 pthread_mutex_unlock(cond->mutex); /* 12 */
32412 pthread_mutex_lock(mutex); /* 13 */

32413 pthread_cond_signal(cond):
32414 pthread_mutex_lock(cond->mutex); /* 2 */
32415 cond->value++; /* 3 */
32416 if (cond->waiter) { /* 4 */
32417 sleeper = cond->waiter; /* 5 */
32418 cond->waiter = sleeper->next_cond; /* 6 */
32419 able_to_run(sleeper); /* 7 */
32420 }
32421 pthread_mutex_unlock(cond->mutex); /* 8 */

```

32422 The effect is that more than one thread can return from its call to *pthread\_cond\_wait()* or  
 32423 *pthread\_cond\_timedwait()* as a result of one call to *pthread\_cond\_signal()*. This effect is called  
 32424 “spurious wakeup”. Note that the situation is self-correcting in that the number of threads that  
 32425 are so awakened is finite; for example, the next thread to call *pthread\_cond\_wait()* after the  
 32426 sequence of events above blocks.

32427 While this problem could be resolved, the loss of efficiency for a fringe condition that occurs  
 32428 only rarely is unacceptable, especially given that one has to check the predicate associated with a  
 32429 condition variable anyway. Correcting this problem would unnecessarily reduce the degree of  
 32430 concurrency in this basic building block for all higher-level synchronization operations.

32431 An added benefit of allowing spurious wakeups is that applications are forced to code a  
 32432 predicate-testing-loop around the condition wait. This also makes the application tolerate  
 32433 superfluous condition broadcasts or signals on the same condition variable that may be coded in

32434 some other part of the application. The resulting applications are thus more robust. Therefore,  
32435 IEEE Std. 1003.1-200x explicitly documents that spurious wakeups may occur.

32436 **FUTURE DIRECTIONS**

32437 None.

32438 **SEE ALSO**

32439 *pthread\_cond\_destroy()*, *pthread\_cond\_timedwait()*, *pthread\_cond\_wait()*, the Base Definitions  
32440 volume of IEEE Std. 1003.1-200x, <pthread.h>

32441 **CHANGE HISTORY**

32442 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32443 **Issue 6**

32444 The *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* functions are marked as part of the  
32445 Threads option.

32446 The APPLICATION USAGE section is added.

## 32447 NAME

32448 pthread\_cond\_destroy, pthread\_cond\_init — destroy and initialize condition variables

## 32449 SYNOPSIS

32450 THR #include &lt;pthread.h&gt;

```

32451 int pthread_cond_destroy(pthread_cond_t *cond);
32452 int pthread_cond_init(pthread_cond_t *restrict cond,
32453 const pthread_condattr_t *restrict attr);
32454 pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
32455

```

## 32456 DESCRIPTION

32457 The *pthread\_cond\_destroy()* function destroys the given condition variable specified by *cond*; the  
 32458 object becomes, in effect, uninitialized. An implementation may cause *pthread\_cond\_destroy()* to  
 32459 set the object referenced by *cond* to an invalid value. A destroyed condition variable object can be  
 32460 re-initialized using *pthread\_cond\_init()*; the results of otherwise referencing the object after it has  
 32461 been destroyed are undefined.

32462 It shall be safe to destroy an initialized condition variable upon which no threads are currently  
 32463 blocked. Attempting to destroy a condition variable upon which other threads are currently  
 32464 blocked results in undefined behavior.

32465 The *pthread\_cond\_init()* function initializes the condition variable referenced by *cond* with  
 32466 attributes referenced by *attr*. If *attr* is NULL, the default condition variable attributes are used;  
 32467 the effect is the same as passing the address of a default condition variable attributes object.  
 32468 Upon successful initialization, the state of the condition variable becomes initialized.

32469 Only *cond* itself may be used for performing synchronization. The result of referring to copies of  
 32470 *cond* in calls to *pthread\_cond\_wait()*, *pthread\_cond\_timedwait()*, *pthread\_cond\_signal()*,  
 32471 *pthread\_cond\_broadcast()*, and *pthread\_cond\_destroy()* is undefined.

32472 Attempting to initialize an already initialized condition variable results in undefined behavior.

32473 In cases where default condition variable attributes are appropriate, the macro  
 32474 PTHREAD\_COND\_INITIALIZER can be used to initialize condition variables that are statically  
 32475 allocated. The effect shall be equivalent to dynamic initialization by a call to *pthread\_cond\_init()*  
 32476 with parameter *attr* specified as NULL, except that no error checks are performed.

## 32477 RETURN VALUE

32478 If successful, the *pthread\_cond\_destroy()* and *pthread\_cond\_init()* functions shall return zero;  
 32479 otherwise, an error number shall be returned to indicate the error.

32480 The [EBUSY] and [EINVAL] error checks, if implemented, shall act as if they were performed  
 32481 immediately at the beginning of processing for the function and caused an error return prior to  
 32482 modifying the state of the condition variable specified by *cond*.

## 32483 ERRORS

32484 The *pthread\_cond\_destroy()* function may fail if:

32485 [EBUSY] The implementation has detected an attempt to destroy the object referenced  
 32486 by *cond* while it is referenced (for example, while being used in a  
 32487 *pthread\_cond\_wait()* or *pthread\_cond\_timedwait()*) by another thread.

32488 [EINVAL] The value specified by *cond* is invalid.

32489 The *pthread\_cond\_init()* function shall fail if:

32490 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
 32491 another condition variable.

32492 [ENOMEM] Insufficient memory exists to initialize the condition variable. |

32493 The `pthread_cond_init()` function may fail if:

32494 [EBUSY] The implementation has detected an attempt to re-initialize the object |

32495 referenced by `cond`, a previously initialized, but not yet destroyed, condition |

32496 variable.

32497 [EINVAL] The value specified by `attr` is invalid. |

32498 These functions shall not return an error code of [EINTR]. |

**32499 EXAMPLES**

32500 A condition variable can be destroyed immediately after all the threads that are blocked on it are  
32501 awakened. For example, consider the following code:

```
32502 struct list {
32503 pthread_mutex_t lm;
32504 ...
32505 }
32506 struct elt {
32507 key k;
32508 int busy;
32509 pthread_cond_t notbusy;
32510 ...
32511 }
32512 /* Find a list element and reserve it. */
32513 struct elt *
32514 list_find(struct list *lp, key k)
32515 {
32516 struct elt *ep;
32517 pthread_mutex_lock(&lp->lm);
32518 while ((ep = find_elt(l, k) != NULL) && ep->busy)
32519 pthread_cond_wait(&ep->notbusy, &lp->lm);
32520 if (ep != NULL)
32521 ep->busy = 1;
32522 pthread_mutex_unlock(&lp->lm);
32523 return(ep);
32524 }
32525 delete_elt(struct list *lp, struct elt *ep)
32526 {
32527 pthread_mutex_lock(&lp->lm);
32528 assert(ep->busy);
32529 ... remove ep from list ...
32530 ep->busy = 0; /* Paranoid. */
32531 (A) pthread_cond_broadcast(&ep->notbusy);
32532 pthread_mutex_unlock(&lp->lm);
32533 (B) pthread_cond_destroy(&rp->notbusy);
32534 free(ep);
32535 }
```

32536 In this example, the condition variable and its list element may be freed (line B) immediately  
32537 after all threads waiting for it are awakened (line A), since the mutex and the code ensure that no  
32538 other thread can touch the element to be deleted.

## 32539 APPLICATION USAGE

32540 None.

## 32541 RATIONALE

32542 See *pthread\_mutex\_init()*; a similar rationale applies to condition variables.

## 32543 FUTURE DIRECTIONS

32544 None.

## 32545 SEE ALSO

32546 *pthread\_cond\_broadcast()*, *pthread\_cond\_signal()*, *pthread\_cond\_timedwait()*, *pthread\_cond\_wait()*,  
32547 the Base Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

## 32548 CHANGE HISTORY

32549 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

## 32550 Issue 6

32551 The *pthread\_cond\_destroy()* and *pthread\_cond\_init()* functions are marked as part of the Threads  
32552 option.

32553 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

32554 The **restrict** keyword is added to the *pthread\_cond\_init()* prototype for alignment with the  
32555 ISO/IEC 9899:1999 standard.



32556 **NAME**

32557 pthread\_cond\_init — initialize condition variables

32558 **SYNOPSIS**

32559 THR #include &lt;pthread.h&gt;

32560 int pthread\_cond\_init(pthread\_cond\_t \*restrict cond,

32561 const pthread\_condattr\_t \*restrict attr);

32562 pthread\_cond\_t cond = PTHREAD\_COND\_INITIALIZER;

32563

32564 **DESCRIPTION**32565 Refer to *pthread\_cond\_destroy()*.

32566 **NAME**

32567 pthread\_cond\_signal — signal a condition

32568 **SYNOPSIS**

32569 THR #include <pthread.h>

32570 int pthread\_cond\_signal(pthread\_cond\_t \*cond);

32571

32572 **DESCRIPTION**

32573 Refer to *pthread\_cond\_broadcast()*.

32574 **NAME**

32575 pthread\_cond\_timedwait, pthread\_cond\_wait — wait on a condition

32576 **SYNOPSIS**

32577 THR #include &lt;pthread.h&gt;

```

32578 int pthread_cond_timedwait(pthread_cond_t *restrict cond,
32579 pthread_mutex_t *restrict mutex,
32580 const struct timespec *restrict abstime);
32581 int pthread_cond_wait(pthread_cond_t *restrict cond,
32582 pthread_mutex_t *restrict mutex);
32583

```

32584 **DESCRIPTION**

32585 The *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()* functions are used to block on a condition  
 32586 variable. They shall be called with *mutex* locked by the calling thread or undefined behavior  
 32587 results.

32588 These functions atomically release *mutex* and cause the calling thread to block on the condition  
 32589 variable *cond*; atomically here means “atomically with respect to access by another thread to the  
 32590 mutex and then the condition variable”. That is, if another thread is able to acquire the mutex  
 32591 after the about-to-block thread has released it, then a subsequent call to *pthread\_cond\_broadcast()*  
 32592 or *pthread\_cond\_signal()* in that thread shall behave as if it were issued after the about-to-block  
 32593 thread has blocked.

32594 Upon successful return, the mutex has been locked and is owned by the calling thread.

32595 When using condition variables there is always a Boolean predicate involving shared variables  
 32596 associated with each condition wait that is true if the thread should proceed. Spurious wakeups  
 32597 from the *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()* functions may occur. Since the return  
 32598 from *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()* does not imply anything about the value  
 32599 of this predicate, the predicate should be re-evaluated upon such return.

32600 The effect of using more than one mutex for concurrent *pthread\_cond\_timedwait()* or  
 32601 *pthread\_cond\_wait()* operations on the same condition variable is undefined; that is, a condition  
 32602 variable becomes bound to a unique mutex when a thread waits on the condition variable, and  
 32603 this (dynamic) binding ends when the wait returns.

32604 A condition wait (whether timed or not) is a cancelation point. When the cancelability enable  
 32605 state of a thread is set to `PTHREAD_CANCEL_DEFERRED`, a side effect of acting upon a  
 32606 cancelation request while in a condition wait is that the mutex is (in effect) re-acquired before  
 32607 calling the first cancelation cleanup handler. The effect is as if the thread were unblocked,  
 32608 allowed to execute up to the point of returning from the call to *pthread\_cond\_timedwait()* or  
 32609 *pthread\_cond\_wait()*, but at that point notices the cancelation request and instead of returning to  
 32610 the caller of *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()*, starts the thread cancelation  
 32611 activities, which includes calling cancelation cleanup handlers.

32612 A thread that has been unblocked because it has been canceled while blocked in a call to  
 32613 *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()* shall not consume any condition signal that  
 32614 may be directed concurrently at the condition variable if there are other threads blocked on the  
 32615 condition variable.

32616 The *pthread\_cond\_timedwait()* function is the same as *pthread\_cond\_wait()* except that an error is  
 32617 returned if the absolute time specified by *abstime* passes (that is, system time equals or exceeds  
 32618 *abstime*) before the condition *cond* is signaled or broadcasted, or if the absolute time specified by  
 32619 *abstime* has already been passed at the time of the call. If the Clock Selection option is supported,  
 32620 the condition variable shall have a clock attribute which specifies the clock that shall be used to

32621 measure the time specified by the *abstime* argument. When such timeouts occur,  
32622 *pthread\_cond\_timedwait()* shall nonetheless release and re-acquire the mutex referenced by *mutex*.  
32623 The *pthread\_cond\_timedwait()* function is also a cancellation point.

32624 If a signal is delivered to a thread waiting for a condition variable, upon return from the signal  
32625 handler the thread resumes waiting for the condition variable as if it was not interrupted, or it  
32626 shall return zero due to spurious wakeup.

#### 32627 RETURN VALUE

32628 Except in the case of [ETIMEDOUT], all these error checks shall act as if they were performed  
32629 immediately at the beginning of processing for the function and shall cause an error return, in  
32630 effect, prior to modifying the state of the mutex specified by *mutex* or the condition variable  
32631 specified by *cond*.

32632 Upon successful completion, a value of zero shall be returned; otherwise, an error number shall  
32633 be returned to indicate the error.

#### 32634 ERRORS

32635 The *pthread\_cond\_timedwait()* function shall fail if:

32636 [ETIMEDOUT] The time specified by *abstime* to *pthread\_cond\_timedwait()* has passed.

32637 The *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()* functions may fail if:

32638 [EINVAL] The value specified by *cond*, *mutex*, or *abstime* is invalid.

32639 [EINVAL] Different mutexes were supplied for concurrent *pthread\_cond\_timedwait()* or  
32640 *pthread\_cond\_wait()* operations on the same condition variable.

32641 [EPERM] The mutex was not owned by the current thread at the time of the call.

32642 These functions shall not return an error code of [EINTR].

#### 32643 EXAMPLES

32644 None.

#### 32645 APPLICATION USAGE

32646 None.

#### 32647 RATIONALE

##### 32648 Condition Wait Semantics

32649 It is important to note that when *pthread\_cond\_wait()* and *pthread\_cond\_timedwait()* return  
32650 without error, the associated predicate may still be false. Similarly, when  
32651 *pthread\_cond\_timedwait()* returns with the timeout error, the associated predicate may be true  
32652 due to an unavoidable race between the expiration of the timeout and the predicate state change.

32653 Some implementations, particularly on a multiprocessor, may sometimes cause multiple threads  
32654 to wake up when the condition variable is signaled simultaneously on different processors.

32655 In general, whenever a condition wait returns, the thread has to re-evaluate the predicate  
32656 associated with the condition wait to determine whether it can safely proceed, should wait  
32657 again, or should declare a timeout. A return from the wait does not imply that the associated  
32658 predicate is either true or false.

32659 It is thus recommended that a condition wait be enclosed in the equivalent of a “while loop”  
32660 that checks the predicate.

32661 **Timed Wait Semantics**

32662 An absolute time measure was chosen for specifying the timeout parameter for two reasons.  
 32663 First, a relative time measure can be easily implemented on top of a function that specifies  
 32664 absolute time, but there is a race condition associated with specifying an absolute timeout on top  
 32665 of a function that specifies relative timeouts. For example, assume that `clock_gettime()` returns  
 32666 the current time and `cond_relative_timed_wait()` uses relative timeouts:

```
32667 clock_gettime(CLOCK_REALTIME, &now)
32668 reltime = sleep_til_this_absolute_time -now;
32669 cond_relative_timed_wait(c, m, &reltime);
```

32670 If the thread is preempted between the first statement and the last statement, the thread blocks  
 32671 for too long. Blocking, however, is irrelevant if an absolute timeout is used. An absolute timeout  
 32672 also need not be recomputed if it is used multiple times in a loop, such as that enclosing a  
 32673 condition wait.

32674 For cases when the system clock is advanced discontinuously by an operator, it is expected that  
 32675 implementations process any timed wait expiring at an intervening time as if that time had  
 32676 actually occurred.

32677 **Cancelation and Condition Wait**

32678 A condition wait, whether timed or not, is a cancelation point. That is, the functions  
 32679 `pthread_cond_wait()` or `pthread_cond_timedwait()` are points where a pending (or concurrent)  
 32680 cancelation request is noticed. The reason for this is that an indefinite wait is possible at these  
 32681 points—whatever event is being waited for, even if the program is totally correct, might never  
 32682 occur; for example, some input data being awaited might never be sent. By making condition  
 32683 wait a cancelation point, the thread can be canceled and perform its cancelation cleanup handler  
 32684 even though it may be stuck in some indefinite wait.

32685 A side effect of acting on a cancelation request while a thread is blocked on a condition variable  
 32686 is to re-acquire the mutex before calling any of the cancelation cleanup handlers. This is done in  
 32687 order to ensure that the cancelation cleanup handler is executed in the same state as the critical  
 32688 code that lies both before and after the call to the condition wait function. This rule is also  
 32689 required when interfacing to POSIX threads from languages, such as Ada or C++, which may  
 32690 choose to map cancelation onto a language exception; this rule ensures that each exception  
 32691 handler guarding a critical section can always safely depend upon the fact that the associated  
 32692 mutex has already been locked regardless of exactly where within the critical section the  
 32693 exception was raised. Without this rule, there would not be a uniform rule that exception  
 32694 handlers could follow regarding the lock, and so coding would become very cumbersome.

32695 Therefore, since *some* statement has to be made regarding the state of the lock when a  
 32696 cancelation is delivered during a wait, a definition has been chosen that makes application  
 32697 coding most convenient and error free.

32698 When acting on a cancelation request while a thread is blocked on a condition variable, the  
 32699 implementation is required to ensure that the thread does not consume any condition signals  
 32700 directed at that condition variable if there are any other threads waiting on that condition  
 32701 variable. This rule is specified in order to avoid deadlock conditions that could occur if these two  
 32702 independent requests (one acting on a thread and the other acting on the condition variable)  
 32703 were not processed independently.

32704 **Performance of Mutexes and Condition Variables**

32705 Mutexes are expected to be locked only for a few instructions. This practice is almost  
 32706 automatically enforced by the desire of programmers to avoid long serial regions of execution  
 32707 (which would reduce total effective parallelism).

32708 When using mutexes and condition variables, one tries to ensure that the usual case is to lock the  
 32709 mutex, access shared data, and unlock the mutex. Waiting on a condition variable should be a  
 32710 relatively rare situation. For example, when implementing a read-write lock, code that acquires a  
 32711 read-lock typically needs only to increment the count of readers (under mutual-exclusion) and  
 32712 return. The calling thread would actually wait on the condition variable only when there is  
 32713 already an active writer. So the efficiency of a synchronization operation is bounded by the cost  
 32714 of mutex lock/unlock and not by condition wait. Note that in the usual case there is no context  
 32715 switch.

32716 This is not to say that the efficiency of condition waiting is unimportant. Since there needs to be  
 32717 at least one context switch per Ada rendezvous, the efficiency of waiting on a condition variable  
 32718 is important. The cost of waiting on a condition variable should be little more than the minimal  
 32719 cost for a context switch plus the time to unlock and lock the mutex.

32720 **Features of Mutexes and Condition Variables**

32721 It had been suggested that the mutex acquisition and release be decoupled from condition wait.  
 32722 This was rejected because it is the combined nature of the operation that, in fact, facilitates  
 32723 realtime implementations. Those implementations can atomically move a high-priority thread  
 32724 between the condition variable and the mutex in a manner that is transparent to the caller. This  
 32725 can prevent extra context switches and provide more deterministic acquisition of a mutex when  
 32726 the waiting thread is signaled. Thus, fairness and priority issues can be dealt with directly by the  
 32727 scheduling discipline. Furthermore, the current condition wait operation matches existing  
 32728 practice.

32729 **Scheduling Behavior of Mutexes and Condition Variables**

32730 Synchronization primitives that attempt to interfere with scheduling policy by specifying an  
 32731 ordering rule are considered undesirable. Threads waiting on mutexes and condition variables  
 32732 are selected to proceed in an order dependent upon the scheduling policy rather than in some  
 32733 fixed order (for example, FIFO or priority). Thus, the scheduling policy determines which  
 32734 thread(s) are awakened and allowed to proceed.

32735 **Timed Condition Wait**

32736 The *pthread\_cond\_timedwait()* function allows an application to give up waiting for a particular  
 32737 condition after a given amount of time. An example of its use follows:

```
32738 (void) pthread_mutex_lock(&t.mn);
32739 t.waiters++;
32740 clock_gettime(CLOCK_REALTIME, &ts);
32741 ts.tv_sec += 5;
32742 rc = 0;
32743 while (! mypredicate(&t) && rc == 0)
32744 rc = pthread_cond_timedwait(&t.cond, &t.mn, &ts);
32745 t.waiters--;
32746 if (rc == 0) setmystate(&t);
32747 (void) pthread_mutex_unlock(&t.mn);
```

32748 By making the timeout parameter absolute, it does not need to be recomputed each time the  
32749 program checks its blocking predicate. If the timeout was relative, it would have to be  
32750 recomputed before each call. This would be especially difficult since such code would need to  
32751 take into account the possibility of extra wakeups that result from extra broadcasts or signals on  
32752 the condition variable that occur before either the predicate is true or the timeout is due.

32753 **FUTURE DIRECTIONS**

32754 None.

32755 **SEE ALSO**

32756 *pthread\_cond\_signal()*, *pthread\_cond\_broadcast()*, the Base Definitions volume of  
32757 IEEE Std. 1003.1-200x, <pthread.h>

32758 **CHANGE HISTORY**

32759 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32760 **Issue 6**

32761 The *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()* functions are marked as part of the  
32762 Threads option.

32763 The Open Group corrigenda item U021/9 has been applied, correcting the prototype for the  
32764 *pthread\_cond\_wait()* function.

32765 The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by adding semantics  
32766 for the Clock Selection option.

32767 The ERRORS section has an additional case for [EPERM] in response to IEEE PASC  
32768 Interpretation 1003.1c #28.

32769 The **restrict** keyword is added to the *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()*  
32770 prototypes for alignment with the ISO/IEC 9899:1999 standard.

32771 **NAME**

32772 pthread\_cond\_wait — wait on a condition

32773 **SYNOPSIS**

32774 THR #include <pthread.h>

32775 int pthread\_cond\_wait(pthread\_cond\_t \*restrict cond,  
32776 pthread\_mutex\_t \*restrict mutex);

32777

32778 **DESCRIPTION**

32779 Refer to *pthread\_cond\_timedwait()*.



32780 **NAME**

32781 pthread\_condattr\_destroy, pthread\_condattr\_init — destroy and initialize condition variable  
 32782 attributes object

32783 **SYNOPSIS**

32784 THR #include <pthread.h>

32785 int pthread\_condattr\_destroy(pthread\_condattr\_t \*attr);

32786 int pthread\_condattr\_init(pthread\_condattr\_t \*attr);

32787

32788 **DESCRIPTION**

32789 The *pthread\_condattr\_destroy()* function destroys a condition variable attributes object; the object  
 32790 becomes, in effect, uninitialized. An implementation may cause *pthread\_condattr\_destroy()* to set  
 32791 the object referenced by *attr* to an invalid value. A destroyed condition variable attributes object  
 32792 can be re-initialized using *pthread\_condattr\_init()*; the results of otherwise referencing the object  
 32793 after it has been destroyed are undefined.

32794 The *pthread\_condattr\_init()* function initializes a condition variable attributes object *attr* with the  
 32795 default value for all of the attributes defined by the implementation.

32796 Attempting to initialize an already initialized condition variable attributes object results in  
 32797 undefined behavior.

32798 After a condition variable attributes object has been used to initialize one or more condition  
 32799 variables, any function affecting the attributes object (including destruction) does not affect any  
 32800 previously initialized condition variables.

32801 Additional attributes, their default values, and the names of the associated functions to get and  
 32802 set those attribute values are implementation-defined.

32803 **RETURN VALUE**

32804 If successful, the *pthread\_condattr\_destroy()* and *pthread\_condattr\_init()* functions shall return  
 32805 zero; otherwise, an error number shall be returned to indicate the error.

32806 **ERRORS**

32807 The *pthread\_condattr\_destroy()* function may fail if:

32808 [EINVAL] The value specified by *attr* is invalid.

32809 The *pthread\_condattr\_init()* function shall fail if:

32810 [ENOMEM] Insufficient memory exists to initialize the condition variable attributes object.

32811 These functions shall not return an error code of [EINTR].

32812 **EXAMPLES**

32813 None.

32814 **APPLICATION USAGE**

32815 None.

32816 **RATIONALE**

32817 See *pthread\_attr\_init()* and *pthread\_mutex\_init()*.

32818 A *process-shared* attribute has been defined for condition variables for the same reason it has been  
 32819 defined for mutexes.

32820 **FUTURE DIRECTIONS**

32821 None.

32822 **SEE ALSO**

32823 *pthread\_cond\_destroy()*, *pthread\_condattr\_getpshared()*, *pthread\_create()*, *pthread\_mutex\_destroy()*,  
32824 the Base Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

32825 **CHANGE HISTORY**

32826 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32827 **Issue 6**

32828 The *pthread\_condattr\_destroy()* and *pthread\_condattr\_init()* functions are marked as part of the  
32829 Threads option.

32830 **NAME**

32831 pthread\_condattr\_getclock, pthread\_condattr\_setclock — get and set the clock selection  
 32832 condition variable attribute

32833 **SYNOPSIS**

32834 THR CS #include <pthread.h>

```
32835 int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
32836 clockid_t *restrict clock_id);
32837 int pthread_condattr_setclock(pthread_condattr_t *attr,
32838 clockid_t clock_id);
32839
```

32840 **DESCRIPTION**

32841 The *pthread\_condattr\_getclock()* function obtains the value of the clock attribute from the  
 32842 attributes object referenced by *attr*. The *pthread\_condattr\_setclock()* function is used to set the  
 32843 clock attribute in an initialized attributes object referenced by *attr*. If *pthread\_condattr\_setclock()*  
 32844 is called with a *clock\_id* argument that refers to a CPU-time clock, the call shall fail.

32845 The clock attribute is the clock ID of the clock that shall be used to measure the timeout service  
 32846 of *pthread\_cond\_timedwait()*. The default value of the clock attribute shall refer to the system  
 32847 clock.

32848 **RETURN VALUE**

32849 If successful, the *pthread\_condattr\_getclock()* function shall return zero and store the value of the  
 32850 clock attribute of *attr* into the object referenced by the *clock\_id* argument. Otherwise, an error  
 32851 number shall be returned to indicate the error.

32852 If successful, the *pthread\_condattr\_setclock()* function shall return zero; otherwise, an error  
 32853 number shall be returned to indicate the error.

32854 **ERRORS**

32855 These functions may fail if:

32856 [EINVAL] The value specified by *attr* is invalid.

32857 The *pthread\_condattr\_setclock()* function may fail if:

32858 [EINVAL] The value specified by *clock\_id* does not refer to a known clock, or is a CPU-  
 32859 time clock.

32860 These functions shall not return an error code of [EINTR].

32861 **EXAMPLES**

32862 None.

32863 **APPLICATION USAGE**

32864 None.

32865 **RATIONALE**

32866 None.

32867 **FUTURE DIRECTIONS**

32868 None.

32869 **SEE ALSO**

32870 *pthread\_cond\_init()*, *pthread\_cond\_timedwait()*, *pthread\_condattr\_destroy()*,  
 32871 *pthread\_condattr\_getshared()* (on page 1553), *pthread\_condattr\_init()*,  
 32872 *pthread\_condattr\_setshared()* (on page 1557), *pthread\_create()*, *pthread\_mutex\_init()*, the Base  
 32873 Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

32874 **CHANGE HISTORY**

32875 First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.

32876 **NAME**

32877 pthread\_condattr\_getpshared, pthread\_condattr\_setpshared — get and set the process-shared  
 32878 condition variable attributes

32879 **SYNOPSIS**

32880 THR TSH #include <pthread.h>

```
32881 int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
32882 int *restrict pshared);
32883 int pthread_condattr_setpshared(pthread_condattr_t *attr,
32884 int pshared);
32885
```

32886 **DESCRIPTION**

32887 The *pthread\_condattr\_getpshared()* function obtains the value of the *process-shared* attribute from  
 32888 the attributes object referenced by *attr*. The *pthread\_condattr\_setpshared()* function is used to set  
 32889 the *process-shared* attribute in an initialized attributes object referenced by *attr*.

32890 The *process-shared* attribute is set to `PTHREAD_PROCESS_SHARED` to permit a condition  
 32891 variable to be operated upon by any thread that has access to the memory where the condition  
 32892 variable is allocated, even if the condition variable is allocated in memory that is shared by  
 32893 multiple processes. If the *process-shared* attribute is `PTHREAD_PROCESS_PRIVATE`, the  
 32894 condition variable is only operated upon by threads created within the same process as the  
 32895 thread that initialized the condition variable; if threads of differing processes attempt to operate  
 32896 on such a condition variable, the behavior is undefined. The default value of the attribute is  
 32897 `PTHREAD_PROCESS_PRIVATE`.

32898 Additional attributes, their default values, and the names of the associated functions to get and  
 32899 set those attribute values are implementation-defined.

32900 **RETURN VALUE**

32901 If successful, the *pthread\_condattr\_setpshared()* function shall return zero; otherwise, an error  
 32902 number shall be returned to indicate the error.

32903 If successful, the *pthread\_condattr\_getpshared()* function shall return zero and store the value of  
 32904 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,  
 32905 an error number shall be returned to indicate the error.

32906 **ERRORS**

32907 The *pthread\_condattr\_getpshared()* and *pthread\_condattr\_setpshared()* functions may fail if:

32908 [EINVAL] The value specified by *attr* is invalid.

32909 The *pthread\_condattr\_setpshared()* function may fail if:

32910 [EINVAL] The new value specified for the attribute is outside the range of legal values  
 32911 for that attribute.

32912 These functions shall not return an error code of [EINTR].

32913 **EXAMPLES**

32914 None.

32915 **APPLICATION USAGE**

32916 None.

32917 **RATIONALE**

32918 None.

32919 **FUTURE DIRECTIONS**

32920 None.

32921 **SEE ALSO**

32922 *pthread\_create()*, *pthread\_cond\_destroy()*, *pthread\_condattr\_destroy()*, *pthread\_mutex\_destroy()*, the  
32923 Base Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

32924 **CHANGE HISTORY**

32925 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32926 **Issue 6**

32927 The *pthread\_condattr\_getpshared()* and *pthread\_condattr\_setpshared()* functions are marked as part  
32928 of the Threads and Thread Process-Shared Synchronization options.

32929 The **restrict** keyword is added to the *pthread\_condattr\_getpshared()* prototype for alignment with  
32930 the ISO/IEC 9899:1999 standard.

32931 **NAME**

32932 pthread\_condattr\_init — initialize condition variable attributes object

32933 **SYNOPSIS**

32934 THR #include &lt;pthread.h&gt;

32935 int pthread\_condattr\_init(pthread\_condattr\_t \*attr);

32936

32937 **DESCRIPTION**32938 Refer to *pthread\_condattr\_destroy()*.

32939 **NAME**

32940 pthread\_condattr\_setclock — set the clock selection condition variable attribute

32941 **SYNOPSIS**

32942 THR CS #include <pthread.h>

32943 int pthread\_condattr\_setclock(pthread\_condattr\_t \*attr,  
32944 clockid\_t clock\_id);

32945

32946 **DESCRIPTION**

32947 Refer to *pthread\_condattr\_getclock()*.



32948 **NAME**

32949 pthread\_condattr\_setpshared — set the process-shared condition variable attributes

32950 **SYNOPSIS**

32951 THR TSH #include &lt;pthread.h&gt;

32952 int pthread\_condattr\_setpshared(pthread\_condattr\_t \*attr,  
32953 int pshared);

32954

32955 **DESCRIPTION**32956 Refer to *pthread\_condattr\_getpshared()*.

## 32957 NAME

32958 pthread\_create — thread creation

## 32959 SYNOPSIS

32960 THR #include &lt;pthread.h&gt;

```
32961 int pthread_create(pthread_t *restrict thread,
32962 const pthread_attr_t *restrict attr,
32963 void *(*start_routine)(void*), void *arg);
32964
```

## 32965 DESCRIPTION

32966 The *pthread\_create()* function is used to create a new thread, with attributes specified by *attr*,  
 32967 within a process. If *attr* is NULL, the default attributes are used. If the attributes specified by *attr*  
 32968 are modified later, the thread's attributes are not affected. Upon successful completion,  
 32969 *pthread\_create()* shall store the ID of the created thread in the location referenced by *thread*.

32970 The thread is created executing *start\_routine* with *arg* as its sole argument. If the *start\_routine*  
 32971 returns, the effect shall be as if there was an implicit call to *pthread\_exit()* using the return value  
 32972 of *start\_routine* as the exit status. Note that the thread in which *main()* was originally invoked  
 32973 differs from this. When it returns from *main()*, the effect shall be as if there was an implicit call  
 32974 to *exit()* using the return value of *main()* as the exit status.

32975 The signal state of the new thread shall be initialized as follows:

- 32976 • The signal mask shall be inherited from the creating thread.
- 32977 • The set of signals pending for the new thread shall be empty.

32978 If *pthread\_create()* fails, no new thread is created and the contents of the location referenced by  
 32979 *thread* are undefined.

32980 TCT If `_POSIX_THREAD_CPUTIME` is defined, the new thread shall have a CPU-time clock  
 32981 accessible, and the initial value of this clock shall be set to zero.

## 32982 RETURN VALUE

32983 If successful, the *pthread\_create()* function shall return zero; otherwise, an error number shall be  
 32984 returned to indicate the error.

## 32985 ERRORS

32986 The *pthread\_create()* function shall fail if:

32987 [EAGAIN] The system lacked the necessary resources to create another thread, or the  
 32988 system-imposed limit on the total number of threads in a process  
 32989 PTHREAD\_THREADS\_MAX would be exceeded.

32990 [EINVAL] The value specified by *attr* is invalid.

32991 [EPERM] The caller does not have appropriate permission to set the required  
 32992 scheduling parameters or scheduling policy.

32993 The *pthread\_create()* function shall not return an error code of [EINTR].

32994 **EXAMPLES**

32995 None.

32996 **APPLICATION USAGE**

32997 None.

32998 **RATIONALE**

32999 A suggested alternative to *pthread\_create()* would be to define two separate operations: create  
 33000 and start. Some applications would find such behavior more natural. Ada, in particular,  
 33001 separates the “creation” of a task from its “activation”.

33002 Splitting the operation was rejected by the standard developers for many reasons:

- 33003 • The number of calls required to start a thread would increase from one to two and thus place  
 33004 an additional burden on applications that do not require the additional synchronization. The  
 33005 second call, however, could be avoided by the additional complication of a start-up state  
 33006 attribute.
- 33007 • An extra state would be introduced: “created but not started”. This would require the  
 33008 standard to specify the behavior of the thread operations when the target has not yet started  
 33009 executing.
- 33010 • For those applications that require such behavior, it is possible to simulate the two separate  
 33011 steps with the facilities that are currently provided. The *start\_routine()* can synchronize by  
 33012 waiting on a condition variable that is signaled by the start operation.

33013 An Ada implementor can choose to create the thread at either of two points in the Ada program:  
 33014 when the task object is created, or when the task is activated (generally at a “begin”). If the first  
 33015 approach is adopted, the *start\_routine()* needs to wait on a condition variable to receive the  
 33016 order to begin “activation”. The second approach requires no such condition variable or extra  
 33017 synchronization. In either approach, a separate Ada task control block would need to be created  
 33018 when the task object is created to hold rendezvous queues, and so on.

33019 An extension of the preceding model would be to allow the state of the thread to be modified  
 33020 between the create and start. This would allow the thread attributes object to be eliminated. This  
 33021 has been rejected because:

- 33022 • All state in the thread attributes object has to be able to be set for the thread. This would  
 33023 require the definition of functions to modify thread attributes. There would be no reduction  
 33024 in the number of function calls required to set up the thread. In fact, for an application that  
 33025 creates all threads using identical attributes, the number of function calls required to set up  
 33026 the threads would be dramatically increased. Use of a thread attributes object permits the  
 33027 application to make one set of attribute setting function calls. Otherwise, the set of attribute  
 33028 setting function calls needs to be made for each thread creation.
- 33029 • Depending on the implementation architecture, functions to set thread state would require  
 33030 kernel calls, or for other implementation reasons would not be able to be implemented as  
 33031 macros, thereby increasing the cost of thread creation.
- 33032 • The ability for applications to segregate threads by class would be lost.

33033 Another suggested alternative uses a model similar to that for process creation, such as “thread  
 33034 fork”. The fork semantics would provide more flexibility and the “create” function can be  
 33035 implemented simply by doing a thread fork followed immediately by a call to the desired “start  
 33036 routine” for the thread. This alternative has these problems:

- 33037 • For many implementations, the entire stack of the calling thread would need to be  
 33038 duplicated, since in many architectures there is no way to determine the size of the calling  
 33039 frame.

33040           • Efficiency is reduced since at least some part of the stack has to be copied, even though in  
33041           most cases the thread never needs the copied context, since it merely calls the desired start  
33042           routine.

33043 **FUTURE DIRECTIONS**

33044           None.

33045 **SEE ALSO**

33046           *fork()*, *pthread\_exit()*, *pthread\_join()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
33047           <pthread.h>

33048 **CHANGE HISTORY**

33049           First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33050 **Issue 6**

33051           The *pthread\_create()* function is marked as part of the Threads option.

33052           The following new requirements on POSIX implementations derive from alignment with the  
33053           Single UNIX Specification:

33054           • The [EPERM] mandatory error condition is added.

33055           The thread CPU-time clock semantics are added for alignment with IEEE Std. 1003.1d-1999.

33056           The **restrict** keyword is added to the *pthread\_create()* prototype for alignment with the  
33057           ISO/IEC 9899:1999 standard.

33058 **NAME**

33059 pthread\_detach — detach a thread

33060 **SYNOPSIS**

33061 THR #include &lt;pthread.h&gt;

33062 int pthread\_detach(pthread\_t thread);

33063

33064 **DESCRIPTION**

33065 The *pthread\_detach()* function is used to indicate to the implementation that storage for the  
 33066 thread *thread* can be reclaimed when that thread terminates. If *thread* has not terminated,  
 33067 *pthread\_detach()* shall not cause it to terminate. The effect of multiple *pthread\_detach()* calls on  
 33068 the same target thread is unspecified.

33069 **RETURN VALUE**

33070 If the call succeeds, *pthread\_detach()* shall return 0; otherwise, an error number shall be returned  
 33071 to indicate the error.

33072 **ERRORS**33073 The *pthread\_detach()* function shall fail if:

33074 [EINVAL] The implementation has detected that the value specified by *thread* does not  
 33075 refer to a joinable thread.

33076 [ESRCH] No thread could be found corresponding to that specified by the given thread  
 33077 ID.

33078 The *pthread\_detach()* function shall not return an error code of [EINTR].33079 **EXAMPLES**

33080 None.

33081 **APPLICATION USAGE**

33082 None.

33083 **RATIONALE**

33084 The *pthread\_join()* or *pthread\_detach()* functions should eventually be called for every thread that  
 33085 is created so that storage associated with the thread may be reclaimed.

33086 It has been suggested that a “detach” function is not necessary; the *detachstate* thread creation  
 33087 attribute is sufficient, since a thread need never be dynamically detached. However, need arises  
 33088 in at least two cases:

33089 1. In a cancellation handler for a *pthread\_join()* it is nearly essential to have a *pthread\_detach()*  
 33090 function in order to detach the thread on which *pthread\_join()* was waiting. Without it, it  
 33091 would be necessary to have the handler do another *pthread\_join()* to attempt to detach the  
 33092 thread, which would both delay the cancellation processing for an unbounded period and  
 33093 introduce a new call to *pthread\_join()*, which might itself need a cancellation handler. A  
 33094 dynamic detach is nearly essential in this case.

33095 2. In order to detach the “initial thread” (as may be desirable in processes that set up server  
 33096 threads).

33097 **FUTURE DIRECTIONS**

33098 None.

33099 **SEE ALSO**

33100 *pthread\_join()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

33101 **CHANGE HISTORY**

33102 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33103 **Issue 6**

33104 The *pthread\_detach()* function is marked as part of the Threads option.

33105 **NAME**

33106 pthread\_equal — compare thread IDs

33107 **SYNOPSIS**

33108 THR #include &lt;pthread.h&gt;

33109 int pthread\_equal(pthread\_t t1, pthread\_t t2);

33110

33111 **DESCRIPTION**33112 This function compares the thread IDs *t1* and *t2*.33113 **RETURN VALUE**33114 The *pthread\_equal()* function shall return a non-zero value if *t1* and *t2* are equal; otherwise, zero  
33115 shall be returned.33116 If either *t1* or *t2* are not valid thread IDs, the behavior is undefined.33117 **ERRORS**

33118 No errors are defined.

33119 The *pthread\_equal()* function shall not return an error code of [EINTR].33120 **EXAMPLES**

33121 None.

33122 **APPLICATION USAGE**

33123 None.

33124 **RATIONALE**33125 Implementations may choose to define a thread ID as a structure. This allows additional  
33126 flexibility and robustness over using an **int**. For example, a thread ID could include a sequence  
33127 number that allows detection of “dangling IDs” (copies of a thread ID that has been detached).  
33128 Because the C language does not support comparison on structure types, the *pthread\_equal()*  
33129 function is provided to compare thread IDs.33130 **FUTURE DIRECTIONS**

33131 None.

33132 **SEE ALSO**33133 *pthread\_create()*, *pthread\_self()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
33134 <pthread.h>33135 **CHANGE HISTORY**

33136 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33137 **Issue 6**33138 The *pthread\_equal()* function is marked as part of the Threads option.

33139 **NAME**

33140 pthread\_exit — thread termination

33141 **SYNOPSIS**

33142 THR #include &lt;pthread.h&gt;

33143 void pthread\_exit(void \*value\_ptr);

33144

33145 **DESCRIPTION**

33146 The *pthread\_exit()* function shall terminate the calling thread and make the value *value\_ptr*  
33147 available to any successful join with the terminating thread. Any cancellation cleanup handlers  
33148 that have been pushed and not yet popped shall be popped in the reverse order that they were  
33149 pushed and then executed. After all cancellation cleanup handlers have been executed, if the  
33150 thread has any thread-specific data, appropriate destructor functions shall be called in an  
33151 unspecified order. Thread termination does not release any application visible process resources,  
33152 including, but not limited to, mutexes and file descriptors, nor does it perform any process-level  
33153 cleanup actions, including, but not limited to, calling any *atexit()* routines that may exist.

33154 An implicit call to *pthread\_exit()* is made when a thread other than the thread in which *main()*  
33155 was first invoked returns from the start routine that was used to create it. The function's return  
33156 value serves as the thread's exit status.

33157 The behavior of *pthread\_exit()* is undefined if called from a cancellation cleanup handler or  
33158 destructor function that was invoked as a result of either an implicit or explicit call to  
33159 *pthread\_exit()*.

33160 After a thread has terminated, the result of access to local (auto) variables of the thread is  
33161 undefined. Thus, references to local variables of the exiting thread should not be used for the  
33162 *pthread\_exit()* *value\_ptr* parameter value.

33163 The process shall exit with an exit status of 0 after the last thread has been terminated. The  
33164 behavior shall be as if the implementation called *exit()* with a zero argument at thread  
33165 termination time.

33166 **RETURN VALUE**33167 The *pthread\_exit()* function cannot return to its caller.33168 **ERRORS**

33169 No errors are defined.

33170 The *pthread\_exit()* function shall not return an error code of [EINTR].33171 **EXAMPLES**

33172 None.

33173 **APPLICATION USAGE**

33174 None.

33175 **RATIONALE**

33176 The normal mechanism by which a thread terminates is to return from the routine that was  
33177 specified in the *pthread\_create()* call that started it. The *pthread\_exit()* function provides the  
33178 capability for a thread to terminate without requiring a return from the start routine of that  
33179 thread, thereby providing a function analogous to *exit()*.

33180 Regardless of the method of thread termination, any cancellation cleanup handlers that have  
33181 been pushed and not yet popped are executed, and the destructors for any existing thread-  
33182 specific data are executed. This volume of IEEE Std. 1003.1-200x requires that cancellation  
33183 cleanup handlers be popped and called in order. After all cancellation cleanup handlers have



33184 been executed, thread-specific data destructors are called, in an unspecified order, for each item  
33185 of thread-specific data that exists in the thread. This ordering is necessary because cancellation  
33186 cleanup handlers may rely on thread-specific data.

33187 As the meaning of the status is determined by the application (except when the thread has been  
33188 canceled, in which case it is PTHREAD\_CANCELED), the implementation has no idea what an  
33189 illegal status value is, which is why no address error checking is done.

33190 **FUTURE DIRECTIONS**

33191 None.

33192 **SEE ALSO**

33193 *exit()*, *pthread\_create()*, *pthread\_join()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |  
33194 <pthread.h>

33195 **CHANGE HISTORY**

33196 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33197 **Issue 6**

33198 The *pthread\_exit()* function is marked as part of the Threads option. |

## 33199 NAME

33200 pthread\_getconcurrency, pthread\_setconcurrency — get or set level of concurrency

## 33201 SYNOPSIS

```
33202 xSI #include <pthread.h>
```

```
33203 int pthread_getconcurrency(void);
```

```
33204 int pthread_setconcurrency(int new_level);
```

33205

## 33206 DESCRIPTION

33207 Unbound threads in a process may or may not be required to be simultaneously active. By  
33208 default, the threads implementation ensures that a sufficient number of threads are active so that  
33209 the process can continue to make progress. While this conserves system resources, it may not  
33210 produce the most effective level of concurrency.

33211 The *pthread\_setconcurrency()* function allows an application to inform the threads  
33212 implementation of its desired concurrency level, *new\_level*. The actual level of concurrency  
33213 provided by the implementation as a result of this function call is unspecified.

33214 If *new\_level* is zero, it causes the implementation to maintain the concurrency level at its  
33215 discretion as if *pthread\_setconcurrency()* had never been called.

33216 The *pthread\_getconcurrency()* function shall return the value set by a previous call to the  
33217 *pthread\_setconcurrency()* function. If the *pthread\_setconcurrency()* function was not previously  
33218 called, this function shall return zero to indicate that the implementation is maintaining the  
33219 concurrency level.

33220 When an application calls *pthread\_setconcurrency()* it is informing the implementation of its  
33221 desired concurrency level. The implementation uses this as a hint, not a requirement.

33222 If an implementation does not support multiplexing of user threads on top of several kernel-  
33223 scheduled entities, the *pthread\_setconcurrency()* and *pthread\_getconcurrency()* functions are  
33224 provided for source code compatibility but they have no effect when called. To maintain the  
33225 function semantics, the *new\_level* parameter is saved when *pthread\_setconcurrency()* is called so  
33226 that a subsequent call to *pthread\_getconcurrency()* returns the same value.

## 33227 RETURN VALUE

33228 If successful, the *pthread\_setconcurrency()* function shall return zero; otherwise, an error number  
33229 shall be returned to indicate the error.

33230 The *pthread\_getconcurrency()* function shall always return the concurrency level set by a previous  
33231 call to *pthread\_setconcurrency()*. If the *pthread\_setconcurrency()* function has never been called,  
33232 *pthread\_getconcurrency()* shall return zero.

## 33233 ERRORS

33234 The *pthread\_setconcurrency()* function shall fail if:

33235 [EINVAL] The value specified by *new\_level* is negative.

33236 [EAGAIN] The value specific by *new\_level* would cause a system resource to be exceeded.

33237 These functions shall not return an error code of [EINTR].

33238 **EXAMPLES**

33239 None.

33240 **APPLICATION USAGE**

33241 Use of these functions changes the state of the underlying concurrency upon which the  
33242 application depends. Library developers are advised to not use the *pthread\_getconcurrency()* and  
33243 *pthread\_setconcurrency()* functions since their use may conflict with an applications use of these  
33244 functions.

33245 **RATIONALE**

33246 None.

33247 **FUTURE DIRECTIONS**

33248 None.

33249 **SEE ALSO**

33250 The Base Definitions volume of IEEE Std. 1003.1-200x, &lt;pthread.h&gt;

33251 **CHANGE HISTORY**

33252 First released in Issue 5.

33253 **NAME**

33254 pthread\_getcpuclockid — access a thread CPU-time clock (**REALTIME**)

33255 **SYNOPSIS**

```
33256 TCT #include <pthread.h>
```

```
33257 #include <time.h>
```

```
33258 int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);
```

```
33259
```

33260 **DESCRIPTION**

33261 The *pthread\_getcpuclockid()* function shall return in *clock\_id* the clock ID of the CPU-time clock of  
33262 the thread specified by *thread\_id*, if the thread specified by *thread\_id* exists.

33263 **RETURN VALUE**

33264 Upon successful completion, *pthread\_getcpuclockid()* shall return zero; otherwise, an error  
33265 number shall be returned to indicate the error.

33266 **ERRORS**

33267 The *pthread\_getcpuclockid()* function may fail if:

33268 [ESRCH] The value specified by *thread\_id* does not refer to an existing thread.

33269 **EXAMPLES**

33270 None.

33271 **APPLICATION USAGE**

33272 The *pthread\_getcpuclockid()* function is part of the Thread CPU-Time Clocks option and need not  
33273 be provided on all implementations.

33274 **RATIONALE**

33275 None.

33276 **FUTURE DIRECTIONS**

33277 None.

33278 **SEE ALSO**

33279 *clock\_getcpuclockid()*, *clock\_getres()*, *timer\_create()*, the Base Definitions volume of  
33280 IEEE Std. 1003.1-200x, <pthread.h>, <time.h>

33281 **CHANGE HISTORY**

33282 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

33283 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

## 33284 NAME

33285 pthread\_getschedparam, pthread\_setschedparam — dynamic thread scheduling parameters  
 33286 access (**REALTIME THREADS**)

## 33287 SYNOPSIS

```
33288 TPS #include <pthread.h>
33289
33289 int pthread_getschedparam(pthread_t thread, int *restrict policy,
33290 struct sched_param *restrict param);
33291 int pthread_setschedparam(pthread_t thread, int policy,
33292 const struct sched_param *param);
33293
```

## 33294 DESCRIPTION

33295 The *pthread\_getschedparam()* and *pthread\_setschedparam()* functions allow the scheduling policy  
 33296 and scheduling parameters of individual threads within a multi-threaded process to be retrieved  
 33297 and set. For SCHED\_FIFO and SCHED\_RR, the only required member of the **sched\_param**  
 33298 structure is the priority *sched\_priority*. For SCHED\_OTHER, the affected scheduling parameters  
 33299 are implementation-defined.

33300 The *pthread\_getschedparam()* function shall retrieve the scheduling policy and scheduling  
 33301 parameters for the thread whose thread ID is given by *thread* and shall store those values in  
 33302 *policy* and *param*, respectively. The priority value returned from *pthread\_getschedparam()* shall be  
 33303 the value specified by the most recent *pthread\_setschedparam()* or *pthread\_create()* call affecting  
 33304 the target thread. It shall not reflect any temporary adjustments to its priority as a result of any  
 33305 priority inheritance or ceiling functions. The *pthread\_setschedparam()* function sets the scheduling  
 33306 policy and associated scheduling parameters for the thread whose thread ID is given by *thread* to  
 33307 the policy and associated parameters provided in *policy* and *param*, respectively.

33308 The *policy* parameter may have the value SCHED\_OTHER, SCHED\_FIFO, or SCHED\_RR. The  
 33309 scheduling parameters for the SCHED\_OTHER policy are implementation-defined. The  
 33310 SCHED\_FIFO and SCHED\_RR policies shall have a single scheduling parameter, *priority*.

33311 TSP If **\_POSIX\_THREAD\_SPORADIC\_SERVER** is defined, then the *policy* argument may have the  
 33312 value SCHED\_SPORADIC, with the exception for the *pthread\_setschedparam()* function that if the  
 33313 scheduling policy was not SCHED\_SPORADIC at the time of the call, it is implementation-  
 33314 defined whether the function is supported; in other words, the implementation need not allow  
 33315 the application to dynamically change the scheduling policy to SCHED\_SPORADIC. The  
 33316 sporadic server scheduling policy has the associated parameters *sched\_ss\_low\_priority*,  
 33317 *sched\_ss\_repl\_period*, *sched\_ss\_init\_budget*, *sched\_priority*, and *sched\_ss\_max\_repl*. The specified  
 33318 *sched\_ss\_repl\_period* shall be greater than or equal to the specified *sched\_ss\_init\_budget* for the  
 33319 function to succeed; if it is not, then the function shall fail. The value of *sched\_ss\_max\_repl* shall  
 33320 be within the inclusive range [1,{SS\_REPL\_MAX}] for the function to succeed; if not, the function  
 33321 shall fail.

33322 If the *pthread\_setschedparam()* function fails, no scheduling parameters are changed for the target  
 33323 thread.

## 33324 RETURN VALUE

33325 If successful, the *pthread\_getschedparam()* and *pthread\_setschedparam()* functions shall return zero;  
 33326 otherwise, an error number shall be returned to indicate the error.

## 33327 ERRORS

33328 The *pthread\_getschedparam()* function may fail if:

33329 [ESRCH] The value specified by *thread* does not refer to a existing thread.

- 33330 The *pthread\_setschedparam()* function may fail if:
- 33331 [EINVAL] The value specified by *policy* or one of the scheduling parameters associated  
33332 with the scheduling policy *policy* is invalid.
- 33333 [ENOTSUP] An attempt was made to set the policy or scheduling parameters to an  
33334 unsupported value.
- 33335 TSP [ENOTSUP] An attempt was made to dynamically change the scheduling policy to  
33336 SCHED\_SPORADIC, and the implementation does not support this change.
- 33337 [EPERM] The caller does not have the appropriate permission to set either the  
33338 scheduling parameters or the scheduling policy of the specified thread.
- 33339 [EPERM] The implementation does not allow the application to modify one of the  
33340 parameters to the value specified.
- 33341 [ESRCH] The value specified by *thread* does not refer to a existing thread.
- 33342 These functions shall not return an error code of [EINTR].
- 33343 **EXAMPLES**
- 33344 None.
- 33345 **APPLICATION USAGE**
- 33346 None.
- 33347 **RATIONALE**
- 33348 None.
- 33349 **FUTURE DIRECTIONS**
- 33350 None.
- 33351 **SEE ALSO**
- 33352 *sched\_getparam()*, *sched\_getscheduler()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
33353 <pthread.h>, <sched.h>
- 33354 **CHANGE HISTORY**
- 33355 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
- 33356 **Issue 6**
- 33357 The *pthread\_getschedparam()* and *pthread\_setschedparam()* functions are marked as part of the  
33358 Thread Execution Scheduling option.
- 33359 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
33360 implementation does not support the Thread Execution Scheduling option.
- 33361 The Open Group corrigenda item U026/2 has been applied correcting the prototype for the  
33362 *pthread\_setschedparam()* function so that its second argument is of type **int**.
- 33363 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std. 1003.1d-1999.
- 33364 The **restrict** keyword is added to the *pthread\_getschedparam()* prototype for alignment with the  
33365 ISO/IEC 9899:1999 standard.
- 33366 The Open Group corrigenda item U047/1 has been applied.

33367 **NAME**

33368 pthread\_getspecific, pthread\_setspecific — thread-specific data management

33369 **SYNOPSIS**

33370 THR #include &lt;pthread.h&gt;

33371 void \*pthread\_getspecific(pthread\_key\_t key);

33372 int pthread\_setspecific(pthread\_key\_t key, const void \*value);

33373

33374 **DESCRIPTION**33375 The *pthread\_getspecific()* function shall return the value currently bound to the specified *key* on  
33376 behalf of the calling thread.33377 The *pthread\_setspecific()* function shall associate a thread-specific *value* with a *key* obtained via a  
33378 previous call to *pthread\_key\_create()*. Different threads may bind different values to the same  
33379 key. These values are typically pointers to blocks of dynamically allocated memory that have  
33380 been reserved for use by the calling thread.33381 The effect of calling *pthread\_getspecific()* or *pthread\_setspecific()* with a *key* value not obtained  
33382 from *pthread\_key\_create()* or after *key* has been deleted with *pthread\_key\_delete()* is undefined.33383 Both *pthread\_getspecific()* and *pthread\_setspecific()* may be called from a thread-specific data  
33384 destructor function. A call to *pthread\_getspecific()* for the thread-specific data key being  
33385 destroyed shall return the value NULL, unless the value is changed (after the destructor starts)  
33386 by a call to *pthread\_setspecific()*. Calling *pthread\_setspecific()* thread-specific data destructor  
33387 routine may result either in lost storage (after at least PTHREAD\_DESTRUCTOR\_ITERATIONS)  
33388 or an infinite loop.

33389 Both functions may be implemented as macros.

33390 **RETURN VALUE**33391 The *pthread\_getspecific()* function shall return the thread-specific data value associated with the  
33392 given *key*. If no thread-specific data value is associated with *key*, then the value NULL shall be  
33393 returned.33394 If successful, the *pthread\_setspecific()* function shall return zero; otherwise, an error number shall  
33395 be returned to indicate the error.33396 **ERRORS**33397 No errors are returned from *pthread\_getspecific()*.33398 The *pthread\_setspecific()* function shall fail if:

33399 [ENOMEM] Insufficient memory exists to associate the value with the key.

33400 The *pthread\_setspecific()* function may fail if:

33401 [EINVAL] The key value is invalid.

33402 These functions shall not return an error code of [EINTR].

33403 **EXAMPLES**

33404           None.

33405 **APPLICATION USAGE**

33406           None.

33407 **RATIONALE**

33408           Performance and ease-of-use of *pthread\_getspecific()* is critical for functions that rely on  
33409           maintaining state in thread-specific data. Since no errors are required to be detected by it, and  
33410           since the only error that could be detected is the use of an invalid key, the function to  
33411           *pthread\_getspecific()* has been designed to favor speed and simplicity over error reporting.

33412 **FUTURE DIRECTIONS**

33413           None.

33414 **SEE ALSO**

33415           *pthread\_key\_create()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**pthread.h**>

33416 **CHANGE HISTORY**

33417           First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33418 **Issue 6**

33419           The *pthread\_getspecific()* and *pthread\_setspecific()* functions are marked as part of the Threads  
33420           option.



33421 **NAME**

33422 pthread\_join — wait for thread termination

33423 **SYNOPSIS**

33424 THR #include &lt;pthread.h&gt;

33425 int pthread\_join(pthread\_t *thread*, void \*\**value\_ptr*);

33426

33427 **DESCRIPTION**

33428 The *pthread\_join()* function shall suspend execution of the calling thread until the target *thread*  
 33429 terminates, unless the target *thread* has already terminated. On return from a successful  
 33430 *pthread\_join()* call with a non-NULL *value\_ptr* argument, the value passed to *pthread\_exit()* by  
 33431 the terminating thread shall be made available in the location referenced by *value\_ptr*. When a  
 33432 *pthread\_join()* returns successfully, the target thread has been terminated. The results of multiple  
 33433 simultaneous calls to *pthread\_join()* specifying the same target thread are undefined. If the  
 33434 thread calling *pthread\_join()* is canceled, then the target thread shall not be detached.

33435 It is unspecified whether a thread that has exited but remains unjoined counts against  
 33436 \_PTHREAD\_THREADS\_MAX.

33437 **RETURN VALUE**

33438 If successful, the *pthread\_join()* function shall return zero; otherwise, an error number shall be  
 33439 returned to indicate the error.

33440 **ERRORS**33441 The *pthread\_join()* function shall fail if:

33442 [EINVAL] The implementation has detected that the value specified by *thread* does not  
 33443 refer to a joinable thread.

33444 [ESRCH] No thread could be found corresponding to that specified by the given thread  
 33445 ID.

33446 The *pthread\_join()* function may fail if:

33447 [EDEADLK] A deadlock was detected or the value of *thread* specifies the calling thread.

33448 The *pthread\_join()* function shall not return an error code of [EINTR].33449 **EXAMPLES**

33450 An example of thread creation and deletion follows:

```

33451 typedef struct {
33452 int *ar;
33453 long n;
33454 } subarray;
33455
33456 void *
33457 incer(void *arg)
33458 {
33459 long i;
33460 for (i = 0; i < ((subarray *)arg)->n; i++)
33461 ((subarray *)arg)->ar[i]++;
33462 }
33463
33464 main()
33465 {
33466 int ar[1000000];

```

```

33465 pthread_t th1, th2;
33466 subarray sb1, sb2;

33467 sb1.ar = &ar[0];
33468 sb1.n = 500000;
33469 (void) pthread_create(&th1, NULL, incer, &sb1);

33470 sb2.ar = &ar[500000];
33471 sb2.n = 500000;
33472 (void) pthread_create(&th2, NULL, incer, &sb2);

33473 (void) pthread_join(th1, NULL);
33474 (void) pthread_join(th2, NULL);
33475 }
```

**33476 APPLICATION USAGE**

33477 None.

**33478 RATIONALE**

33479 The *pthread\_join()* function is a convenience that has proven useful in multi-threaded  
33480 applications. It is true that a programmer could simulate this function if it were not provided by  
33481 passing extra state as part of the argument to the *start\_routine()*. The terminating thread would  
33482 set a flag to indicate termination and broadcast a condition that is part of that state; a joining  
33483 thread would wait on that condition variable. While such a technique would allow a thread to  
33484 wait on more complex conditions (for example, waiting for multiple threads to terminate),  
33485 waiting on individual thread termination is considered widely useful. Also, including the  
33486 *pthread\_join()* function in no way precludes a programmer from coding such complex waits.  
33487 Thus, while not a primitive, including *pthread\_join()* in this volume of IEEE Std. 1003.1-200x was  
33488 considered valuable.

33489 The *pthread\_join()* function provides a simple mechanism allowing an application to wait for a  
33490 thread to terminate. After the thread terminates, the application may then choose to clean up  
33491 resources that were used by the thread. For instance, after *pthread\_join()* returns, any  
33492 application-provided stack storage could be reclaimed.

33493 The *pthread\_join()* or *pthread\_detach()* function should eventually be called for every thread that  
33494 is created with the *detachstate* attribute set to *PTHREAD\_CREATE\_JOINABLE* so that storage  
33495 associated with the thread may be reclaimed.

33496 The interaction between *pthread\_join()* and cancelation is well-defined for the following reasons:

- 33497 • The *pthread\_join()* function, like all other non-async-cancel-safe functions, can only be called  
33498 with deferred cancelability type.
- 33499 • Cancelation cannot occur in the disabled cancelability state.

33500 Thus, only the default cancelability state need be considered. As specified, either the  
33501 *pthread\_join()* call is canceled, or it succeeds, but not both. The difference is obvious to the  
33502 application, since either a cancelation handler is run or *pthread\_join()* returns. There are no race  
33503 conditions since *pthread\_join()* was called in the deferred cancelability state.

**33504 FUTURE DIRECTIONS**

33505 None.

**33506 SEE ALSO**

33507 *pthread\_create()*, *wait()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

33508 **CHANGE HISTORY**

33509 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33510 **Issue 6**

33511 The *pthread\_join()* function is marked as part of the Threads option.

33512 **NAME**

33513 pthread\_key\_create — thread-specific data key creation

33514 **SYNOPSIS**

33515 THR #include &lt;pthread.h&gt;

33516 int pthread\_key\_create(pthread\_key\_t \*key, void (\*destructor)(void\*));

33517

33518 **DESCRIPTION**

33519 The *pthread\_key\_create()* function shall create a thread-specific data key visible to all threads in  
 33520 the process. Key values provided by *pthread\_key\_create()* are opaque objects used to locate  
 33521 thread-specific data. Although the same key value may be used by different threads, the values  
 33522 bound to the key by *pthread\_setspecific()* are maintained on a per-thread basis and persist for the  
 33523 life of the calling thread.

33524 Upon key creation, the value NULL shall be associated with the new key in all active threads.  
 33525 Upon thread creation, the value NULL shall be associated with all defined keys in the new  
 33526 thread.

33527 An optional destructor function may be associated with each key value. At thread exit, if a key  
 33528 value has a non-NULL destructor pointer, and the thread has a non-NULL value associated with  
 33529 that key, the value of the key is set to NULL, and then the function pointed to is called with the  
 33530 previously associated value as its sole argument. The order of destructor calls is unspecified if  
 33531 more than one destructor exists for a thread when it exits.

33532 If, after all the destructors have been called for all non-NULL values with associated destructors,  
 33533 there are still some non-NULL values with associated destructors, then the process is repeated.  
 33534 If, after at least {PTHREAD\_DESTRUCTOR\_ITERATIONS} iterations of destructor calls for  
 33535 outstanding non-NULL values, there are still some non-NULL values with associated  
 33536 destructors, implementations may stop calling destructors, or they may continue calling  
 33537 destructors until no non-NULL values with associated destructors exist, even though this might  
 33538 result in an infinite loop.

33539 **RETURN VALUE**

33540 If successful, the *pthread\_key\_create()* function shall store the newly created key value at *\*key* and  
 33541 shall return zero. Otherwise, an error number shall be returned to indicate the error.

33542 **ERRORS**33543 The *pthread\_key\_create()* function shall fail if:

33544 [EAGAIN] The system lacked the necessary resources to create another thread-specific  
 33545 data key, or the system-imposed limit on the total number of keys per process  
 33546 PTHREAD\_KEYS\_MAX has been exceeded.

33547 [ENOMEM] Insufficient memory exists to create the key.

33548 The *pthread\_key\_create()* function shall not return an error code of [EINTR].

33549 **EXAMPLES**

33550 The following example demonstrates a function that initializes a thread-specific data key when  
 33551 it is first called, and associates a thread-specific object with each calling thread, initializing this  
 33552 object when necessary.

```

33553 static pthread_key_t key;
33554 static pthread_once_t key_once = PTHREAD_ONCE_INIT;

33555 static void
33556 make_key()
33557 {
33558 (void) pthread_key_create(&key, NULL);
33559 }

33560 func()
33561 {
33562 void *ptr;

33563 (void) pthread_once(&key_once, make_key);
33564 if ((ptr = pthread_getspecific(key)) == NULL) {
33565 ptr = malloc(OBJECT_SIZE);
33566 ...
33567 (void) pthread_setspecific(key, ptr);
33568 }
33569 ...
33570 }

```

33571 Note that the key has to be initialized before *pthread\_getspecific()* or *pthread\_setspecific()* can be  
 33572 used. The *pthread\_key\_create()* call could either be explicitly made in a module initialization  
 33573 routine, or it can be done implicitly by the first call to a module as in this example. Any attempt  
 33574 to use the key before it is initialized is a programming error, making the code below incorrect.

```

33575 static pthread_key_t key;

33576 func()
33577 {
33578 void *ptr;

33579 /* KEY NOT INITIALIZED!!! THIS WON'T WORK!!! */
33580 if ((ptr = pthread_getspecific(key)) == NULL &&
33581 pthread_setspecific(key, NULL) != 0) {
33582 pthread_key_create(&key, NULL);
33583 ...
33584 }
33585 }

```

33586 **APPLICATION USAGE**

33587 None.

## 33588 RATIONALE

33589 **Destructor Functions**

33590 Normally, the value bound to a key on behalf of a particular thread is a pointer to storage  
33591 allocated dynamically on behalf of the calling thread. The destructor functions specified with  
33592 *pthread\_key\_create()* are intended to be used to free this storage when the thread exits. Thread  
33593 cancelation cleanup handlers cannot be used for this purpose because thread-specific data may  
33594 persist outside the lexical scope in which the cancelation cleanup handlers operate.

33595 If the value associated with a key needs to be updated during the lifetime of the thread, it may  
33596 be necessary to release the storage associated with the old value before the new value is bound.  
33597 Although the *pthread\_setspecific()* function could do this automatically, this feature is not needed  
33598 often enough to justify the added complexity. Instead, the programmer is responsible for freeing  
33599 the stale storage:

```
33600 pthread_getspecific(key, &old);
33601 new = allocate();
33602 destructor(old);
33603 pthread_setspecific(key, new);
```

33604 **Note:** The above example could leak storage if run with asynchronous cancelation enabled.  
33605 No such problems occur in the default cancelation state if no cancelation points  
33606 occur between the get and set.

33607 There is no notion of a destructor-safe function. If an application does not call *pthread\_exit()*  
33608 from a signal handler, or if it blocks any signal whose handler may call *pthread\_exit()* while  
33609 calling async-unsafe functions, all functions may be safely called from destructors.

33610 **Non-Idempotent Data Key Creation**

33611 There were requests to make *pthread\_key\_create()* idempotent with respect to a given *key* address  
33612 parameter. This would allow applications to call *pthread\_key\_create()* multiple times for a given  
33613 *key* address and be guaranteed that only one key would be created. Doing so would require the  
33614 key value to be previously initialized (possibly at compile time) to a known null value and  
33615 would require that implicit mutual-exclusion be performed based on the address and contents of  
33616 the *key* parameter in order to guarantee that exactly one key would be created.

33617 Unfortunately, the implicit mutual-exclusion would not be limited to only *pthread\_key\_create()*.  
33618 On many implementations, implicit mutual-exclusion would also have to be performed by  
33619 *pthread\_getspecific()* and *pthread\_setspecific()* in order to guard against using incompletely stored  
33620 or not-yet-visible key values. This could significantly increase the cost of important operations,  
33621 particularly *pthread\_getspecific()*.

33622 Thus, this proposal was rejected. The *pthread\_key\_create()* function performs no implicit  
33623 synchronization. It is the responsibility of the programmer to ensure that it is called exactly once  
33624 per key before use of the key. Several straightforward mechanisms can already be used to  
33625 accomplish this, including calling explicit module initialization functions, using mutexes, and  
33626 using *pthread\_once()*. This places no significant burden on the programmer, introduces no  
33627 possibly confusing *ad hoc* implicit synchronization mechanism, and potentially allows  
33628 commonly used thread-specific data operations to be more efficient.

33629 **FUTURE DIRECTIONS**

33630 None.

33631 **SEE ALSO**

33632 *pthread\_getspecific()*, *pthread\_key\_delete()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |  
33633 **<pthread.h>**

33634 **CHANGE HISTORY**

33635 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33636 **Issue 6**

33637 The *pthread\_key\_create()* function is marked as part of the Threads option. |

33638 IEEE PASC Interpretation 1003.1c #8 is applied, updating the DESCRIPTION. |

33639 **NAME**

33640 pthread\_key\_delete — thread-specific data key deletion

33641 **SYNOPSIS**33642 THR `#include <pthread.h>`33643 `int pthread_key_delete(pthread_key_t key);`

33644

33645 **DESCRIPTION**

33646 The *pthread\_key\_delete()* function shall delete a thread-specific data key previously returned by  
33647 *pthread\_key\_create()*. The thread-specific data values associated with *key* need not be NULL at  
33648 the time *pthread\_key\_delete()* is called. It is the responsibility of the application to free any  
33649 application storage or perform any cleanup actions for data structures related to the deleted key  
33650 or associated thread-specific data in any threads; this cleanup can be done either before or after  
33651 *pthread\_key\_delete()* is called. Any attempt to use *key* following the call to *pthread\_key\_delete()*  
33652 results in undefined behavior.

33653 The *pthread\_key\_delete()* function shall be callable from within destructor functions. No  
33654 destructor functions shall be invoked by *pthread\_key\_delete()*. Any destructor function that may  
33655 have been associated with *key* shall no longer be called upon thread exit.

33656 **RETURN VALUE**

33657 If successful, the *pthread\_key\_delete()* function shall return zero; otherwise, an error number shall  
33658 be returned to indicate the error.

33659 **ERRORS**33660 The *pthread\_key\_delete()* function may fail if:33661 [EINVAL] The *key* value is invalid.33662 The *pthread\_key\_delete()* function shall not return an error code of [EINTR].33663 **EXAMPLES**

33664 None.

33665 **APPLICATION USAGE**

33666 None.

33667 **RATIONALE**

33668 A thread-specific data key deletion function has been included in order to allow the resources  
33669 associated with an unused thread-specific data key to be freed. Unused thread-specific data keys  
33670 can arise, among other scenarios, when a dynamically loaded module that allocated a key is  
33671 unloaded.

33672 Portable applications are responsible for performing any cleanup actions needed for data  
33673 structures associated with the key to be deleted, including data referenced by thread-specific  
33674 data values. No such cleanup is done by *pthread\_key\_delete()*. In particular, destructor functions  
33675 are not called. There are several reasons for this division of responsibility:

- 33676 1. The associated destructor functions used to free thread-specific data at thread exit time are  
33677 only guaranteed to work correctly when called in the thread that allocated the thread-  
33678 specific data. (Destructors themselves may utilize thread-specific data.) Thus, they cannot  
33679 be used to free thread-specific data in other threads at key deletion time. Attempting to  
33680 have them called by other threads at key deletion time would require other threads to be  
33681 asynchronously interrupted. But since interrupted threads could be in an arbitrary state,  
33682 including holding locks necessary for the destructor to run, this approach would fail. In  
33683 general, there is no safe mechanism whereby an implementation could free thread-specific  
33684 data at key deletion time.



33685           2. Even if there were a means of safely freeing thread-specific data associated with keys to be  
33686 deleted, doing so would require that implementations be able to enumerate the threads  
33687 with non-NULL data and potentially keep them from creating more thread-specific data  
33688 while the key deletion is occurring. This special case could cause extra synchronization in  
33689 the normal case, which would otherwise be unnecessary.

33690           For an application to know that it is safe to delete a key, it has to know that all the threads that  
33691 might potentially ever use the key do not attempt to use it again. For example, it could know this  
33692 if all the client threads have called a cleanup procedure declaring that they are through with the  
33693 module that is being shut down, perhaps by zero'ing a reference count.

33694 **FUTURE DIRECTIONS**

33695           None.

33696 **SEE ALSO**

33697           *pthread\_key\_create()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

33698 **CHANGE HISTORY**

33699           First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33700 **Issue 6**

33701           The *pthread\_key\_delete()* function is marked as part of the Threads option.

33702 **NAME**

33703 pthread\_kill — send a signal to a thread

33704 **SYNOPSIS**

33705 THR #include &lt;signal.h&gt;

33706 int pthread\_kill(pthread\_t thread, int sig);

33707

33708 **DESCRIPTION**33709 The *pthread\_kill()* function is used to request that a signal be delivered to the specified thread.33710 As in *kill()*, if *sig* is zero, error checking is performed but no signal is actually sent.33711 **RETURN VALUE**

33712 Upon successful completion, the function shall return a value of zero. Otherwise, the function

33713 shall return an error number. If the *pthread\_kill()* function fails, no signal shall be sent.33714 **ERRORS**33715 The *pthread\_kill()* function shall fail if:

33716 [ESRCH] No thread could be found corresponding to that specified by the given thread ID. |

33718 [EINVAL] The value of the *sig* argument is an invalid or unsupported signal number. |33719 The *pthread\_kill()* function shall not return an error code of [EINTR]. |33720 **EXAMPLES**

33721 None.

33722 **APPLICATION USAGE**33723 The *pthread\_kill()* function provides a mechanism for asynchronously directing a signal at a  
33724 thread in the calling process. This could be used, for example, by one thread to affect broadcast  
33725 delivery of a signal to a set of threads.33726 Note that *pthread\_kill()* only causes the signal to be handled in the context of the given thread;  
33727 the signal action (termination or stopping) affects the process as a whole.33728 **RATIONALE**

33729 None.

33730 **FUTURE DIRECTIONS**

33731 None.

33732 **SEE ALSO**33733 *kill()*, *pthread\_self()*, *raise()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <signal.h> |33734 **CHANGE HISTORY**

33735 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33736 **Issue 6**33737 The *pthread\_kill()* function is marked as part of the Threads option. |

33738 The APPLICATION USAGE section is added.

33739 **NAME**

33740 pthread\_mutex\_destroy, pthread\_mutex\_init — destroy and initialize a mutex

33741 **SYNOPSIS**

33742 THR #include &lt;pthread.h&gt;

```

33743 int pthread_mutex_destroy(pthread_mutex_t *mutex);
33744 int pthread_mutex_init(pthread_mutex_t *restrict mutex,
33745 const pthread_mutexattr_t *restrict attr);
33746 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
33747

```

33748 **DESCRIPTION**

33749 The *pthread\_mutex\_destroy()* function destroys the mutex object referenced by *mutex*; the mutex  
 33750 object becomes, in effect, uninitialized. An implementation may cause *pthread\_mutex\_destroy()* to  
 33751 set the object referenced by *mutex* to an invalid value. A destroyed mutex object can be re-  
 33752 initialized using *pthread\_mutex\_init()*; the results of otherwise referencing the object after it has  
 33753 been destroyed are undefined.

33754 It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked  
 33755 mutex results in undefined behavior.

33756 The *pthread\_mutex\_init()* function initializes the mutex referenced by *mutex* with attributes  
 33757 specified by *attr*. If *attr* is NULL, the default mutex attributes are used; the effect shall be the  
 33758 same as passing the address of a default mutex attributes object. Upon successful initialization,  
 33759 the state of the mutex becomes initialized and unlocked.

33760 Only *mutex* itself may be used for performing synchronization. The result of referring to copies  
 33761 of *mutex* in calls to *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, *pthread\_mutex\_unlock()*, and  
 33762 *pthread\_mutex\_destroy()* is undefined.

33763 Attempting to initialize an already initialized mutex results in undefined behavior.

33764 In cases where default mutex attributes are appropriate, the macro  
 33765 PTHREAD\_MUTEX\_INITIALIZER can be used to initialize mutexes that are statically allocated.  
 33766 The effect shall be equivalent to dynamic initialization by a call to *pthread\_mutex\_init()* with  
 33767 parameter *attr* specified as NULL, except that no error checks are performed.

33768 **RETURN VALUE**

33769 If successful, the *pthread\_mutex\_destroy()* and *pthread\_mutex\_init()* functions shall return zero;  
 33770 otherwise, an error number shall be returned to indicate the error.

33771 The [EBUSY] and [EINVAL] error checks, if implemented, act as if they were performed  
 33772 immediately at the beginning of processing for the function and shall cause an error return prior  
 33773 to modifying the state of the mutex specified by *mutex*.

33774 **ERRORS**

33775 The *pthread\_mutex\_destroy()* function may fail if:

33776 [EBUSY] The implementation has detected an attempt to destroy the object referenced  
 33777 by *mutex* while it is locked or referenced (for example, while being used in a  
 33778 *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()*) by another thread.

33779 [EINVAL] The value specified by *mutex* is invalid.

33780 The *pthread\_mutex\_init()* function shall fail if:

33781 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
 33782 another mutex.

|       |          |                                                                                      |  |
|-------|----------|--------------------------------------------------------------------------------------|--|
| 33783 | [ENOMEM] | Insufficient memory exists to initialize the mutex.                                  |  |
| 33784 | [EPERM]  | The caller does not have the privilege to perform the operation.                     |  |
| 33785 |          | The <i>pthread_mutex_init()</i> function may fail if:                                |  |
| 33786 | [EBUSY]  | The implementation has detected an attempt to re-initialize the object               |  |
| 33787 |          | referenced by <i>mutex</i> , a previously initialized, but not yet destroyed, mutex. |  |
| 33788 | [EINVAL] | The value specified by <i>attr</i> is invalid.                                       |  |
| 33789 |          | These functions shall not return an error code of [EINTR].                           |  |

**33790 EXAMPLES**

33791 None.

**33792 APPLICATION USAGE**

33793 None.

**33794 RATIONALE****33795 Alternate Implementations Possible**

33796 This volume of IEEE Std. 1003.1-200x supports several alternative implementations of mutexes.  
33797 An implementation may store the lock directly in the object of type **pthread\_mutex\_t**.  
33798 Alternatively, an implementation may store the lock in the heap and merely store a pointer,  
33799 handle, or unique ID in the mutex object. Either implementation has advantages or may be  
33800 required on certain hardware configurations. So that portable code can be written that is  
33801 invariant to this choice, this volume of IEEE Std. 1003.1-200x does not define assignment or  
33802 equality for this type, and it uses the term “initialize” to reinforce the (more restrictive) notion  
33803 that the lock may actually reside in the mutex object itself.

33804 Note that this precludes an over-specification of the type of the mutex or condition variable and  
33805 motivates the opacity of the type.

33806 An implementation is permitted, but not required, to have *pthread\_mutex\_destroy()* store an  
33807 illegal value into the mutex. This may help detect erroneous programs that try to lock (or  
33808 otherwise reference) a mutex that has already been destroyed.

**33809 Tradeoff Between Error Checks and Performance Supported**

33810 Many of the error checks were made optional in order to let implementations trade off  
33811 performance *versus* degree of error checking according to the needs of their specific applications  
33812 and execution environment. As a general rule, errors or conditions caused by the system (such as  
33813 insufficient memory) always need to be reported, but errors due to an erroneously coded  
33814 application (such as failing to provide adequate synchronization to prevent a mutex from being  
33815 deleted while in use) are made optional.

33816 A wide range of implementations is thus made possible. For example, an implementation  
33817 intended for application debugging may implement all of the error checks, but an  
33818 implementation running a single, provably correct application under very tight performance  
33819 constraints in an embedded computer might implement minimal checks. An implementation  
33820 might even be provided in two versions, similar to the options that compilers provide: a full-  
33821 checking, but slower version; and a limited-checking, but faster version. To forbid this  
33822 optionality would be a disservice to users.

33823 By carefully limiting the use of “undefined behavior” only to things that an erroneous (badly  
33824 coded) application might do, and by defining that resource-not-available errors are mandatory,  
33825 this volume of IEEE Std. 1003.1-200x ensures that a fully-conforming application is portable

33826 across the full range of implementations, while not forcing all implementations to add overhead  
33827 to check for numerous things that a correct program never does.

### 33828 **Why No Limits Defined**

33829 Defining symbols for the maximum number of mutexes and condition variables was considered  
33830 but rejected because the number of these objects may change dynamically. Furthermore, many  
33831 implementations place these objects into application memory; thus, there is no explicit  
33832 maximum.

### 33833 **Static Initializers for Mutexes and Condition Variables**

33834 Providing for static initialization of statically allocated synchronization objects allows modules  
33835 with private static synchronization variables to avoid runtime initialization tests and overhead.  
33836 Furthermore, it simplifies the coding of self-initializing modules. Such modules are common in  
33837 C libraries, where for various reasons the design calls for self-initialization instead of requiring  
33838 an explicit module initialization function to be called. An example use of static initialization  
33839 follows.

33840 Without static initialization, a self-initializing routine *foo()* might look as follows:

```
33841 static pthread_once_t foo_once = PTHREAD_ONCE_INIT;
33842 static pthread_mutex_t foo_mutex;

33843 void foo_init()
33844 {
33845 pthread_mutex_init(&foo_mutex, NULL);
33846 }

33847 void foo()
33848 {
33849 pthread_once(&foo_once, foo_init);
33850 pthread_mutex_lock(&foo_mutex);
33851 /* Do work. */
33852 pthread_mutex_unlock(&foo_mutex);
33853 }
```

33854 With static initialization, the same routine could be coded as follows:

```
33855 static pthread_mutex_t foo_mutex = PTHREAD_MUTEX_INITIALIZER;

33856 void foo()
33857 {
33858 pthread_mutex_lock(&foo_mutex);
33859 /* Do work. */
33860 pthread_mutex_unlock(&foo_mutex);
33861 }
```

33862 Note that the static initialization both eliminates the need for the initialization test inside  
33863 *pthread\_once()* and the fetch of *&foo\_mutex* to learn the address to be passed to  
33864 *pthread\_mutex\_lock()* or *pthread\_mutex\_unlock()*.

33865 Thus, the C code written to initialize static objects is simpler on all systems and is also faster on a  
33866 large class of systems; those where the (entire) synchronization object can be stored in  
33867 application memory.

33868 Yet the locking performance question is likely to be raised for machines that require mutexes to  
33869 be allocated out of special memory. Such machines actually have to have mutexes and possibly

33870 condition variables contain pointers to the actual hardware locks. For static initialization to work  
33871 on such machines, *pthread\_mutex\_lock()* also has to test whether or not the pointer to the actual  
33872 lock has been allocated. If it has not, *pthread\_mutex\_lock()* has to initialize it before use. The  
33873 reservation of such resources can be made when the program is loaded, and hence return codes  
33874 have not been added to mutex locking and condition variable waiting to indicate failure to  
33875 complete initialization.

33876 This runtime test in *pthread\_mutex\_lock()* would at first seem to be extra work; an extra test is  
33877 required to see whether the pointer has been initialized. On most machines this would actually  
33878 be implemented as a fetch of the pointer, testing the pointer against zero, and then using the  
33879 pointer if it has already been initialized. While the test might seem to add extra work, the extra  
33880 effort of testing a register is usually negligible since no extra memory references are actually  
33881 done. As more and more machines provide caches, the real expenses are memory references, not  
33882 instructions executed.

33883 Alternatively, depending on the machine architecture, there are often ways to eliminate *all*  
33884 overhead in the most important case: on the lock operations that occur *after* the lock has been  
33885 initialized. This can be done by shifting more overhead to the less frequent operation:  
33886 initialization. Since out-of-line mutex allocation also means that an address has to be  
33887 dereferenced to find the actual lock, one technique that is widely applicable is to have static  
33888 initialization store a bogus value for that address; in particular, an address that causes a machine  
33889 fault to occur. When such a fault occurs upon the first attempt to lock such a mutex, validity  
33890 checks can be done, and then the correct address for the actual lock can be filled in. Subsequent  
33891 lock operations incur no extra overhead since they do not “fault”. This is merely one technique  
33892 that can be used to support static initialization, while not adversely affecting the performance of  
33893 lock acquisition. No doubt there are other techniques that are highly machine-dependent.

33894 The locking overhead for machines doing out-of-line mutex allocation is thus similar for  
33895 modules being implicitly initialized, where it is improved for those doing mutex allocation  
33896 entirely inline. The inline case is thus made much faster, and the out-of-line case is not  
33897 significantly worse.

33898 Besides the issue of locking performance for such machines, a concern is raised that it is possible  
33899 that threads would serialize contending for initialization locks when attempting to finish  
33900 initializing statically allocated mutexes. (Such finishing would typically involve taking an  
33901 internal lock, allocating a structure, storing a pointer to the structure in the mutex, and releasing  
33902 the internal lock.) First, many implementations would reduce such serialization by hashing on  
33903 the mutex address. Second, such serialization can only occur a bounded number of times. In  
33904 particular, it can happen at most as many times as there are statically allocated synchronization  
33905 objects. Dynamically allocated objects would still be initialized via *pthread\_mutex\_init()* or  
33906 *pthread\_cond\_init()*.

33907 Finally, if none of the above optimization techniques for out-of-line allocation yields sufficient  
33908 performance for an application on some implementation, the application can avoid static  
33909 initialization altogether by explicitly initializing all synchronization objects with the  
33910 corresponding *pthread\_\*\_init()* functions, which are supported by all implementations. An  
33911 implementation can also document the tradeoffs and advise which initialization technique is  
33912 more efficient for that particular implementation.

33913 **Destroying Mutexes**

33914 A mutex can be destroyed immediately after it is unlocked. For example, consider the following  
33915 code:

```
33916 struct obj {
33917 pthread_mutex_t om;
33918 int refcnt;
33919 ...
33920 };

33921 obj_done(struct obj *op)
33922 {
33923 pthread_mutex_lock(&op->om);
33924 if (--op->refcnt == 0) {
33925 pthread_mutex_unlock(&op->om);
33926 (A) pthread_mutex_destroy(&op->om);
33927 (B) free(op);
33928 } else
33929 (C) pthread_mutex_unlock(&op->om);
33930 }
```

33931 In this case *obj* is reference counted and *obj\_done()* is called whenever a reference to the object is  
33932 dropped. Implementations are required to allow an object to be destroyed and freed and  
33933 potentially unmapped (for example, lines A and B) immediately after the object is unlocked (line  
33934 C).

33935 **FUTURE DIRECTIONS**

33936 None.

33937 **SEE ALSO**

33938 *pthread\_mutex\_getprioceiling()*, *pthread\_mutex\_lock()*, *pthread\_mutex\_timedlock()*,  
33939 *pthread\_mutexattr\_getpshared()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
33940 <pthread.h>

33941 **CHANGE HISTORY**

33942 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33943 **Issue 6**

33944 The *pthread\_mutex\_destroy()* and *pthread\_mutex\_init()* functions are marked as part of the  
33945 Threads option.

33946 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with  
33947 IEEE Std. 1003.1d-1999.

33948 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

33949 The **restrict** keyword is added to the *pthread\_mutex\_init()* prototype for alignment with the  
33950 ISO/IEC 9899:1999 standard.

33951 **NAME**

33952 pthread\_mutex\_getprioceiling, pthread\_mutex\_setprioceiling — change the priority ceiling of a  
 33953 mutex (**REALTIME THREADS**)

33954 **SYNOPSIS**

```
33955 TPP #include <pthread.h>
33956 int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict mutex,
33957 int *restrict prioceiling);
33958 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
33959 int prioceiling, int *restrict old_ceiling);
33960
```

33961 **DESCRIPTION**

33962 The *pthread\_mutex\_getprioceiling()* function shall return the current priority ceiling of the mutex.

33963 The *pthread\_mutex\_setprioceiling()* function either locks the mutex if it is unlocked, or blocks until  
 33964 it can successfully lock the mutex, then it changes the mutex's priority ceiling and releases the  
 33965 mutex. When the change is successful, the previous value of the priority ceiling is returned in  
 33966 *old\_ceiling*. The process of locking the mutex need not adhere to the priority protect protocol.

33967 If the *pthread\_mutex\_setprioceiling()* function fails, the mutex priority ceiling shall not be  
 33968 changed.

33969 **RETURN VALUE**

33970 If successful, the *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions shall  
 33971 return zero; otherwise, an error number shall be returned to indicate the error.

33972 **ERRORS**

33973 The *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions may fail if:

33974 [EINVAL] The priority requested by *prioceiling* is out of range.

33975 [EINVAL] The value specified by *mutex* does not refer to a currently existing mutex.

33976 [EPERM] The caller does not have the privilege to perform the operation.

33977 These functions shall not return an error code of [EINTR].

33978 **EXAMPLES**

33979 None.

33980 **APPLICATION USAGE**

33981 None.

33982 **RATIONALE**

33983 None.

33984 **FUTURE DIRECTIONS**

33985 None.

33986 **SEE ALSO**

33987 *pthread\_mutex\_destroy()*, *pthread\_mutex\_lock()*, *pthread\_mutex\_timedlock()*, the Base Definitions  
 33988 volume of IEEE Std. 1003.1-200x, <pthread.h>

33989 **CHANGE HISTORY**

33990 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33991 Marked as part of the Realtime Threads Feature Group.



33992 **Issue 6**

33993 The *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions are marked as  
33994 part of the Thread Priority Protection option.

33995 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
33996 implementation does not support the Thread Priority Protection option.

33997 The [ENOSYS] error denoting non-support of the priority ceiling protocol for mutexes has been  
33998 removed. This is since if the implementation provides the functions (regardless of whether  
33999 `_POSIX_PTHREAD_PRIO_PROTECT` is defined), they must function as in the DESCRIPTION  
34000 and therefore the priority ceiling protocol for mutexes is supported.

34001 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with  
34002 IEEE Std. 1003.1d-1999.

34003 The **restrict** keyword is added to the *pthread\_mutex\_getprioceiling()* and  
34004 *pthread\_mutex\_setprioceiling()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

34005 **NAME**

34006 pthread\_mutex\_init — initialize a mutex

34007 **SYNOPSIS**

34008 THR #include <pthread.h>

34009 int pthread\_mutex\_init(pthread\_mutex\_t \*restrict mutex,

34010 const pthread\_mutexattr\_t \*restrict attr);

34011 pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

34012

34013 **DESCRIPTION**

34014 Refer to *pthread\_mutex\_destroy()*.

34015 **NAME**

34016 pthread\_mutex\_lock, pthread\_mutex\_trylock, pthread\_mutex\_unlock — lock and unlock a  
 34017 mutex

34018 **SYNOPSIS**

34019 THR #include <pthread.h>

34020 int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);

34021 int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);

34022 int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

34023

34024 **DESCRIPTION**

34025 The mutex object referenced by *mutex* shall be locked by calling *pthread\_mutex\_lock()*. If the  
 34026 mutex is already locked, the calling thread shall block until the mutex becomes available. This  
 34027 operation shall return with the mutex object referenced by *mutex* in the locked state with the  
 34028 calling thread as its owner.

34029 XSI If the mutex type is PTHREAD\_MUTEX\_NORMAL, deadlock detection shall not be provided.  
 34030 Attempting to relock the mutex causes deadlock. If a thread attempts to unlock a mutex that it  
 34031 has not locked or a mutex which is unlocked, undefined behavior results.

34032 If the mutex type is PTHREAD\_MUTEX\_ERRORCHECK, then error checking shall be provided.  
 34033 If a thread attempts to relock a mutex that it has already locked, an error is returned. If a thread  
 34034 attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error is  
 34035 returned.

34036 If the mutex type is PTHREAD\_MUTEX\_RECURSIVE, then the mutex shall maintain the  
 34037 concept of a lock count. When a thread successfully acquires a mutex for the first time, the lock  
 34038 count is set to one. Every time a thread relocks this mutex, the lock count is incremented by one.  
 34039 Each time the thread unlocks the mutex, the lock count is decremented by one. When the lock  
 34040 count reaches zero, the mutex becomes available for other threads to acquire. If a thread  
 34041 attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error is  
 34042 returned.

34043 If the mutex type is PTHREAD\_MUTEX\_DEFAULT, attempting to recursively lock the mutex  
 34044 results in undefined behavior. Attempting to unlock the mutex if it was not locked by the calling  
 34045 thread results in undefined behavior. Attempting to unlock the mutex if it is not locked results in  
 34046 undefined behavior.

34047 The *pthread\_mutex\_trylock()* function is identical to *pthread\_mutex\_lock()* except that if the mutex  
 34048 object referenced by *mutex* is currently locked (by any thread, including the current thread), the  
 34049 call shall return immediately.

34050 XSI The *pthread\_mutex\_unlock()* function releases the mutex object referenced by *mutex*. The manner  
 34051 in which a mutex is released is dependent upon the mutex's type attribute. If there are threads  
 34052 blocked on the mutex object referenced by *mutex* when *pthread\_mutex\_unlock()* is called,  
 34053 resulting in the mutex becoming available, the scheduling policy is used to determine which  
 34054 thread shall acquire the mutex.

34055 XSI (In the case of PTHREAD\_MUTEX\_RECURSIVE mutexes, the mutex shall become available  
 34056 when the count reaches zero and the calling thread no longer has any locks on this mutex).

34057 If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the  
 34058 thread shall resume waiting for the mutex as if it was not interrupted.

34059 **RETURN VALUE**

34060 If successful, the *pthread\_mutex\_lock()* and *pthread\_mutex\_unlock()* functions shall return zero;  
34061 otherwise, an error number shall be returned to indicate the error.

34062 The *pthread\_mutex\_trylock()* function shall return zero if a lock on the mutex object referenced by  
34063 *mutex* is acquired. Otherwise, an error number is returned to indicate the error.

34064 **ERRORS**

34065 The *pthread\_mutex\_lock()* and *pthread\_mutex\_trylock()* functions shall fail if:

34066 [EINVAL] The *mutex* was created with the protocol attribute having the value  
34067 PTHREAD\_PRIO\_PROTECT and the calling thread's priority is higher than  
34068 the mutex's current priority ceiling.

34069 The *pthread\_mutex\_trylock()* function shall fail if:

34070 [EBUSY] The *mutex* could not be acquired because it was already locked.

34071 The *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, and *pthread\_mutex\_unlock()* functions may  
34072 fail if:

34073 [EINVAL] The value specified by *mutex* does not refer to an initialized mutex object.

34074 XSI [EAGAIN] The mutex could not be acquired because the maximum number of recursive  
34075 locks for *mutex* has been exceeded.

34076 The *pthread\_mutex\_lock()* function may fail if:

34077 [EDEADLK] The current thread already owns the mutex.

34078 The *pthread\_mutex\_unlock()* function may fail if:

34079 [EPERM] The current thread does not own the mutex.

34080 These functions shall not return an error code of [EINTR].

34081 **EXAMPLES**

34082 None.

34083 **APPLICATION USAGE**

34084 None.

34085 **RATIONALE**

34086 Mutex objects are intended to serve as a low-level primitive from which other thread  
34087 synchronization functions can be built. As such, the implementation of mutexes should be as  
34088 efficient as possible, and this has ramifications on the features available at the interface.

34089 The mutex functions and the particular default settings of the mutex attributes have been  
34090 motivated by the desire to not preclude fast, inlined implementations of mutex locking and  
34091 unlocking.

34092 For example, deadlocking on a double-lock is explicitly allowed behavior in order to avoid  
34093 requiring more overhead in the basic mechanism than is absolutely necessary. (More "friendly"  
34094 mutexes that detect deadlock or that allow multiple locking by the same thread are easily  
34095 constructed by the user via the other mechanisms provided. For example, *pthread\_self()* can be  
34096 used to record mutex ownership.) Implementations might also choose to provide such extended  
34097 features as options via special mutex attributes.

34098 Since most attributes only need to be checked when a thread is going to be blocked, the use of  
34099 attributes does not slow the (common) mutex-locking case.

34100 Likewise, while being able to extract the thread ID of the owner of a mutex might be desirable, it  
34101 would require storing the current thread ID when each mutex is locked, and this could incur  
34102 unacceptable levels of overhead. Similar arguments apply to a *mutex\_tryunlock* operation.

34103 **FUTURE DIRECTIONS**

34104 None.

34105 **SEE ALSO**

34106 *pthread\_mutex\_destroy()*, *pthread\_mutex\_timedlock()*, the Base Definitions volume of  
34107 IEEE Std. 1003.1-200x, <pthread.h>

34108 **CHANGE HISTORY**

34109 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34110 **Issue 6**

34111 The *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, and *pthread\_mutex\_unlock()* functions are  
34112 marked as part of the Threads option.

34113 The following new requirements on POSIX implementations derive from alignment with the  
34114 Single UNIX Specification:

- 34115 • The behavior when attempting to relock a mutex is defined.

34116 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with  
34117 IEEE Std. 1003.1d-1999.

34118 **NAME**

34119 pthread\_mutex\_setprioceiling — change the priority ceiling of a mutex (**REALTIME**  
34120 **THREADS**)

34121 **SYNOPSIS**

34122 TPP #include <pthread.h>

34123 int pthread\_mutex\_setprioceiling(pthread\_mutex\_t \*restrict mutex,  
34124 int prioceiling, int \*restrict old\_ceiling);

34125

34126 **DESCRIPTION**

34127 Refer to *pthread\_mutex\_getprioceiling()*.

34128 **NAME**34129 pthread\_mutex\_timedlock — lock a mutex (**REALTIME THREADS**)34130 **SYNOPSIS**

34131 THR TMO #include &lt;pthread.h&gt;

34132 #include &lt;time.h&gt;

34133 int pthread\_mutex\_timedlock(pthread\_mutex\_t \*restrict mutex,

34134 const struct timespec \*restrict abs\_timeout);

34135

34136 **DESCRIPTION**

34137 The *pthread\_mutex\_timedlock()* function is called to lock the mutex object referenced by *mutex*. If  
 34138 the mutex is already locked, the calling thread blocks until the mutex becomes available as in the  
 34139 *pthread\_mutex\_lock()* function. If the mutex cannot be locked without waiting for another thread  
 34140 to unlock the mutex, this wait shall be terminated when the specified timeout expires.

34141 The timeout expires when the absolute time specified by *abs\_timeout* passes, as measured by the  
 34142 clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 34143 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already been passed at the time  
 34144 of the call. If the Timers option is supported, the timeout is based on the CLOCK\_REALTIME  
 34145 clock; if the Timers option is not supported, the timeout is based on the system clock as returned  
 34146 by the *time()* function. The resolution of the timeout is the resolution of the clock on which it is  
 34147 based. The **timespec** datatype is defined as a structure in the <time.h> header.

34148 Under no circumstance will the function fail with a timeout if the mutex can be locked  
 34149 immediately. The validity of the *abs\_timeout* parameter need not be checked if the mutex can be  
 34150 locked immediately.

34151 As a consequence of the priority inheritance rules (for mutexes initialized with the  
 34152 PRIO\_INHERIT protocol), if a timed mutex wait is terminated because its timeout expires, the  
 34153 priority of the owner of the mutex will be adjusted as necessary to reflect the fact that this thread  
 34154 is no longer among the threads waiting for the mutex.

34155 **RETURN VALUE**

34156 If successful, the *pthread\_mutex\_timedlock()* function shall return zero; otherwise, an error  
 34157 number shall be returned to indicate the error.

34158 **ERRORS**

34159 The *pthread\_mutex\_timedlock()* function shall fail if:

34160 [EINVAL] The mutex was created with the protocol attribute having the value  
 34161 PTHREAD\_PRIO\_PROTECT and the calling thread's priority is higher than  
 34162 the mutex' current priority ceiling.

34163 [EINVAL] The process or thread would have blocked, and the *abs\_timeout* parameter  
 34164 specified a nanoseconds field value less than zero or greater than or equal to  
 34165 1 000 million.

34166 [ETIMEDOUT] The mutex could not be locked before the specified timeout expired.

34167 The *pthread\_mutex\_timedlock()* function may fail if:

34168 [EINVAL] The value specified by *mutex* does not refer to an initialized mutex object.

34169 XSI [EAGAIN] The mutex could not be acquired because the maximum number of recursive  
 34170 locks for mutex has been exceeded.

34171 [EDEADLK] The current thread already owns the mutex.

34172 This function shall not return an error code of [EINTR].

34173 **EXAMPLES**

34174 None.

34175 **APPLICATION USAGE**

34176 The *pthread\_mutex\_timedlock()* function is part of the Threads and Timeouts options and need  
34177 not be provided on all implementations.

34178 **RATIONALE**

34179 None.

34180 **FUTURE DIRECTIONS**

34181 None.

34182 **SEE ALSO**

34183 *pthread\_mutex\_destroy()*, *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, *time()*, the Base  
34184 Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>, <time.h>

34185 **CHANGE HISTORY**

34186 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.



34187 **NAME**

34188 pthread\_mutex\_trylock, pthread\_mutex\_unlock — lock and unlock a mutex

34189 **SYNOPSIS**

34190 THR #include &lt;pthread.h&gt;

34191 int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);

34192 int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

34193

34194 **DESCRIPTION**34195 Refer to *pthread\_mutex\_lock()*.

## 34196 NAME

34197 pthread\_mutexattr\_destroy, pthread\_mutexattr\_init — destroy and initialize mutex attributes  
34198 object

## 34199 SYNOPSIS

```
34200 THR #include <pthread.h>
```

```
34201 int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
```

```
34202 int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

34203

## 34204 DESCRIPTION

34205 The *pthread\_mutexattr\_destroy()* function destroys a mutex attributes object; the object becomes,  
34206 in effect, uninitialized. An implementation may cause *pthread\_mutexattr\_destroy()* to set the  
34207 object referenced by *attr* to an invalid value. A destroyed mutex attributes object can be re-  
34208 initialized using *pthread\_mutexattr\_init()*; the results of otherwise referencing the object after it  
34209 has been destroyed are undefined.

34210 The *pthread\_mutexattr\_init()* function initializes a mutex attributes object *attr* with the default  
34211 value for all of the attributes defined by the implementation.

34212 The effect of initializing an already initialized mutex attributes object is undefined.

34213 After a mutex attributes object has been used to initialize one or more mutexes, any function  
34214 affecting the attributes object (including destruction) does not affect any previously initialized  
34215 mutexes.

## 34216 RETURN VALUE

34217 Upon successful completion, *pthread\_mutexattr\_destroy()* and *pthread\_mutexattr\_init()* shall  
34218 return zero; otherwise, an error number shall be returned to indicate the error.

## 34219 ERRORS

34220 The *pthread\_mutexattr\_destroy()* function may fail if:

34221 [EINVAL] The value specified by *attr* is invalid.

34222 The *pthread\_mutexattr\_init()* function shall fail if:

34223 [ENOMEM] Insufficient memory exists to initialize the mutex attributes object.

34224 These functions shall not return an error code of [EINTR].

## 34225 EXAMPLES

34226 None.

## 34227 APPLICATION USAGE

34228 None.

## 34229 RATIONALE

34230 See *pthread\_attr\_init()* for a general explanation of attributes. Attributes objects allow  
34231 implementations to experiment with useful extensions and permit extension of this volume of  
34232 IEEE Std. 1003.1-200x without changing the existing functions. Thus, they provide for future  
34233 extensibility of this volume of IEEE Std. 1003.1-200x and reduce the temptation to standardize  
34234 prematurely on semantics that are not yet widely implemented or understood.

34235 Examples of possible additional mutex attributes that have been discussed are *spin\_only*,  
34236 *limited\_spin*, *no\_spin*, *recursive*, and *metered*. (To explain what the latter attributes might mean:  
34237 recursive mutexes would allow for multiple re-locking by the current owner; metered mutexes  
34238 would transparently keep records of queue length, wait time, and so on.) Since there is not yet  
34239 wide agreement on the usefulness of these resulting from shared implementation and usage

34240 experience, they are not yet specified in this volume of IEEE Std. 1003.1-200x. Mutex attributes  
 34241 objects, however, make it possible to test out these concepts for possible standardization at a  
 34242 later time.

### 34243 **Mutex Attributes and Performance**

34244 Care has been taken to ensure that the default values of the mutex attributes have been defined  
 34245 such that mutexes initialized with the defaults have simple enough semantics so that the locking  
 34246 and unlocking can be done with the equivalent of a test-and-set instruction (plus possibly a few  
 34247 other basic instructions).

34248 There is at least one implementation method that can be used to reduce the cost of testing at  
 34249 lock-time if a mutex has non-default attributes. One such method that an implementation can  
 34250 employ (and this can be made fully transparent to fully conforming POSIX applications) is to  
 34251 secretly pre-lock any mutexes that are initialized to non-default attributes. Any later attempt to  
 34252 lock such a mutex causes the implementation to branch to the “slow path” as if the mutex were  
 34253 unavailable; then, on the slow path, the implementation can do the “real work” to lock a non-  
 34254 default mutex. The underlying unlock operation is more complicated since the implementation  
 34255 never really wants to release the pre-lock on this kind of mutex. This illustrates that, depending  
 34256 on the hardware, there may be certain optimizations that can be used so that whatever mutex  
 34257 attributes are considered “most frequently used” can be processed most efficiently.

### 34258 **Process Shared Memory and Synchronization**

34259 The existence of memory mapping functions in this volume of IEEE Std. 1003.1-200x leads to the  
 34260 possibility that an application may allocate the synchronization objects from this section in  
 34261 memory that is accessed by multiple processes (and therefore, by threads of multiple processes).

34262 In order to permit such usage, while at the same time keeping the usual case (that is, usage  
 34263 within a single process) efficient, a process-shared option has been defined.

34264 If an implementation supports the `_POSIX_THREAD_PROCESS_SHARED` option, then the  
 34265 *process-shared* attribute can be used to indicate that mutexes or condition variables may be  
 34266 accessed by threads of multiple processes.

34267 The default setting of `PTHREAD_PROCESS_PRIVATE` has been chosen for the *process-shared*  
 34268 attribute so that the most efficient forms of these synchronization objects are created by default.

34269 Synchronization variables that are initialized with the `PTHREAD_PROCESS_PRIVATE` *process-*  
 34270 *shared* attribute may only be operated on by threads in the process that initialized them.  
 34271 Synchronization variables that are initialized with the `PTHREAD_PROCESS_SHARED` *process-*  
 34272 *shared* attribute may be operated on by any thread in any process that has access to it. In  
 34273 particular, these processes may exist beyond the lifetime of the initializing process. For example,  
 34274 the following code implements a simple counting semaphore in a mapped file that may be used  
 34275 by many processes.

```
34276 /* sem.h */
34277 struct semaphore {
34278 pthread_mutex_t lock;
34279 pthread_cond_t nonzero;
34280 unsigned count;
34281 };
34282 typedef struct semaphore semaphore_t;
34283 semaphore_t *semaphore_create(char *semaphore_name);
34284 semaphore_t *semaphore_open(char *semaphore_name);
34285 void semaphore_post(semaphore_t *semap);
```

```
34286 void semaphore_wait(semaphore_t *semap);
34287 void semaphore_close(semaphore_t *semap);

34288 /* sem.c */
34289 #include <sys/types.h>
34290 #include <sys/stat.h>
34291 #include <sys/mman.h>
34292 #include <fcntl.h>
34293 #include <pthread.h>
34294 #include "sem.h"

34295 semaphore_t *
34296 semaphore_create(char *semaphore_name)
34297 {
34298 int fd;
34299 semaphore_t *semap;
34300 pthread_mutexattr_t psharedm;
34301 pthread_condattr_t psharedc;

34302 fd = open(semaphore_name, O_RDWR | O_CREAT | O_EXCL, 0666);
34303 if (fd < 0)
34304 return (NULL);
34305 (void) ftruncate(fd, sizeof(semaphore_t));
34306 (void) pthread_mutexattr_init(&psharedm);
34307 (void) pthread_mutexattr_setpshared(&psharedm,
34308 PTHREAD_PROCESS_SHARED);
34309 (void) pthread_condattr_init(&psharedc);
34310 (void) pthread_condattr_setpshared(&psharedc,
34311 PTHREAD_PROCESS_SHARED);
34312 semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
34313 PROT_READ | PROT_WRITE, MAP_SHARED,
34314 fd, 0);
34315 close (fd);
34316 (void) pthread_mutex_init(&semap->lock, &psharedm);
34317 (void) pthread_cond_init(&semap->nonzero, &psharedc);
34318 semap->count = 0;
34319 return (semap);
34320 }

34321 semaphore_t *
34322 semaphore_open(char *semaphore_name)
34323 {
34324 int fd;
34325 semaphore_t *semap;

34326 fd = open(semaphore_name, O_RDWR, 0666);
34327 if (fd < 0)
34328 return (NULL);
34329 semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
34330 PROT_READ | PROT_WRITE, MAP_SHARED,
34331 fd, 0);
34332 close (fd);
34333 return (semap);
34334 }
```

```

34335 void
34336 semaphore_post(semaphore_t *semap)
34337 {
34338 pthread_mutex_lock(&semap->lock);
34339 if (semap->count == 0)
34340 pthread_cond_signal(&semap->nonzero);
34341 semap->count++;
34342 pthread_mutex_unlock(&semap->lock);
34343 }
34344 void
34345 semaphore_wait(semaphore_t *semap)
34346 {
34347 pthread_mutex_lock(&semap->lock);
34348 while (semap->count == 0)
34349 pthread_cond_wait(&semap->nonzero, &semap->lock);
34350 semap->count--;
34351 pthread_mutex_unlock(&semap->lock);
34352 }
34353 void
34354 semaphore_close(semaphore_t *semap)
34355 {
34356 munmap((void *) semap, sizeof(semaphore_t));
34357 }

```

34358 The following code is for three separate processes that create, post, and wait on a semaphore in  
34359 the file **/tmp/semaphore**. Once the file is created, the post and wait programs increment and  
34360 decrement the counting semaphore (waiting and waking as required) even though they did not  
34361 initialize the semaphore.

```

34362 /* create.c */
34363 #include "pthread.h"
34364 #include "sem.h"
34365 int
34366 main()
34367 {
34368 semaphore_t *semap;
34369 semap = semaphore_create("/tmp/semaphore");
34370 if (semap == NULL)
34371 exit(1);
34372 semaphore_close(semap);
34373 return (0);
34374 }
34375 /* post */
34376 #include "pthread.h"
34377 #include "sem.h"
34378 int
34379 main()
34380 {
34381 semaphore_t *semap;

```

```
34382 semap = semaphore_open("/tmp/semaphore");
34383 if (semap == NULL)
34384 exit(1);
34385 semaphore_post(semap);
34386 semaphore_close(semap);
34387 return (0);
34388 }
34389 /* wait */
34390 #include "pthread.h"
34391 #include "sem.h"
34392 int
34393 main()
34394 {
34395 semaphore_t *semap;
34396
34397 semap = semaphore_open("/tmp/semaphore");
34398 if (semap == NULL)
34399 exit(1);
34399 semaphore_wait(semap);
34400 semaphore_close(semap);
34401 return (0);
34402 }
```

**34403 FUTURE DIRECTIONS**

34404 None.

**34405 SEE ALSO**

34406 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, *pthread\_mutexattr\_destroy()*, the  
34407 Base Definitions volume of IEEE Std. 1003.1-200x, <**pthread.h**>

**34408 CHANGE HISTORY**

34409 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**34410 Issue 6**

34411 The *pthread\_mutexattr\_destroy()* and *pthread\_mutexattr\_init()* functions are marked as part of the  
34412 Threads option.

34413 IEEE PASC Interpretation 1003.1c #27 is applied, updating the ERRORS section.

34414 **NAME**

34415 pthread\_mutexattr\_getprioceiling, pthread\_mutexattr\_setprioceiling — get and set prioceiling  
 34416 attribute of mutex attributes object (**REALTIME THREADS**)

34417 **SYNOPSIS**

```
34418 TPP #include <pthread.h>
34419 int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t *restrict attr, |
34420 int *restrict prioceiling); |
34421 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr, |
34422 int prioceiling); |
34423
```

34424 **DESCRIPTION**

34425 The *pthread\_mutexattr\_getprioceiling()* and *pthread\_mutexattr\_setprioceiling()* functions,  
 34426 respectively, get and set the priority ceiling attribute of a mutex attributes object pointed to by  
 34427 *attr* which was previously created by the function *pthread\_mutexattr\_init()*.

34428 The *prioceiling* attribute contains the priority ceiling of initialized mutexes. The values of  
 34429 *prioceiling* are within the maximum range of priorities defined by SCHED\_FIFO.

34430 The *prioceiling* attribute defines the priority ceiling of initialized mutexes, which is the minimum  
 34431 priority level at which the critical section guarded by the mutex is executed. In order to avoid  
 34432 priority inversion, the priority ceiling of the mutex is set to a priority higher than or equal to the  
 34433 highest priority of all the threads that may lock that mutex. The values of *prioceiling* are within  
 34434 the maximum range of priorities defined under the SCHED\_FIFO scheduling policy.

34435 **RETURN VALUE**

34436 Upon successful completion, the *pthread\_mutexattr\_getprioceiling()* and  
 34437 *pthread\_mutexattr\_setprioceiling()* functions shall return zero; otherwise, an error number shall be  
 34438 returned to indicate the error.

34439 **ERRORS**

34440 The *pthread\_mutexattr\_getprioceiling()* and *pthread\_mutexattr\_setprioceiling()* functions may fail if:

34441 [EINVAL] The value specified by *attr* or *prioceiling* is invalid. |

34442 [EPERM] The caller does not have the privilege to perform the operation. |

34443 These functions shall not return an error code of [EINTR]. |

34444 **EXAMPLES**

34445 None.

34446 **APPLICATION USAGE**

34447 None.

34448 **RATIONALE**

34449 None.

34450 **FUTURE DIRECTIONS**

34451 None.

34452 **SEE ALSO**

34453 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, the Base Definitions volume of  
 34454 IEEE Std. 1003.1-200x, <pthread.h>

## 34455 CHANGE HISTORY

34456 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34457 Marked as part of the Realtime Threads Feature Group.

## 34458 Issue 6

34459 The `pthread_mutexattr_getprioceiling()` and `pthread_mutexattr_setprioceiling()` functions are  
34460 marked as part of the Thread Priority Protection option.

34461 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
34462 implementation does not support the Thread Priority Protection option.

34463 The [ENOTSUP] error condition has been removed since these functions do not have a *protocol*  
34464 argument.

34465 The **restrict** keyword is added to the `pthread_mutexattr_getprioceiling()` prototype for alignment  
34466 with the ISO/IEC 9899:1999 standard.



34467 **NAME**

34468 pthread\_mutexattr\_getprotocol, pthread\_mutexattr\_setprotocol — get and set protocol attribute  
 34469 of mutex attributes object (**REALTIME THREADS**)

34470 **SYNOPSIS**

```
34471 TPP|TPI #include <pthread.h>
34472 int pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict attr, |
34473 int *restrict protocol); |
34474 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr, |
34475 int protocol); |
34476
```

34477 **DESCRIPTION**

34478 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions, respectively,  
 34479 get and set the protocol attribute of a mutex attributes object pointed to by *attr* which was  
 34480 previously created by the function *pthread\_mutexattr\_init()*.

34481 The *protocol* attribute defines the protocol to be followed in utilizing mutexes. The value of  
 34482 *protocol* may be one of `PTHREAD_PRIO_NONE`, `PTHREAD_PRIO_INHERIT`, or  
 34483 `PTHREAD_PRIO_PROTECT`, which are defined by the header `<pthread.h>`.

34484 When a thread owns a mutex with the `PTHREAD_PRIO_NONE` *protocol* attribute, its priority  
 34485 and scheduling are not affected by its mutex ownership.

34486 TPI When a thread is blocking higher priority threads because of owning one or more mutexes with  
 34487 the `PTHREAD_PRIO_INHERIT` protocol attribute, it executes at the higher of its priority or the  
 34488 priority of the highest priority thread waiting on any of the mutexes owned by this thread and  
 34489 initialized with this protocol.

34490 TPP When a thread owns one or more mutexes initialized with the `PTHREAD_PRIO_PROTECT`  
 34491 protocol, it executes at the higher of its priority or the highest of the priority ceilings of all the  
 34492 mutexes owned by this thread and initialized with this attribute, regardless of whether other  
 34493 threads are blocked on any of these mutexes or not.

34494 While a thread is holding a mutex which has been initialized with the  
 34495 `PTHREAD_PRIO_INHERIT` or `PTHREAD_PRIO_PROTECT` protocol attributes, it shall not be  
 34496 subject to being moved to the tail of the scheduling queue at its priority in the event that its  
 34497 original priority is changed, such as by a call to *sched\_setparam()*. Likewise, when a thread  
 34498 unlocks a mutex that has been initialized with the `PTHREAD_PRIO_INHERIT` or  
 34499 `PTHREAD_PRIO_PROTECT` protocol attributes, it shall not be subject to being moved to the tail  
 34500 of the scheduling queue at its priority in the event that its original priority is changed.

34501 If a thread simultaneously owns several mutexes initialized with different protocols, it shall  
 34502 execute at the highest of the priorities that it would have obtained by each of these protocols.

34503 TPI When a thread makes a call to *pthread\_mutex\_lock()*, the mutex was initialized with the protocol  
 34504 attribute having the value `PTHREAD_PRIO_INHERIT`, when the calling thread is blocked  
 34505 because the mutex is owned by another thread, that owner thread shall inherit the priority level  
 34506 of the calling thread as long as it continues to own the mutex. The implementation shall update  
 34507 its execution priority to the maximum of its assigned priority and all its inherited priorities.  
 34508 Furthermore, if this owner thread itself becomes blocked on another mutex, the same priority  
 34509 inheritance effect shall be propagated to this other owner thread, in a recursive manner.

34510 **RETURN VALUE**

34511 Upon successful completion, the *pthread\_mutexattr\_getprotocol()* and  
34512 *pthread\_mutexattr\_setprotocol()* functions shall return zero; otherwise, an error number shall be  
34513 returned to indicate the error.

34514 **ERRORS**

34515 The *pthread\_mutexattr\_setprotocol()* function shall fail if:

34516 [ENOTSUP] The value specified by *protocol* is an unsupported value.

34517 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions may fail if:

34518 [EINVAL] The value specified by *attr* or *protocol* is invalid.

34519 [EPERM] The caller does not have the privilege to perform the operation.

34520 These functions shall not return an error code of [EINTR].

34521 **EXAMPLES**

34522 None.

34523 **APPLICATION USAGE**

34524 None.

34525 **RATIONALE**

34526 None.

34527 **FUTURE DIRECTIONS**

34528 None.

34529 **SEE ALSO**

34530 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, the Base Definitions volume of  
34531 IEEE Std. 1003.1-200x, <pthread.h>

34532 **CHANGE HISTORY**

34533 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34534 Marked as part of the Realtime Threads Feature Group.

34535 **Issue 6**

34536 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions are marked as  
34537 part of either the Thread Priority Protection or Threads Priority Inheritance options.

34538 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
34539 implementation does not support the Thread Priority Protection or Threads Priority Inheritance  
34540 options.

34541 The **restrict** keyword is added to the *pthread\_mutexattr\_getprotocol()* prototype for alignment  
34542 with the ISO/IEC 9899:1999 standard.

34543 **NAME**

34544 pthread\_mutexattr\_getpshared, pthread\_mutexattr\_setpshared — get and set process-shared  
 34545 attribute

34546 **SYNOPSIS**

```
34547 THR TSH #include <pthread.h>
34548 int pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict attr, |
34549 int *restrict pshared); |
34550 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr, |
34551 int pshared); |
34552
```

34553 **DESCRIPTION**

34554 The *pthread\_mutexattr\_getpshared()* function obtains the value of the *process-shared* attribute from  
 34555 the attributes object referenced by *attr*. The *pthread\_mutexattr\_setpshared()* function is used to set  
 34556 the *process-shared* attribute in an initialized attributes object referenced by *attr*.

34557 The *process-shared* attribute is set to PTHREAD\_PROCESS\_SHARED to permit a mutex to be  
 34558 operated upon by any thread that has access to the memory where the mutex is allocated, even if  
 34559 the mutex is allocated in memory that is shared by multiple processes. If the *process-shared*  
 34560 attribute is PTHREAD\_PROCESS\_PRIVATE, the mutex shall only be operated upon by threads  
 34561 created within the same process as the thread that initialized the mutex; if threads of differing  
 34562 processes attempt to operate on such a mutex, the behavior is undefined. The default value of  
 34563 the attribute shall be PTHREAD\_PROCESS\_PRIVATE.

34564 **RETURN VALUE**

34565 Upon successful completion, *pthread\_mutexattr\_setpshared()* shall return zero; otherwise, an error  
 34566 number shall be returned to indicate the error.

34567 Upon successful completion, *pthread\_mutexattr\_getpshared()* shall return zero and stores the  
 34568 value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter.  
 34569 Otherwise, an error number shall be returned to indicate the error.

34570 **ERRORS**

34571 The *pthread\_mutexattr\_getpshared()* and *pthread\_mutexattr\_setpshared()* functions may fail if:

34572 [EINVAL] The value specified by *attr* is invalid.

34573 The *pthread\_mutexattr\_setpshared()* function may fail if:

34574 [EINVAL] The new value specified for the attribute is outside the range of legal values  
 34575 for that attribute.

34576 These functions shall not return an error code of [EINTR].

34577 **EXAMPLES**

34578 None.

34579 **APPLICATION USAGE**

34580 None.

34581 **RATIONALE**

34582 None.

34583 **FUTURE DIRECTIONS**

34584 None.

34585 **SEE ALSO**

34586 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, *pthread\_mutexattr\_destroy()*, the  
34587 Base Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

34588 **CHANGE HISTORY**

34589 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34590 **Issue 6**

34591 The *pthread\_mutexattr\_getpshared()* and *pthread\_mutexattr\_setpshared()* functions are marked as  
34592 part of the Threads and Thread Process-Shared Synchronization options.

34593 The **restrict** keyword is added to the *pthread\_mutexattr\_getpshared()* prototype for alignment  
34594 with the ISO/IEC 9899:1999 standard.

34595 **NAME**

34596 pthread\_mutexattr\_gettype, pthread\_mutexattr\_settype — get or set a mutex type

34597 **SYNOPSIS**

34598 XSI #include &lt;pthread.h&gt;

34599 int pthread\_mutexattr\_gettype(const pthread\_mutexattr\_t \*restrict attr,  
34600 int \*restrict type);

34601 int pthread\_mutexattr\_settype(pthread\_mutexattr\_t \*attr, int type);

34602

34603 **DESCRIPTION**34604 The *pthread\_mutexattr\_gettype()* and *pthread\_mutexattr\_settype()* functions, respectively, get and  
34605 set the mutex *type* attribute. This attribute is set in the *type* parameter to these functions. The  
34606 default value of the *type* attribute is PTHREAD\_MUTEX\_DEFAULT.34607 The type of mutex is contained in the *type* attribute of the mutex attributes. Valid mutex types  
34608 include:

## 34609 PTHREAD\_MUTEX\_NORMAL

34610 This type of mutex does not detect deadlock. A thread attempting to relock this mutex  
34611 without first unlocking it shall deadlock. Attempting to unlock a mutex locked by a  
34612 different thread results in undefined behavior. Attempting to unlock an unlocked mutex  
34613 results in undefined behavior.

## 34614 PTHREAD\_MUTEX\_ERRORCHECK

34615 This type of mutex provides error checking. A thread attempting to relock this mutex  
34616 without first unlocking it shall return with an error. A thread attempting to unlock a mutex  
34617 which another thread has locked shall return with an error. A thread attempting to unlock  
34618 an unlocked mutex shall return with an error.

## 34619 PTHREAD\_MUTEX\_RECURSIVE

34620 A thread attempting to relock this mutex without first unlocking it shall succeed in locking  
34621 the mutex. The relocking deadlock which can occur with mutexes of type  
34622 PTHREAD\_MUTEX\_NORMAL cannot occur with this type of mutex. Multiple locks of this  
34623 mutex require the same number of unlocks to release the mutex before another thread can  
34624 acquire the mutex. A thread attempting to unlock a mutex which another thread has locked  
34625 shall return with an error. A thread attempting to unlock an unlocked mutex shall return  
34626 with an error.

## 34627 PTHREAD\_MUTEX\_DEFAULT

34628 Attempting to recursively lock a mutex of this type results in undefined behavior.  
34629 Attempting to unlock a mutex of this type which was not locked by the calling thread  
34630 results in undefined behavior. Attempting to unlock a mutex of this type which is not  
34631 locked results in undefined behavior. An implementation is allowed to map this mutex to  
34632 one of the other mutex types.34633 **RETURN VALUE**34634 Upon successful completion, the *pthread\_mutexattr\_gettype()* function shall return zero and store  
34635 the value of the *type* attribute of *attr* into the object referenced by the *type* parameter. Otherwise,  
34636 an error shall be returned to indicate the error.34637 If successful, the *pthread\_mutexattr\_settype()* function shall return zero; otherwise, an error  
34638 number shall be returned to indicate the error.

34639 **ERRORS**

34640 The *pthread\_mutexattr\_settype()* function shall fail if:

34641 [EINVAL] The value *type* is invalid.

34642 The *pthread\_mutexattr\_gettype()* and *pthread\_mutexattr\_settype()* functions may fail if:

34643 [EINVAL] The value specified by *attr* is invalid.

34644 These functions shall not return an error code of [EINTR].

34645 **EXAMPLES**

34646 None.

34647 **APPLICATION USAGE**

34648 It is advised that an application should not use a PTHREAD\_MUTEX\_RECURSIVE mutex with  
34649 condition variables because the implicit unlock performed for a *pthread\_cond\_timedwait()* or  
34650 *pthread\_cond\_wait()* may not actually release the mutex (if it had been locked multiple times). If  
34651 this happens, no other thread can satisfy the condition of the predicate.

34652 **RATIONALE**

34653 None.

34654 **FUTURE DIRECTIONS**

34655 None.

34656 **SEE ALSO**

34657 *pthread\_cond\_timedwait()*, *pthread\_cond\_wait()*, the Base Definitions volume of  
34658 IEEE Std. 1003.1-200x, <pthread.h>

34659 **CHANGE HISTORY**

34660 First released in Issue 5.

34661 **Issue 6**

34662 The Open Group corrigenda item U033/3 has been applied. The SYNOPSIS for  
34663 *pthread\_mutexattr\_gettype()* is updated so that the first argument is of type  
34664 **constpthread\_mutexattr\_t\***.

34665 The **restrict** keyword is added to the *pthread\_mutexattr\_gettype()* prototype for alignment with  
34666 the ISO/IEC 9899:1999 standard.

34667 **NAME**

34668 pthread\_mutexattr\_init — initialize mutex attributes object

34669 **SYNOPSIS**

34670 THR #include &lt;pthread.h&gt;

34671 int pthread\_mutexattr\_init(pthread\_mutexattr\_t \*attr);

34672

34673 **DESCRIPTION**34674 Refer to *pthread\_mutexattr\_destroy()*.

34675 **NAME**

34676 pthread\_mutexattr\_setprioceiling — set prioceiling attribute of mutex attributes object  
34677 (**REALTIME THREADS**)

34678 **SYNOPSIS**

34679 TPP #include <pthread.h>

```
34680 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
34681 int prioceiling);
```

34682

34683 **DESCRIPTION**

34684 Refer to *pthread\_mutexattr\_getprioceiling()*.



34685 **NAME**

34686 pthread\_mutexattr\_setprotocol — set protocol attribute of mutex attributes object (**REALTIME**  
34687 **THREADS**)

34688 **SYNOPSIS**

34689 TPP|TPI #include <pthread.h>

```
34690 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
34691 int protocol);
```

34692

34693 **DESCRIPTION**

34694 Refer to *pthread\_mutexattr\_setprotocol()*.

34695 **NAME**

34696 pthread\_mutexattr\_setpshared — set process-shared attribute

34697 **SYNOPSIS**

34698 THR TSH #include <pthread.h>

```
34699 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
34700 int pshared);
```

34701

34702 **DESCRIPTION**

34703 Refer to *pthread\_mutexattr\_getpshared()*.

34704 **NAME**

34705 pthread\_mutexattr\_settype — set a mutex type

34706 **SYNOPSIS**34707 XSI `#include <pthread.h>`34708 `int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);`

34709

34710 **DESCRIPTION**34711 Refer to *pthread\_mutexattr\_gettype()*.

## 34712 NAME

34713 pthread\_once — dynamic package initialization

## 34714 SYNOPSIS

34715 THR #include &lt;pthread.h&gt;

```

34716 int pthread_once(pthread_once_t *once_control,
34717 void (*init_routine)(void));
34718 pthread_once_t once_control = PTHREAD_ONCE_INIT;
34719
```

## 34720 DESCRIPTION

34721 The first call to *pthread\_once()* by any thread in a process, with a given *once\_control*, shall call the  
 34722 *init\_routine* with no arguments. Subsequent calls of *pthread\_once()* with the same *once\_control*  
 34723 shall not call the *init\_routine*. On return from *pthread\_once()*, it is guaranteed that *init\_routine* has  
 34724 completed. The *once\_control* parameter is used to determine whether the associated initialization  
 34725 routine has been called.

34726 The *pthread\_once()* function is not a cancellation point. However, if *init\_routine* is a cancellation  
 34727 point and is canceled, the effect on *once\_control* shall be as if *pthread\_once()* was never called.

34728 The constant PTHREAD\_ONCE\_INIT is defined by the header <pthread.h>.

34729 The behavior of *pthread\_once()* is undefined if *once\_control* has automatic storage duration or is  
 34730 not initialized by PTHREAD\_ONCE\_INIT.

## 34731 RETURN VALUE

34732 Upon successful completion, *pthread\_once()* shall return zero; otherwise, an error number shall  
 34733 be returned to indicate the error.

## 34734 ERRORS

34735 **Notes to Reviewers**

34736 *This section with side shading will not appear in the final copy. - Ed.*

34737 D1, XSH, ERN 255 notes that no error is returned for invalid parameters and proposes the  
 34738 following:

34739 [EINVAL] If either *once\_control* or *init\_routine* is invalid.

34740 No errors are defined.

34741 The *pthread\_once()* function shall not return an error code of [EINTR].

## 34742 EXAMPLES

34743 None.

## 34744 APPLICATION USAGE

34745 None.

## 34746 RATIONALE

34747 Some C libraries are designed for dynamic initialization. That is, the global initialization for the  
 34748 library is performed when the first procedure in the library is called. In a single-threaded  
 34749 program, this is normally implemented using a static variable whose value is checked on entry  
 34750 to a routine, as follows:

```

34751 static int random_is_initialized = 0;
34752 extern int initialize_random();

```

```

34753 int random_function()
34754 {
34755 if (random_is_initialized == 0) {
34756 initialize_random();
34757 random_is_initialized = 1;
34758 }
34759 ... /* Operations performed after initialization. */
34760 }

```

34761 To keep the same structure in a multi-threaded program, a new primitive is needed. Otherwise,  
 34762 library initialization has to be accomplished by an explicit call to a library-exported initialization  
 34763 function prior to any use of the library.

34764 For dynamic library initialization in a multi-threaded process, a simple initialization flag is not  
 34765 sufficient; the flag needs to be protected against modification by multiple threads  
 34766 simultaneously calling into the library. Protecting the flag requires the use of a mutex; however,  
 34767 mutexes have to be initialized before they are used. Ensuring that the mutex is only initialized  
 34768 once requires a recursive solution to this problem.

34769 The use of *pthread\_once()* not only supplies an implementation-guaranteed means of dynamic  
 34770 initialization, it provides an aid to the reliable construction of multi-threaded and realtime  
 34771 systems. The preceding example then becomes:

```

34772 #include <pthread.h>
34773 static pthread_once_t random_is_initialized = PTHREAD_ONCE_INIT;
34774 extern int initialize_random();
34775
34776 int random_function()
34777 {
34778 (void) pthread_once(&random_is_initialized, initialize_random);
34779 ... /* Operations performed after initialization. */
34780 }

```

34780 Note that a **pthread\_once\_t** cannot be an array because some compilers do not accept the  
 34781 construct **&<array\_name>**.

#### 34782 FUTURE DIRECTIONS

34783 None.

#### 34784 SEE ALSO

34785 The Base Definitions volume of IEEE Std. 1003.1-200x, **<pthread.h>**

#### 34786 CHANGE HISTORY

34787 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 34788 Issue 6

34789 The *pthread\_once()* function is marked as part of the Threads option.

## 34790 NAME

34791 pthread\_rwlock\_destroy, pthread\_rwlock\_init — destroy and initialize a read-write lock object

## 34792 SYNOPSIS

34793 THR #include <pthread.h>

```
34794 int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
34795 int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
34796 const pthread_rwlockattr_t *restrict attr);
34797
```

## 34798 DESCRIPTION

34799 The *pthread\_rwlock\_destroy()* function shall destroy the read-write lock object referenced by  
 34800 *rwlock* and release any resources used by the lock. The effect of subsequent use of the lock is  
 34801 undefined until the lock is re-initialized by another call to *pthread\_rwlock\_init()*. An  
 34802 implementation may cause *pthread\_rwlock\_destroy()* to set the object referenced by *rwlock* to an  
 34803 invalid value. Results are undefined if *pthread\_rwlock\_destroy()* is called when any thread holds  
 34804 *rwlock*. Attempting to destroy an uninitialized read-write lock results in undefined behavior.

34805 The *pthread\_rwlock\_init()* function shall allocate any resources required to use the read-write  
 34806 lock referenced by *rwlock* and initializes the lock to an unlocked state with attributes referenced  
 34807 by *attr*. If *attr* is NULL, the default read-write lock attributes are used; the effect is the same as  
 34808 passing the address of a default read-write lock attributes object. Once initialized, the lock can be  
 34809 used any number of times without being re-initialized. Results are undefined if  
 34810 *pthread\_rwlock\_init()* is called specifying an already initialized read-write lock. Results are  
 34811 undefined if a read-write lock is used without first being initialized.

34812 If the *pthread\_rwlock\_init()* function fails, *rwlock* is not initialized and the contents of *rwlock* are  
 34813 undefined.

34814 Only the object referenced by *rwlock* may be used for performing synchronization. The result of  
 34815 referring to copies of that object in calls to *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_rdlock()*,  
 34816 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*,  
 34817 *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*, or *pthread\_rwlock\_wrlock()* is undefined.

## 34818 RETURN VALUE

34819 If successful, the *pthread\_rwlock\_destroy()* and *pthread\_rwlock\_init()* functions shall return zero;  
 34820 otherwise, an error number shall be returned to indicate the error.

34821 The [EBUSY] and [EINVAL] error checks, if implemented, act as if they were performed  
 34822 immediately at the beginning of processing for the function and caused an error return prior to  
 34823 modifying the state of the read-write lock specified by *rwlock*.

## 34824 ERRORS

34825 The *pthread\_rwlock\_destroy()* function may fail if:

34826 [EBUSY] The implementation has detected an attempt to destroy the object referenced  
 34827 by *rwlock* while it is locked.

34828 [EINVAL] The value specified by *rwlock* is invalid.

34829 The *pthread\_rwlock\_init()* function shall fail if:

34830 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
 34831 another read-write lock.

34832 [ENOMEM] Insufficient memory exists to initialize the read-write lock.

34833 MAN [EPERM] The caller does not have the privilege to perform the operation.

- 34834 The `pthread_rwlock_init()` function may fail if:
- 34835 [EBUSY] The implementation has detected an attempt to re-initialize the object  
34836 referenced by `rwlock`, a previously initialized but not yet destroyed read-write  
34837 lock.
- 34838 [EINVAL] The value specified by `attr` is invalid.

### 34839 **Notes to Reviewers**

- 34840 *This section with side shading will not appear in the final copy. - Ed.*
- 34841 D1, XSH, ERN 259 noted that no error return is described for when the argument `rwlock` is  
34842 invalid and proposes adding as a may fail case:
- 34843 [EINVAL] The value specified by `rwlock` is invalid.
- 34844 These functions shall not return an error code of [EINTR].

### 34845 **EXAMPLES**

- 34846 None.

### 34847 **APPLICATION USAGE**

- 34848 None.

### 34849 **RATIONALE**

- 34850 None.

### 34851 **FUTURE DIRECTIONS**

- 34852 None.

### 34853 **SEE ALSO**

- 34854 `pthread_rwlock_rdlock()`, `pthread_rwlock_timedrdlock()`, `pthread_rwlock_timedwrlock()`,  
34855 `pthread_rwlock_tryrdlock()`, `pthread_rwlock_trywrlock()`, `pthread_rwlock_unlock()`,  
34856 `pthread_rwlock_wrlock()`, the Base Definitions volume of IEEE Std. 1003.1-200x, <**pthread.h**>

### 34857 **CHANGE HISTORY**

- 34858 First released in Issue 5.

### 34859 **Issue 6**

- 34860 The following changes are made for alignment with IEEE Std. 1003.1j-2000:
- 34861 • The margin code in the SYNOPSIS s changed to RWL and the initializer macro is deleted.
- 34862 • The DESCRIPTION is updated as follows:
- 34863 — It explicitly notes allocation of resources upon initialization of a read-write lock object.
- 34864 — A paragraph is added specifying that copies of read-write lock objects may not be used.
- 34865 • An [EINVAL] error is added to the ERRORS section for `pthread_rwlock_init()`, indicating that  
34866 the `rwlock` value is invalid.
- 34867 • The SEE ALSO section is updated.
- 34868 The **restrict** keyword is added to the `pthread_rwlock_init()` prototype for alignment with the  
34869 ISO/IEC 9899:1999 standard.

34870 **NAME**

34871 pthread\_rwlock\_init — initialize a read-write lock object

34872 **SYNOPSIS**

34873 THR #include <pthread.h>

```
34874 int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
34875 const pthread_rwlockattr_t *restrict attr);
34876
```

34877 **DESCRIPTION**

34878 Refer to *pthread\_rwlock\_destroy()*.



34879 **NAME**

34880 pthread\_rwlock\_rdlock, pthread\_rwlock\_tryrdlock — lock a read-write lock object for reading

34881 **SYNOPSIS**

34882 THR #include &lt;pthread.h&gt;

34883 int pthread\_rwlock\_rdlock(pthread\_rwlock\_t \*rwlock);

34884 int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock);

34885

34886 **DESCRIPTION**

34887 The *pthread\_rwlock\_rdlock()* function shall apply a read lock to the read-write lock referenced by  
 34888 *rwlock*. The calling thread acquires the read lock if a writer does not hold the lock and there are  
 34889 TPS no writers blocked on the lock. If the Thread Execution Scheduling option is supported, and the  
 34890 threads involved in the lock are executing with the scheduling policies SCHED\_FIFO,  
 34891 SCHED\_RR, or SCHED\_SPORADIC, the calling thread shall not acquire the lock if a writer  
 34892 holds the lock or if writers of higher or equal priority are blocked on the lock; otherwise, the  
 34893 calling thread shall acquire the lock. If the Thread Execution Scheduling option is not supported,  
 34894 it is implementation-defined whether the calling thread acquires the lock when a writer does not  
 34895 hold the lock and there are writers blocked on the lock. If a writer holds the lock, the calling  
 34896 thread shall not acquire the read lock. If the read lock is not acquired, the calling thread blocks  
 34897 (that is, it does not return from the *pthread\_rwlock\_rdlock()* call) until it can acquire the lock. The  
 34898 calling thread may deadlock if at the time the call is made it holds a write lock.

34899 A thread may hold multiple concurrent read locks on *rwlock* (that is, successfully call the  
 34900 *pthread\_rwlock\_rdlock()* function *n* times). If so, the application shall ensure that the thread  
 34901 performs matching unlocks (that is, it calls the *pthread\_rwlock\_unlock()* function *n* times).

34902 The maximum number of simultaneous read locks that an implementation guarantees can be  
 34903 applied to a read-write lock shall be implementation-defined. The *pthread\_rwlock\_rdlock()*  
 34904 function may fail if this maximum would be exceeded.

34905 The *pthread\_rwlock\_tryrdlock()* function shall apply a read lock as in the *pthread\_rwlock\_rdlock()*  
 34906 function, with the exception that the function shall fail if the equivalent *pthread\_rwlock\_rdlock()*  
 34907 call would have blocked the calling thread. In no case does the *pthread\_rwlock\_tryrdlock()*  
 34908 function ever block; it always either acquires the lock or fails and returns immediately.

34909 Results are undefined if any of these functions are called with an uninitialized read-write lock.

34910 If a signal is delivered to a thread waiting for a read-write lock for reading, upon return from the  
 34911 signal handler the thread resumes waiting for the read-write lock for reading as if it was not  
 34912 interrupted.

34913 **RETURN VALUE**

34914 If successful, the *pthread\_rwlock\_rdlock()* function shall return zero; otherwise, an error number  
 34915 shall be returned to indicate the error.

34916 The *pthread\_rwlock\_tryrdlock()* function shall return zero if the lock for reading on the read-write  
 34917 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to  
 34918 indicate the error.

34919 **ERRORS**

34920 The *pthread\_rwlock\_tryrdlock()* function shall fail if:

34921 [EBUSY] The read-write lock could not be acquired for reading because a writer holds  
 34922 the lock or a writer with the appropriate priority was blocked on it.

34923 The *pthread\_rwlock\_rdlock()* and *pthread\_rwlock\_tryrdlock()* functions may fail if:

34924 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
34925 object.

34926 [EAGAIN] The read lock could not be acquired because the maximum number of read  
34927 locks for *rwlock* has been exceeded.

34928 The *pthread\_rwlock\_rdlock()* function may fail if:

34929 [EDEADLK] The current thread already owns the read-write lock for writing.

34930 These functions shall not return an error code of [EINTR].

#### 34931 EXAMPLES

34932 None.

#### 34933 APPLICATION USAGE

34934 Applications using these functions may be subject to priority inversion, as discussed in the Base  
34935 Definitions volume of IEEE Std. 1003.1-200x, Section 3.287, Priority Inversion.

#### 34936 RATIONALE

34937 None.

#### 34938 FUTURE DIRECTIONS

34939 None.

#### 34940 SEE ALSO

34941 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_init()*, *pthread\_rwlock\_timedrdlock()*,  
34942 *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*,  
34943 *pthread\_rwlock\_wrlock()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

#### 34944 CHANGE HISTORY

34945 First released in Issue 5.

#### 34946 Issue 6

34947 The following changes are made for alignment with IEEE Std. 1003.1j-2000:

- 34948
- The margin code in the SYNOPSIS is changed to RWL.
  - 34949 • The DESCRIPTION is updated as follows:
    - 34950 — Conditions under which writers have precedence over readers are specified.
    - 34951 — Failure of *pthread\_rwlock\_tryrdlock()* is clarified.
    - 34952 — A paragraph on the maximum number of read locks is added.
  - 34953 • In the ERRORS sections, [EBUSY] is modified to take into account write priority, and  
34954 [EDEADLK] is deleted as a *pthread\_rwlock\_tryrdlock()* error.
  - 34955 • The SEE ALSO section is updated.

34956 **NAME**

34957 pthread\_rwlock\_timedrdlock — lock a read-write lock for reading

34958 **SYNOPSIS**

34959 THR TMO #include &lt;pthread.h&gt;

34960 #include &lt;time.h&gt;

34961 int pthread\_rwlock\_timedrdlock(pthread\_rwlock\_t \*restrict *rwlock*,34962 const struct timespec \*restrict *abs\_timeout*);

34963

34964 **DESCRIPTION**

34965 The *pthread\_rwlock\_timedrdlock()* function applies a read lock to the read-write lock referenced  
 34966 by *rwlock* as in the *pthread\_rwlock\_rdlock()* function. However, if the lock cannot be acquired  
 34967 without waiting for other threads to unlock the lock, this wait shall be terminated when the  
 34968 specified timeout expires. The timeout expires when the absolute time specified by *abs\_timeout*  
 34969 passes, as measured by the clock on which timeouts are based (that is, when the value of that  
 34970 clock equals or exceeds *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already  
 34971 been passed at the time of the call.

34972 TMR If the Timers option is supported, the timeout is based on the `CLOCK_REALTIME` clock; if the  
 34973 Timers option is not supported, the timeout is based on the system clock as returned by the  
 34974 *time()* function. The resolution of the timeout is the resolution of the clock on which it is based.  
 34975 The `timespec` data type is defined as a structure in the `<time.h>` header. Under no circumstances  
 34976 shall the function fail with a timeout if the lock can be acquired immediately. The validity of the  
 34977 *abs\_timeout* parameter need not be checked if the lock can be immediately acquired.

34978 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-  
 34979 write lock via a call to *pthread\_rwlock\_timedrdlock()*, upon return from the signal handler the  
 34980 thread shall resume waiting for the lock as if it was not interrupted.

34981 The calling thread may deadlock if at the time the call is made it holds a write lock on *rwlock*.  
 34982 The results are undefined if this function is called with an uninitialized read-write lock.

34983 **RETURN VALUE**

34984 The *pthread\_rwlock\_timedrdlock()* function shall return zero if the lock for reading on the read-  
 34985 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned  
 34986 to indicate the error.

34987 **ERRORS**34988 The *pthread\_rwlock\_timedrdlock()* function shall fail if:

34989 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

34990 The *pthread\_rwlock\_timedrdlock()* function may fail if:34991 [EAGAIN] The read lock could not be acquired because the maximum number of read  
34992 locks for lock would be exceeded.34993 [EDEADLK] The calling thread already holds a write lock on *rwlock*.34994 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
34995 object, or the *abs\_timeout* nanosecond value is less than zero or greater than or  
34996 equal to 1 000 million.

34997 This function shall not return an error code of [EINTR].

## 34998 EXAMPLES

34999 None.

## 35000 APPLICATION USAGE

35001 Applications using this function may be subject to priority inversion, as discussed in the Base  
35002 Definitions volume of IEEE Std. 1003.1-200x, Section 3.287, Priority Inversion.

35003 The *pthread\_rwlock\_timedrdlock()* function is part of the Threads and Timeouts options and need  
35004 not be provided on all implementations.

## 35005 RATIONALE

35006 None.

## 35007 FUTURE DIRECTIONS

35008 None.

## 35009 SEE ALSO

35010 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_init()*, *pthread\_rwlock\_rdlock()*,  
35011 *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*, *pthread\_rwlock\_trywrlock()*,  
35012 *pthread\_rwlock\_unlock()*, *pthread\_rwlock\_wrlock()*, the Base Definitions volume of  
35013 IEEE Std. 1003.1-200x, <pthread.h>, <time.h>

## 35014 CHANGE HISTORY

35015 First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.

35016 **NAME**

35017 pthread\_rwlock\_timedwrlock — lock a read-write lock for writing

35018 **SYNOPSIS**

35019 THR TMO #include &lt;pthread.h&gt;

35020 #include &lt;time.h&gt;

35021 int pthread\_rwlock\_timedwrlock(pthread\_rwlock\_t \*restrict *rwlock*,35022 const struct timespec \*restrict *abs\_timeout*);

35023

35024 **DESCRIPTION**

35025 The *pthread\_rwlock\_timedwrlock()* function applies a write lock to the read-write lock referenced  
 35026 by *rwlock* as in the *pthread\_rwlock\_wrlock()* function. However, if the lock cannot be acquired  
 35027 without waiting for other threads to unlock the lock, this wait shall be terminated when the  
 35028 specified timeout expires. The timeout expires when the absolute time specified by *abs\_timeout*  
 35029 passes, as measured by the clock on which timeouts are based (that is, when the value of that  
 35030 clock equals or exceeds *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already  
 35031 been passed at the time of the call.

35032 TMR If the Timers option is supported, the timeout is based on the `CLOCK_REALTIME` clock; if the  
 35033 Timers option is not supported, the timeout is based on the system clock as returned by the  
 35034 *time()* function. The resolution of the timeout is the resolution of the clock on which it is based.  
 35035 The `timespec` data type is defined as a structure in the `<time.h>` header. Under no circumstances  
 35036 shall the function fail with a timeout if the lock can be acquired immediately. The validity of the  
 35037 *abs\_timeout* parameter need not be checked if the lock can be immediately acquired.

35038 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-  
 35039 write lock via a call to *pthread\_rwlock\_timedwrlock()*, upon return from the signal handler the  
 35040 thread shall resume waiting for the lock as if it was not interrupted.

35041 The calling thread may deadlock if at the time the call is made it holds the read-write lock. The  
 35042 results are undefined if this function is called with an uninitialized read-write lock.

35043 **RETURN VALUE**

35044 The *pthread\_rwlock\_timedwrlock()* function shall return zero if the lock for writing on the read-  
 35045 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned  
 35046 to indicate the error.

35047 **ERRORS**35048 The *pthread\_rwlock\_timedwrlock()* function shall fail if:

35049 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

35050 The *pthread\_rwlock\_timedwrlock()* function may fail if:35051 [EDEADLK] The calling thread already holds the *rwlock*.

35052 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
 35053 object, or the *abs\_timeout* nanosecond value is less than zero or greater than or  
 35054 equal to 1 000 million.

35055 This function shall not return an error code of [EINTR].

35056 **EXAMPLES**

35057           None.

35058 **APPLICATION USAGE**

35059           Applications using this function may be subject to priority inversion, as discussed in the Base  
35060           Definitions volume of IEEE Std. 1003.1-200x, Section 3.287, Priority Inversion.

35061           The *pthread\_rwlock\_timedwrlock()* function is part of the Threads and Timeouts options and need  
35062           not be provided on all implementations.

35063 **RATIONALE**

35064           None.

35065 **FUTURE DIRECTIONS**

35066           None.

35067 **SEE ALSO**

35068           *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_init()*, *pthread\_rwlock\_rdlock()*,  
35069           *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_tryrdlock()*, *pthread\_rwlock\_trywrlock()*,  
35070           *pthread\_rwlock\_unlock()*, *pthread\_rwlock\_wrlock()*, the Base Definitions volume of  
35071           IEEE Std. 1003.1-200x, <pthread.h>, <time.h>

35072 **CHANGE HISTORY**

35073           First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.

35074 **NAME**

35075 pthread\_rwlock\_tryrdlock — lock a read-write lock object for reading

35076 **SYNOPSIS**

35077 THR #include &lt;pthread.h&gt;

35078 int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock);

35079

35080 **DESCRIPTION**35081 Refer to *pthread\_rwlock\_rdlock()*.

## 35082 NAME

35083 pthread\_rwlock\_trywrlock, pthread\_rwlock\_wrlock — lock a read-write lock object for writing

## 35084 SYNOPSIS

```
35085 THR #include <pthread.h>
```

```
35086 int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);
```

```
35087 int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);
```

35088

## 35089 DESCRIPTION

35090 The *pthread\_rwlock\_trywrlock()* function shall apply a write lock like the *pthread\_rwlock\_wrlock()*  
35091 function, with the exception that the function shall fail if any thread currently holds *rwlock* (for  
35092 reading or writing).

35093 The *pthread\_rwlock\_wrlock()* function shall apply a write lock to the read-write lock referenced  
35094 by *rwlock*. The calling thread acquires the write lock if no other thread (reader or writer) holds  
35095 the read-write lock *rwlock*. Otherwise, the thread blocks (that is, does not return from the  
35096 *pthread\_rwlock\_wrlock()* call) until it can acquire the lock. The calling thread may deadlock if at  
35097 the time the call is made it holds the read-write lock (whether a read or write lock).

35098 Implementations are allowed to favor writers over readers to avoid writer starvation.

35099 Results are undefined if any of these functions are called with an uninitialized read-write lock.

35100 If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the  
35101 signal handler the thread resumes waiting for the read-write lock for writing as if it was not  
35102 interrupted.

## 35103 RETURN VALUE

35104 The *pthread\_rwlock\_trywrlock()* function shall return zero if the lock for writing on the read-write  
35105 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to  
35106 indicate the error.

35107 If successful, the *pthread\_rwlock\_wrlock()* function shall return zero; otherwise, an error number  
35108 shall be returned to indicate the error.

## 35109 ERRORS

35110 The *pthread\_rwlock\_trywrlock()* function shall fail if:

35111 [EBUSY] The read-write lock could not be acquired for writing because it was already  
35112 locked for reading or writing.

35113 The *pthread\_rwlock\_trywrlock()* and *pthread\_rwlock\_wrlock()* functions may fail if:

35114 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
35115 object.

35116 The *pthread\_rwlock\_wrlock()* function may fail if:

35117 [EDEADLK] The current thread already owns the read-write lock for writing or reading.

35118 These functions shall not return an error code of [EINTR].



35119 **EXAMPLES**

35120 None.

35121 **APPLICATION USAGE**

35122 Applications using these functions may be subject to priority inversion, as discussed in the Base  
35123 Definitions volume of IEEE Std. 1003.1-200x, Section 3.287, Priority Inversion.

35124 **RATIONALE**

35125 None.

35126 **FUTURE DIRECTIONS**

35127 None.

35128 **SEE ALSO**

35129 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_init()*, *pthread\_rwlock\_rdlock()*,  
35130 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*,  
35131 *pthread\_rwlock\_unlock()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**pthread.h**>

35132 **CHANGE HISTORY**

35133 First released in Issue 5.

35134 **Issue 6**

35135 The following changes are made for alignment with IEEE Std. 1003.1j-2000:

- 35136
- The margin code in the SYNOPSIS is changed to RWL.
  - The [EDEADLK] error is deleted as a *pthread\_rwlock\_trywrlock()* error.
  - The SEE ALSO section is updated.
- 35137
- 35138

35139 **NAME**

35140 pthread\_rwlock\_unlock — unlock a read-write lock object

35141 **SYNOPSIS**

35142 THR #include &lt;pthread.h&gt;

35143 int pthread\_rwlock\_unlock(pthread\_rwlock\_t \*rwlock);

35144

35145 **DESCRIPTION**

35146 The *pthread\_rwlock\_unlock()* function is called to release a lock held on the read-write lock object  
35147 referenced by *rwlock*. Results are undefined if the read-write lock *rwlock* is not held by the  
35148 calling thread.

35149 If this function is called to release a read lock from the read-write lock object and there are other  
35150 read locks currently held on this read-write lock object, the read-write lock object remains in the  
35151 read locked state. If this function releases the last read lock for this read-write lock object, the  
35152 read-write lock object shall be put in the unlocked state with no owners.

35153 If this function is called to release a write lock for this read-write lock object, the read-write lock  
35154 object shall be put in the unlocked state.

35155 If there are threads blocked on the lock when it becomes available, the scheduling policy is used  
35156 to determine which thread(s) shall acquire the lock. If the Thread Execution Scheduling option is  
35157 supported, when threads executing with the scheduling policies SCHED\_FIFO, SCHED\_RR, or  
35158 SCHED\_SPORADIC are waiting on the lock, they will acquire the lock in priority order when  
35159 the lock becomes available. For equal priority threads, write locks take precedence over read  
35160 locks. If the Thread Execution Scheduling option is not supported, it is implementation-defined  
35161 whether write locks take precedence over read locks.

35162 Results are undefined if any of these functions are called with an uninitialized read-write lock.

35163 **RETURN VALUE**

35164 If successful, the *pthread\_rwlock\_unlock()* function shall return zero; otherwise, an error number  
35165 shall be returned to indicate the error.

35166 **ERRORS**

35167 The *pthread\_rwlock\_unlock()* function may fail if:

35168 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
35169 object.

35170 [EPERM] The current thread does not hold a lock on the read-write lock.

35171 The *pthread\_rwlock\_unlock()* function shall not return an error code of [EINTR].

35172 **EXAMPLES**

35173 None.

35174 **APPLICATION USAGE**

35175 None.

35176 **RATIONALE**

35177 None.

35178 **FUTURE DIRECTIONS**

35179 None.

35180 **SEE ALSO**

35181 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_init()*, *pthread\_rwlock\_rdlock()*,  
35182 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*,  
35183 *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_wrlock()*, the Base Definitions volume of  
35184 IEEE Std. 1003.1-200x, <pthread.h>

35185 **CHANGE HISTORY**

35186 First released in Issue 5.

35187 **Issue 6**

35188 The following changes are made for alignment with IEEE Std. 1003.1j-2000:

- 35189 • The margin code in the SYNOPSIS is changed to RWL.
- 35190 • The DESCRIPTION is updated as follows:
  - 35191 — The conditions under which writers have precedence over readers are specified.
  - 35192 — The concept of read-write lock owner is deleted.
- 35193 • The SEE ALSO section is updated.

35194 **NAME**

35195 pthread\_rwlock\_wrlock — lock a read-write lock object for writing

35196 **SYNOPSIS**

35197 THR #include <pthread.h>

35198 int pthread\_rwlock\_wrlock(pthread\_rwlock\_t \**rwlock*);

35199

35200 **DESCRIPTION**

35201 Refer to *pthread\_rwlock\_trywrlock()*.

35202 **NAME**

35203 pthread\_rwlockattr\_destroy, pthread\_rwlockattr\_init — destroy and initialize read-write lock  
 35204 attributes object

35205 **SYNOPSIS**

```
35206 THR #include <pthread.h>
35207
35207 int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
35208 int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
35209
```

35210 **DESCRIPTION**

35211 The *pthread\_rwlockattr\_destroy()* function shall destroy a read-write lock attributes object. The  
 35212 effect of subsequent use of the object is undefined until the object is re-initialized by another call  
 35213 to *pthread\_rwlockattr\_init()*. An implementation may cause *pthread\_rwlockattr\_destroy()* to set  
 35214 the object referenced by *attr* to an invalid value.

35215 The *pthread\_rwlockattr\_init()* function shall initialize a read-write lock attributes object *attr* with  
 35216 the default value for all of the attributes defined by the implementation.

35217 Results are undefined if *pthread\_rwlockattr\_init()* is called specifying an already initialized read-  
 35218 write lock attributes object.

35219 After a read-write lock attributes object has been used to initialize one or more read-write locks,  
 35220 any function affecting the attributes object (including destruction) does not affect any previously  
 35221 initialized read-write locks.

35222 **RETURN VALUE**

35223 If successful, the *pthread\_rwlockattr\_destroy()* and *pthread\_rwlockattr\_init()* functions shall return  
 35224 zero; otherwise, an error number shall be returned to indicate the error.

35225 **ERRORS**

35226 The *pthread\_rwlockattr\_destroy()* function may fail if:

35227 [EINVAL] The value specified by *attr* is invalid.

35228 The *pthread\_rwlockattr\_init()* function shall fail if:

35229 [ENOMEM] Insufficient memory exists to initialize the read-write lock attributes object.

35230 These functions shall not return an error code of [EINTR].

35231 **EXAMPLES**

35232 None.

35233 **APPLICATION USAGE**

35234 None.

35235 **RATIONALE**

35236 None.

35237 **FUTURE DIRECTIONS**

35238 None.

35239 **SEE ALSO**

35240 *pthread\_rwlock\_init()*, *pthread\_rwlockattr\_getpshared()*, *pthread\_rwlockattr\_setpshared()*, the Base  
 35241 Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

35242 **CHANGE HISTORY**

35243 First released in Issue 5.

35244 **Issue 6**

35245 The following changes are made for alignment with IEEE Std. 1003.1j-2000:

- 35246 • The margin code in the SYNOPSIS is changed to RWL.
- 35247 • The SEE ALSO section is updated.

35248 **NAME**

35249 pthread\_rwlockattr\_getpshared, pthread\_rwlockattr\_setpshared — get and set process-shared  
 35250 attribute of read-write lock attributes object

35251 **SYNOPSIS**

```
35252 THR TSH #include <pthread.h>
35253
35253 int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t *restrict attr,
35254 int *restrict pshared);
35255 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
35256 int pshared);
35257
```

35258 **DESCRIPTION**

35259 The *process-shared* attribute is set to PTHREAD\_PROCESS\_SHARED to permit a read-write lock  
 35260 to be operated upon by any thread that has access to the memory where the read-write lock is  
 35261 allocated, even if the read-write lock is allocated in memory that is shared by multiple processes.  
 35262 If the *process-shared* attribute is PTHREAD\_PROCESS\_PRIVATE, the read-write lock shall only  
 35263 be operated upon by threads created within the same process as the thread that initialized the  
 35264 read-write lock; if threads of differing processes attempt to operate on such a read-write lock,  
 35265 the behavior is undefined. The default value of the *process-shared* attribute is  
 35266 PTHREAD\_PROCESS\_PRIVATE.

35267 The *pthread\_rwlockattr\_getpshared()* function obtains the value of the *process-shared* attribute from  
 35268 the initialized attributes object referenced by *attr*. The *pthread\_rwlockattr\_setpshared()* function is  
 35269 used to set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

35270 Additional attributes, their default values, and the names of the associated functions to get and  
 35271 set those attribute values are implementation-defined.

35272 **RETURN VALUE**

35273 Upon successful completion, the *pthread\_rwlockattr\_getpshared()* shall return zero and store the  
 35274 value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter.  
 35275 Otherwise, an error number shall be returned to indicate the error.

35276 If successful, the *pthread\_rwlockattr\_setpshared()* function shall return zero; otherwise, an error  
 35277 number shall be returned to indicate the error.

35278 **ERRORS**

35279 The *pthread\_rwlockattr\_getpshared()* and *pthread\_rwlockattr\_setpshared()* functions may fail if:

35280 [EINVAL] The value specified by *attr* is invalid.

35281 The *pthread\_rwlockattr\_setpshared()* function may fail if:

35282 [EINVAL] The new value specified for the attribute is outside the range of legal values  
 35283 for that attribute.

35284 These functions shall not return an error code of [EINTR].

35285 **EXAMPLES**

35286 None.

35287 **APPLICATION USAGE**

35288 None.

35289 **RATIONALE**

35290 None.

35291 **FUTURE DIRECTIONS**

35292 None.

35293 **SEE ALSO**

35294 *pthread\_rwlock\_init()*, *pthread\_rwlockattr\_destroy()*, *pthread\_rwlockattr\_init()*, the Base Definitions  
35295 volume of IEEE Std. 1003.1-200x, <pthread.h>

35296 **CHANGE HISTORY**

35297 First released in Issue 5.

35298 **Issue 6**

35299 The following changes are made for alignment with IEEE Std. 1003.1j-2000:

- 35300 • The margin code in the SYNOPSIS is changed to RWL TSH.
- 35301 • The DESCRIPTION notes that additional attributes are implementation-defined.
- 35302 • The SEE ALSO section is updated.

35303 The **restrict** keyword is added to the *pthread\_rwlockattr\_getpshared()* prototype for alignment  
35304 with the ISO/IEC 9899:1999 standard.



35305 **NAME**

35306 pthread\_rwlockattr\_init — initialize read-write lock attributes object

35307 **SYNOPSIS**

35308 XSI #include &lt;pthread.h&gt;

35309 int pthread\_rwlockattr\_init(pthread\_rwlockattr\_t \*attr);

35310

35311 **DESCRIPTION**35312 Refer to *pthread\_rwlockattr\_destroy()*.

35313 **NAME**

35314 pthread\_rwlockattr\_setpshared — set process-shared attribute of read-write lock attributes  
35315 object

35316 **SYNOPSIS**

35317 XSI #include <pthread.h>

```
35318 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
35319 int pshared);
```

35320

35321 **DESCRIPTION**

35322 Refer to *pthread\_rwlockattr\_getpshared()*.

35323 **NAME**

35324 pthread\_self — get calling thread's ID

35325 **SYNOPSIS**

35326 THR #include &lt;pthread.h&gt;

35327 pthread\_t pthread\_self(void);

35328

35329 **DESCRIPTION**35330 The *pthread\_self()* function shall return the thread ID of the calling thread.35331 **RETURN VALUE**

35332 Refer to the DESCRIPTION.

35333 **ERRORS**

35334 No errors are defined.

35335 The *pthread\_self()* function shall not return an error code of [EINTR].35336 **EXAMPLES**

35337 None.

35338 **APPLICATION USAGE**

35339 None.

35340 **RATIONALE**35341 The *pthread\_self()* function provides a capability similar to the *getpid()* function for processes  
35342 and the rationale is the same: the creation call does not provide the thread ID to the created  
35343 thread.35344 **FUTURE DIRECTIONS**

35345 None.

35346 **SEE ALSO**35347 *pthread\_create()*, *pthread\_equal()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
35348 <pthread.h>35349 **CHANGE HISTORY**

35350 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

35351 **Issue 6**35352 The *pthread\_self()* function is marked as part of the Threads option.

## 35353 NAME

35354 pthread\_setcancelstate, pthread\_setcanceltype, pthread\_testcancel — set cancelability state

## 35355 SYNOPSIS

```
35356 THR #include <pthread.h>
```

```
35357 int pthread_setcancelstate(int state, int *oldstate);
```

```
35358 int pthread_setcanceltype(int type, int *oldtype);
```

```
35359 void pthread_testcancel(void);
```

35360

## 35361 DESCRIPTION

35362 The *pthread\_setcancelstate()* function shall atomically both set the calling thread's cancelability  
35363 state to the indicated *state* and return the previous cancelability state at the location referenced  
35364 by *oldstate*. Legal values for *state* are PTHREAD\_CANCEL\_ENABLE and  
35365 PTHREAD\_CANCEL\_DISABLE.

35366 The *pthread\_setcanceltype()* function shall atomically both set the calling thread's cancelability  
35367 type to the indicated *type* and return the previous cancelability type at the location referenced by  
35368 *oldtype*. Legal values for *type* are PTHREAD\_CANCEL\_DEFERRED and  
35369 PTHREAD\_CANCEL\_ASYNCHRONOUS.

35370 The cancelability state and type of any newly created threads, including the thread in which  
35371 *main()* was first invoked, shall be PTHREAD\_CANCEL\_ENABLE and  
35372 PTHREAD\_CANCEL\_DEFERRED respectively.

35373 The *pthread\_testcancel()* function shall create a cancellation point in the calling thread. The  
35374 *pthread\_testcancel()* function shall have no effect if cancelability is disabled.

## 35375 RETURN VALUE

35376 If successful, the *pthread\_setcancelstate()* and *pthread\_setcanceltype()* functions shall return zero;  
35377 otherwise, an error number shall be returned to indicate the error.

## 35378 ERRORS

35379 The *pthread\_setcancelstate()* function may fail if:

35380 [EINVAL] The specified state is not PTHREAD\_CANCEL\_ENABLE or  
35381 PTHREAD\_CANCEL\_DISABLE.

35382 The *pthread\_setcanceltype()* function may fail if:

35383 [EINVAL] The specified type is not PTHREAD\_CANCEL\_DEFERRED or  
35384 PTHREAD\_CANCEL\_ASYNCHRONOUS.

35385 These functions shall not return an error code of [EINTR].

## 35386 EXAMPLES

35387 None.

## 35388 APPLICATION USAGE

35389 None.

## 35390 RATIONALE

35391 The *pthread\_setcancelstate()* and *pthread\_setcanceltype()* functions are used to control the points at  
35392 which a thread may be asynchronously canceled. For cancellation control to be usable in modular  
35393 fashion, some rules need to be followed.

35394 An object can be considered to be a generalization of a procedure. It is a set of procedures and  
35395 global variables written as a unit and called by clients not known by the object. Objects may  
35396 depend on other objects.

35397 First, cancelability should only be disabled on entry to an object, never explicitly enabled. On  
35398 exit from an object, the cancelability state should always be restored to its value on entry to the  
35399 object.

35400 This follows from a modularity argument: if the client of an object (or the client of an object that  
35401 uses that object) has disabled cancelability, it is because the client does not want to be concerned  
35402 about cleaning up if the thread is canceled while executing some sequence of actions. If an object  
35403 is called in such a state and it enables cancelability and a cancelation request is pending for that  
35404 thread, then the thread is canceled, contrary to the wish of the client that disabled.

35405 Second, the cancelability type may be explicitly set to either *deferred* or *asynchronous* upon entry  
35406 to an object. But as with the cancelability state, on exit from an object the cancelability type  
35407 should always be restored to its value on entry to the object.

35408 Finally, only functions that are cancel-safe may be called from a thread that is asynchronously  
35409 cancelable.

#### 35410 FUTURE DIRECTIONS

35411 None.

#### 35412 SEE ALSO

35413 *pthread\_cancel()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

#### 35414 CHANGE HISTORY

35415 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 35416 Issue 6

35417 The *pthread\_setcancelstate()*, *pthread\_setcanceltype()*, and *pthread\_testcancel()* functions are marked  
35418 as part of the Threads option.

35419 **NAME**

35420 pthread\_setconcurrency — set level of concurrency

35421 **SYNOPSIS**

35422 XSI #include <pthread.h>

35423 int pthread\_setconcurrency(int new\_level);

35424

35425 **DESCRIPTION**

35426 Refer to *pthread\_getconcurrency()*.

35427 **NAME**

35428 pthread\_setschedparam — dynamic thread scheduling parameters access (**REALTIME**  
35429 **THREADS**)

35430 **SYNOPSIS**

35431 TPS #include <pthread.h>

35432 int pthread\_setschedparam(pthread\_t *thread*, int *policy*,  
35433 const struct sched\_param \**param*);

35434

35435 **DESCRIPTION**

35436 Refer to *pthread\_getschedparam()*.

35437 **NAME**

35438 pthread\_setspecific — thread-specific data management

35439 **SYNOPSIS**

35440 THR #include <pthread.h>

35441 int pthread\_setspecific(pthread\_key\_t key, const void \*value);

35442

35443 **DESCRIPTION**

35444 Refer to *pthread\_getspecific()*.



35445 **NAME**

35446 pthread\_sigmask, sigprocmask — examine and change blocked signals

35447 **SYNOPSIS**

35448 #include &lt;signal.h&gt;

35449 THR int pthread\_sigmask(int how, const sigset\_t \*restrict set,  
35450 sigset\_t \*restrict oset);35451 int sigprocmask(int how, const sigset\_t \*restrict set,  
35452 sigset\_t \*restrict oset);35453 **DESCRIPTION**35454 THR The *pthread\_sigmask()* function is used to examine or change (or both) the calling thread's signal mask, regardless of the number of threads in the process. The effect shall be the same as described for *sigprocmask()*, without the restriction that the call be made in a single-threaded process.35458 In a single-threaded process, the *sigprocmask()* function allows the calling process to examine or change (or both) the signal mask of the calling thread.35460 If the argument *set* is not a null pointer, it points to a set of signals to be used to change the currently blocked set.35462 The argument *how* indicates the way in which the set is changed, and the application shall ensure it consists of one of the following values:35464 SIG\_BLOCK The resulting set shall be the union of the current set and the signal set pointed to by *set*.35466 SIG\_SETMASK The resulting set shall be the signal set pointed to by *set*.35467 SIG\_UNBLOCK The resulting set shall be the intersection of the current set and the complement of the signal set pointed to by *set*.35469 If the argument *oset* is not a null pointer, the previous mask is stored in the location pointed to by *oset*. If *set* is a null pointer, the value of the argument *how* is not significant and the process' signal mask is unchanged; thus the call can be used to enquire about currently blocked signals.35472 If there are any pending unblocked signals after the call to *sigprocmask()*, at least one of those signals shall be delivered before the call to *sigprocmask()* returns.

35474 It is not possible to block those signals which cannot be ignored. This shall be enforced by the system without causing an error to be indicated.

35476 If any of the SIGFPE, SIGILL, SIGSEGV, or SIGBUS signals are generated while they are blocked, the result is undefined, unless the signal was generated by the *kill()* function, the *sigqueue()* function, or the *raise()* function.35479 If *sigprocmask()* fails, the thread's signal mask is not changed.35480 The use of the *sigprocmask()* function is unspecified in a multi-threaded process.35481 **RETURN VALUE**35482 THR Upon successful completion *pthread\_sigmask()* shall return 0; otherwise, it shall return the corresponding error number.35484 Upon successful completion, *sigprocmask()* shall return 0; otherwise, -1 shall be returned, *errno* shall be set to indicate the error, and the process' signal mask shall be unchanged.

35486 **ERRORS**

35487 THR The *pthread\_sigmask()* and *sigprocmask()* functions shall fail if:

35488 [EINVAL] The value of the *how* argument is not equal to one of the defined values.

35489 THR The *pthread\_sigmask()* function shall not return an error code of [EINTR].

35490 **EXAMPLES**

35491 None.

35492 **APPLICATION USAGE**

35493 None.

35494 **RATIONALE**

35495 When a process' signal mask is changed in a signal-catching function that is installed by  
35496 *sigaction()*, the restoration of the signal mask on return from the signal-catching function  
35497 overrides that change (see *sigaction()*). If the signal-catching function was installed with  
35498 *signal()*, it is unspecified whether this occurs.

35499 See *kill()* for a discussion of the requirement on delivery of signals.

35500 **FUTURE DIRECTIONS**

35501 None.

35502 **SEE ALSO**

35503 *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *sigpending()*,  
35504 *sigqueue()*, *sigsuspend()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <signal.h>

35505 **CHANGE HISTORY**

35506 First released in Issue 3.

35507 Entry included for alignment with the POSIX.1-1988 standard.

35508 **Issue 4**

35509 The DESCRIPTION is changed to indicate that signals can also be generated by *raise()*.

35510 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 35511 • The type of the arguments *set* and *oset* are changed from **sigset\_t\*** to **const sigset\_t\***.

35512 **Issue 5**

35513 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

35514 The *pthread\_sigmask()* function is added for alignment with the POSIX Threads Extension.

35515 **Issue 6**

35516 The *pthread\_sigmask()* function is marked as part of the Threads option.

35517 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 35518 • The DESCRIPTION is updated to explicitly state the functions which may generate the  
35519 signal.

35520 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

35521 The **restrict** keyword is added to the *pthread\_sigmask()* and *sigprocmask()* prototypes for  
35522 alignment with the ISO/IEC 9899: 1999 standard.

35523 **NAME**

35524 pthread\_spin\_destroy, pthread\_spin\_init — destroy or initialize a spin lock object

35525 **SYNOPSIS**

35526 SPI #include &lt;pthread.h&gt;

35527 int pthread\_spin\_destroy(pthread\_spinlock\_t \*lock);

35528 int pthread\_spin\_init(pthread\_spinlock\_t \*lock, int pshared);

35529

35530 **DESCRIPTION**

35531 The *pthread\_spin\_destroy()* function destroys the spin lock referenced by *lock* and releases any  
 35532 resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is  
 35533 reinitialized by another call to *pthread\_spin\_init()*. The results are undefined if  
 35534 *pthread\_spin\_destroy()* is called when a thread holds the lock, or if this function is called with an  
 35535 uninitialized thread spin lock.

35536 The *pthread\_spin\_init()* function allocates any resources required to use the spin lock referenced  
 35537 by *lock* and initializes the lock to an unlocked state.

35538 TSH If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is  
 35539 PTHREAD\_PROCESS\_SHARED, the implementation shall permit the spin lock to be operated  
 35540 upon by any thread that has access to the memory where the spin lock is allocated, even if it is  
 35541 allocated in memory that is shared by multiple processes.

35542 If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is  
 35543 PTHREAD\_PROCESS\_PRIVATE, or if the option is not supported, the spin lock shall only be  
 35544 operated upon by threads created within the same process as the thread that initialized the spin  
 35545 lock. If threads of differing processes attempt to operate on such a spin lock, the behavior is  
 35546 undefined.

35547 The results are undefined if *pthread\_spin\_init()* is called specifying an already initialized spin  
 35548 lock. The results are undefined if a spin lock is used without first being initialized.

35549 If the *pthread\_spin\_init()* function fails, the lock is not initialized and the contents of *lock* are  
 35550 undefined.

35551 Only the object referenced by *lock* may be used for performing synchronization.

35552 The result of referring to copies of that object in calls to *pthread\_spin\_destroy()*,  
 35553 *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*, or *pthread\_spin\_unlock()* is undefined.

35554 **RETURN VALUE**

35555 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 35556 be returned to indicate the error.

35557 **ERRORS**

35558 These functions may fail if:

35559 [EBUSY] The implementation has detected an attempt to initialize or destroy a spin  
 35560 lock while it is in use (for example, while being used in a *pthread\_spin\_lock()*  
 35561 call) by another thread.

35562 [EINVAL] The value specified by *lock* is invalid.

35563 The *pthread\_spin\_init()* function shall fail if:

35564 [EAGAIN] The system lacks the necessary resources to initialize another spin lock.

35565 [ENOMEM] Insufficient memory exists to initialize the lock.

35566 These functions shall not return an error code of [EINTR].

35567 **EXAMPLES**

35568 None.

35569 **APPLICATION USAGE**

35570 The *pthread\_spin\_destroy()* and *pthread\_spin\_init()* functions are part of the Spin Locks option  
35571 and need not be provided on all implementations.

35572 **RATIONALE**

35573 None.

35574 **FUTURE DIRECTIONS**

35575 None.

35576 **SEE ALSO**

35577 *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*, *pthread\_spin\_unlock()*, the Base Definitions volume of  
35578 IEEE Std. 1003.1-200x, <<pthread.h>>

35579 **CHANGE HISTORY**

35580 First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.

35581 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

35582 **NAME**

35583 pthread\_spin\_init — initialize a spin lock object

35584 **SYNOPSIS**

35585 SPI #include &lt;pthread.h&gt;

35586 int pthread\_spin\_init(pthread\_spinlock\_t \*lock, int pshared);

35587

35588 **DESCRIPTION**35589 Refer to *pthread\_spin\_destroy()*.

35590 **NAME**

35591 pthread\_spin\_lock, pthread\_spin\_trylock — lock a spin lock object

35592 **SYNOPSIS**

35593 SPI #include &lt;pthread.h&gt;

35594 int pthread\_spin\_lock(pthread\_spinlock\_t \*lock);

35595 int pthread\_spin\_trylock(pthread\_spinlock\_t \*lock);

35596

35597 **DESCRIPTION**

35598 The *pthread\_spin\_lock()* function locks the spin lock referenced by *lock*. The calling thread  
35599 acquires the lock if it is not held by another thread. Otherwise, the thread spins (that is, does not  
35600 return from the *pthread\_spin\_lock()* call) until the lock becomes available. The results are  
35601 undefined if the calling thread holds the lock at the time the call is made. The  
35602 *pthread\_spin\_trylock()* function locks the spin lock referenced by *lock* if it is not held by any  
35603 thread. Otherwise, the function fails.

35604 The results are undefined if any of these functions is called with an uninitialized spin lock.

35605 **RETURN VALUE**

35606 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
35607 be returned to indicate the error.

35608 **ERRORS**

35609 These functions may fail if:

35610 [EINVAL] The value specified by *lock* does not refer to an initialized spin lock object.35611 The *pthread\_spin\_lock()* function may fail if:

35612 [EDEADLK] The calling thread already holds the lock.

35613 The *pthread\_spin\_trylock()* function shall fail if:

35614 [EBUSY] A thread currently holds the lock.

35615 These functions shall not return an error code of [EINTR].

35616 **EXAMPLES**

35617 None.

35618 **APPLICATION USAGE**

35619 Applications using this function may be subject to priority inversion, as discussed in the Base  
35620 Definitions volume of IEEE Std. 1003.1-200x, Section 3.287, Priority Inversion.

35621 The *pthread\_spin\_lock()* and *pthread\_spin\_trylock()* functions are part of the Spin Locks option  
35622 and need not be provided on all implementations.

35623 **RATIONALE**

35624 None.

35625 **FUTURE DIRECTIONS**

35626 None.

35627 **SEE ALSO**

35628 *pthread\_spin\_init()*, *pthread\_spin\_destroy()*, *pthread\_spin\_unlock()*, the Base Definitions volume of  
35629 IEEE Std. 1003.1-200x, <pthread.h>

35630 **CHANGE HISTORY**

35631 First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.

35632 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

35633 **NAME**

35634 pthread\_spin\_trylock — lock a spin lock object

35635 **SYNOPSIS**

35636 SPI #include <pthread.h>

35637 int pthread\_spin\_trylock(pthread\_spinlock\_t \*lock);

35638

35639 **DESCRIPTION**

35640 Refer to *pthread\_spin\_lock()*.



35641 **NAME**

35642 pthread\_spin\_unlock — unlock a spin lock object

35643 **SYNOPSIS**

35644 SPI #include &lt;pthread.h&gt;

35645 int pthread\_spin\_unlock(pthread\_spinlock\_t \*lock);

35646

35647 **DESCRIPTION**

35648 The *pthread\_spin\_unlock()* function releases the spin lock referenced by *lock* which was locked  
35649 via the *pthread\_spin\_lock()* or *pthread\_spin\_trylock()* functions. The results are undefined if the  
35650 lock is not held by the calling thread. If there are threads spinning on the lock when  
35651 *pthread\_spin\_unlock()* is called, the lock becomes available and an unspecified spinning thread  
35652 shall acquire the lock.

35653 The results are undefined if this function is called with an uninitialized thread spin lock.

35654 **RETURN VALUE**

35655 Upon successful completion, the *pthread\_spin\_unlock()* function shall return zero; otherwise, an  
35656 error number shall be returned to indicate the error.

35657 **ERRORS**35658 The *pthread\_spin\_unlock()* function may fail if:

35659 [EINVAL] An invalid argument was specified.

35660 [EPERM] The calling thread does not hold the lock.

35661 This function shall not return an error code of [EINTR].

35662 **EXAMPLES**

35663 None.

35664 **APPLICATION USAGE**

35665 The *pthread\_spin\_unlock()* function is part of the Spin Locks option and need not be provided on  
35666 all implementations.

35667 **RATIONALE**

35668 None.

35669 **FUTURE DIRECTIONS**

35670 None.

35671 **SEE ALSO**

35672 *pthread\_spin\_init()*, *pthread\_spin\_destroy()*, *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*, the Base  
35673 Definitions volume of IEEE Std. 1003.1-200x, <pthread.h>

35674 **CHANGE HISTORY**

35675 First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.

35676 In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

35677 **NAME**

35678 pthread\_testcancel — set cancelability state

35679 **SYNOPSIS**

35680 THR #include <pthread.h>

35681 void pthread\_testcancel(void);

35682

35683 **DESCRIPTION**

35684 Refer to *pthread\_setcancelstate()*.

35685 **NAME**

35686 ptsname — get name of the slave pseudo-terminal device

35687 **SYNOPSIS**

35688 XSI #include &lt;stdlib.h&gt;

35689 char \*ptsname(int *fildev*);

35690

35691 **DESCRIPTION**

35692 The *ptsname()* function shall return the name of the slave pseudo-terminal device associated  
 35693 with a master pseudo-terminal device. The *fildev* argument is a file descriptor that refers to the  
 35694 master device. The *ptsname()* function shall return a pointer to a string containing the path name  
 35695 of the corresponding slave device.

35696 The *ptsname()* function need not be reentrant. A function that is not required to be reentrant is  
 35697 not required to be thread-safe.

35698 **RETURN VALUE**

35699 Upon successful completion, *ptsname()* shall return a pointer to a string which is the name of the  
 35700 pseudo-terminal slave device. Upon failure, *ptsname()* shall return a null pointer. This could  
 35701 occur if *fildev* is an invalid file descriptor or if the slave device name does not exist in the file  
 35702 system.

35703 **ERRORS**

35704 No errors are defined.

35705 **EXAMPLES**

35706 None.

35707 **APPLICATION USAGE**35708 The value returned may point to a static data area that is overwritten by each call to *ptsname()*.35709 **RATIONALE**

35710 None.

35711 **FUTURE DIRECTIONS**

35712 None.

35713 **SEE ALSO**

35714 *grantpt()*, *open()*, *ttyname()*, *unlockpt()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
 35715 <stdlib.h>

35716 **CHANGE HISTORY**

35717 First released in Issue 4, Version 2.

35718 **Issue 5**

35719 Moved from X/OPEN UNIX extension to BASE.

35720 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

35721 **NAME**

35722       putc — put byte on a stream

35723 **SYNOPSIS**

35724       #include &lt;stdio.h&gt;

35725       int putc(int *c*, FILE \**stream*);35726 **DESCRIPTION**

35727 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
35728       conflict between the requirements described here and the ISO C standard is unintentional. This  
35729       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

35730       The *putc()* function shall be equivalent to *fputc()*, except that if it is implemented as a macro it  
35731       may evaluate *stream* more than once, so the argument should never be an expression with side  
35732       effects.

35733 **RETURN VALUE**35734       Refer to *fputc()*.35735 **ERRORS**35736       Refer to *fputc()*.35737 **EXAMPLES**

35738       None.

35739 **APPLICATION USAGE**

35740       Because it may be implemented as a macro, *putc()* may treat a *stream* argument with side effects  
35741       incorrectly. In particular, *putc(c,\*f++)* does not necessarily work correctly. Therefore, use of this  
35742       function is not recommended in such situations; *fputc()* should be used instead.

35743 **RATIONALE**

35744       None.

35745 **FUTURE DIRECTIONS**

35746       None.

35747 **SEE ALSO**35748       *fputc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>35749 **CHANGE HISTORY**

35750       First released in Issue 1. Derived from Issue 1 of the SVID.

35751 **Issue 4**

35752       The APPLICATION USAGE section now states that the use of this function is not recommended  
35753       with a *stream* argument with side effects.

35754       The following change is incorporated for alignment with the ISO C standard:

- 35755       • The *c* argument is not allowed to be evaluated more than once.

35756 **NAME**35757        `putc_unlocked` — stdio with explicit client locking35758 **SYNOPSIS**35759 TSF        `#include <stdio.h>`35760        `int putc_unlocked(int c, FILE *stream);`

35761

35762 **DESCRIPTION**35763        Refer to `getc_unlocked()`.

35764 **NAME**

35765 putchar — put byte on stdout stream

35766 **SYNOPSIS**

35767 #include <stdio.h>

35768 int putchar(int *c*);

35769 **DESCRIPTION**

35770 *cx* The functionality described on this reference page is aligned with the ISO C standard. Any  
35771 conflict between the requirements described here and the ISO C standard is unintentional. This  
35772 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

35773 The function call *putchar(c)* shall be equivalent to *putc(c,stdout)*.

35774 **RETURN VALUE**

35775 Refer to *fputc()*.

35776 **ERRORS**

35777 Refer to *fputc()*.

35778 **EXAMPLES**

35779 None.

35780 **APPLICATION USAGE**

35781 None.

35782 **RATIONALE**

35783 None.

35784 **FUTURE DIRECTIONS**

35785 None.

35786 **SEE ALSO**

35787 *putc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**stdio.h**>

35788 **CHANGE HISTORY**

35789 First released in Issue 1. Derived from Issue 1 of the SVID.

35790 **NAME**

35791 putchar\_unlocked — stdio with explicit client locking

35792 **SYNOPSIS**

35793 TSF #include &lt;stdio.h&gt;

35794 int putchar\_unlocked(int c);

35795

35796 **DESCRIPTION**35797 Refer to *getc\_unlocked()*.

35798 **NAME**

35799 putenv — change or add a value to environment

35800 **SYNOPSIS**

35801 xSI #include &lt;stdlib.h&gt;

35802 int putenv(char \*string);

35803

35804 **DESCRIPTION**

35805 The *putenv()* function uses the *string* argument to set environment variable values. The *string*  
35806 argument should point to a string of the form "*name=value*". The *putenv()* function makes the  
35807 value of the environment variable *name* equal to *value* by altering an existing variable or creating  
35808 a new one. In either case, the string pointed to by *string* becomes part of the environment, so  
35809 altering the string shall change the environment. The space used by *string* is no longer used once  
35810 a new string-defining *name* is passed to *putenv()*.

35811 The *putenv()* function need not be reentrant. A function that is not required to be reentrant is not  
35812 required to be thread-safe.

35813 **RETURN VALUE**

35814 Upon successful completion, *putenv()* shall return 0; otherwise, it shall return a non-zero value  
35815 and set *errno* to indicate the error.

35816 **ERRORS**35817 The *putenv()* function may fail if:

35818 [ENOMEM] Insufficient memory was available.

35819 **EXAMPLES**35820 **Changing the Value of an Environment Variable**

35821 The following example changes the value of the *HOME* environment variable to the value  
35822 */usr/home*.

35823 #include &lt;stdlib.h&gt;

35824 ...

35825 static char \*var = "HOME=/usr/home";

35826 int ret;

35827 ret = putenv(var);

35828 **APPLICATION USAGE**

35829 The *putenv()* function manipulates the environment pointed to by *environ*, and can be used in  
35830 conjunction with *getenv()*.

35831 This routine may use *malloc()* to enlarge the environment.

35832 A potential error is to call *putenv()* with an automatic variable as the argument, then return from  
35833 the calling function while *string* is still part of the environment.

35834 The *setenv()* function is preferred over this function.35835 **RATIONALE**

35836 None.



35837 **FUTURE DIRECTIONS**

35838 None.

35839 **SEE ALSO**35840 *exec*, *getenv()*, *malloc()*, *setenv()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h> |35841 **CHANGE HISTORY**

35842 First released in Issue 1. Derived from Issue 1 of the SVID. |

35843 **Issue 4**

35844 The &lt;stdlib.h&gt; header is added to the SYNOPSIS section.

35845 The type of argument *string* is changed from **char\*** to **const char\***.35846 **Issue 5**35847 The type of the argument to this function is changed from **const char\*** to **char\***. This was indicated as a FUTURE DIRECTION in previous issues.

35849 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

## 35850 NAME

35851 putmsg, putpmsg — send a message on a STREAM (STREAMS)

## 35852 SYNOPSIS

35853 XSR #include &lt;stropts.h&gt;

```

35854 int putmsg(int fildev, const struct strbuf *ctlptr,
35855 const struct strbuf *dataptr, int flags);
35856 int putpmsg(int fildev, const struct strbuf *ctlptr,
35857 const struct strbuf *dataptr, int band, int flags);
35858

```

## 35859 DESCRIPTION

35860 The *putmsg()* function shall create a message from a process buffer(s) and send the message to a  
 35861 STREAMS file. The message may contain either a data part, a control part, or both. The data and  
 35862 control parts are distinguished by placement in separate buffers, as described below. The  
 35863 semantics of each part are defined by the STREAMS module that receives the message.

35864 The *putpmsg()* function does the same thing as *putmsg()*, but the process can send messages in  
 35865 different priority bands. Except where noted, all requirements on *putmsg()* also pertain to  
 35866 *putpmsg()*.

35867 The *fildev* argument specifies a file descriptor referencing an open STREAM. The *ctlptr* and  
 35868 *dataptr* arguments each point to a **strbuf** structure.

35869 The *ctlptr* argument points to the structure describing the control part, if any, to be included in  
 35870 the message. The *buf* member in the **strbuf** structure points to the buffer where the control  
 35871 information resides, and the *len* member indicates the number of bytes to be sent. The *maxlen*  
 35872 member is not used by *putmsg()*. In a similar manner, the argument *dataptr* specifies the data, if  
 35873 any, to be included in the message. The *flags* argument indicates what type of message should be  
 35874 sent and is described further below.

35875 To send the data part of a message, the application shall ensure that *dataptr* is not a null pointer  
 35876 and the *len* member of *dataptr* is 0 or greater. To send the control part of a message, the  
 35877 application shall ensure that the corresponding values are set for *ctlptr*. No data (control) part  
 35878 shall be sent if either *dataptr(ctlptr)* is a null pointer or the *len* member of *dataptr(ctlptr)* is set to  
 35879 -1.

35880 For *putmsg()*, if a control part is specified and *flags* is set to RS\_HIPRI, a high priority message is  
 35881 sent. If no control part is specified, and *flags* is set to RS\_HIPRI, *putmsg()* fails and sets *errno* to  
 35882 [EINVAL]. If *flags* is set to 0, a normal message (priority band equal to 0) is sent. If a control part  
 35883 and data part are not specified and *flags* is set to 0, no message is sent and 0 is returned.

35884 For *putpmsg()*, the flags are different. The *flags* argument is a bitmask with the following  
 35885 mutually-exclusive flags defined: MSG\_HIPRI and MSG\_BAND. If *flags* is set to 0, *putpmsg()*  
 35886 fails and sets *errno* to [EINVAL]. If a control part is specified and *flags* is set to MSG\_HIPRI and  
 35887 *band* is set to 0, a high-priority message is sent. If *flags* is set to MSG\_HIPRI and either no control  
 35888 part is specified or *band* is set to a non-zero value, *putpmsg()* fails and sets *errno* to [EINVAL]. If  
 35889 *flags* is set to MSG\_BAND, then a message is sent in the priority band specified by *band*. If a  
 35890 control part and data part are not specified and *flags* is set to MSG\_BAND, no message is sent  
 35891 and 0 is returned.

35892 The *putmsg()* function blocks if the STREAM write queue is full due to internal flow control  
 35893 conditions, with the following exceptions:

- 35894 • For high-priority messages, *putmsg()* does not block on this condition and continues  
 35895 processing the message.

35896           • For other messages, *putmsg()* does not block but fails when the write queue is full and  
35897            O\_NONBLOCK is set.

35898           The *putmsg()* function also blocks, unless prevented by lack of internal resources, while waiting  
35899           for the availability of message blocks in the STREAM, regardless of priority or whether  
35900           O\_NONBLOCK has been specified. No partial message is sent.

#### 35901 RETURN VALUE

35902           Upon successful completion, *putmsg()* and *putpmsg()* shall return 0; otherwise, they shall return  
35903           -1 and set *errno* to indicate the error.

#### 35904 ERRORS

35905           The *putmsg()* and *putpmsg()* functions shall fail if:

35906           [EAGAIN]        A non-priority message was specified, the O\_NONBLOCK flag is set, and the  
35907                            STREAM write queue is full due to internal flow control conditions; or buffers  
35908                            could not be allocated for the message that was to be created.

35909           [EBADF]        *fildev* is not a valid file descriptor open for writing.

35910           [EINTR]        A signal was caught during *putmsg()*.

35911           [EINVAL]        An undefined value is specified in *flags*, or *flags* is set to RS\_HIPRI or  
35912                            MSG\_HIPRI and no control part is supplied, or the STREAM or multiplexer  
35913                            referenced by *fildev* is linked (directly or indirectly) downstream from a  
35914                            multiplexer, or *flags* is set to MSG\_HIPRI and *band* is non-zero (for *putpmsg()*  
35915                            only).

35916           [ENOSR]        Buffers could not be allocated for the message that was to be created due to  
35917                            insufficient STREAMS memory resources.

35918           [ENOSTR]        A STREAM is not associated with *fildev*.

35919           [ENXIO]        A hangup condition was generated downstream for the specified STREAM.

35920           [EPIPE] or [EIO] The *fildev* argument refers to a STREAMS-based pipe and the other end of the  
35921                            pipe is closed. A SIGPIPE signal is generated for the calling thread.

35922           [ERANGE]        The size of the data part of the message does not fall within the range  
35923                            specified by the maximum and minimum packet sizes of the topmost  
35924                            STREAM module. This value is also returned if the control part of the message  
35925                            is larger than the maximum configured size of the control part of a message,  
35926                            or if the data part of a message is larger than the maximum configured size of  
35927                            the data part of a message.

35928           In addition, *putmsg()* and *putpmsg()* shall fail if the STREAM head had processed an  
35929           asynchronous error before the call. In this case, the value of *errno* does not reflect the result of  
35930           *putmsg()* or *putpmsg()*, but reflects the prior error.

35931 **EXAMPLES**35932 **Sending a High-Priority Message**

35933 The value of *fd* is assumed to refer to an open STREAMS file. This call to *putmsg()* does the  
35934 following:

- 35935 1. Creates a high-priority message with a control part and a data part, using the buffers  
35936 pointed to by *ctrlbuf* and *databuf*, respectively.
- 35937 2. Sends the message to the STREAMS file identified by *fd*.

```
35938 #include <stropts.h>
35939 #include <string.h>
35940 ...
35941 int fd;
35942 char *ctrlbuf = "This is the control part";
35943 char *databuf = "This is the data part";
35944 struct strbuf ctrl;
35945 struct strbuf data;
35946 int ret;

35947 ctrl.buf = ctrlbuf;
35948 ctrl.len = strlen(ctrlbuf);

35949 data.buf = databuf;
35950 data.len = strlen(databuf);

35951 ret = putmsg(fd, &ctrl, &data, MSG_HIPRI);
```

35952 **Using putpmsg()**

35953 This example has the same effect as the previous example. In this example, however, the  
35954 *putpmsg()* function is used to create and send the message to the STREAMS file.

```
35955 #include <stropts.h>
35956 #include <string.h>
35957 ...
35958 int fd;
35959 char *ctrlbuf = "This is the control part";
35960 char *databuf = "This is the data part";
35961 struct strbuf ctrl;
35962 struct strbuf data;
35963 int ret;

35964 ctrl.buf = ctrlbuf;
35965 ctrl.len = strlen(ctrlbuf);

35966 data.buf = databuf;
35967 data.len = strlen(databuf);

35968 ret = putpmsg(fd, &ctrl, &data, 0, MSG_HIPRI);
```

35969 **APPLICATION USAGE**

35970 None.

35971 **RATIONALE**

35972 None.

35973 **FUTURE DIRECTIONS**

35974 None.

35975 **SEE ALSO**35976 *getmsg()*, *poll()*, *read()*, *write()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |  
35977 <**stropts.h**>, Section 2.6 (on page 539)35978 **CHANGE HISTORY**

35979 First released in Issue 4, Version 2.

35980 **Issue 5**

35981 Moved from X/OPEN UNIX extension to BASE.

35982 The following text is removed from the DESCRIPTION: “The STREAM head guarantees that the  
35983 control part of a message generated by *putmsg()* is at least 64 bytes in length”.35984 **Issue 6**

35985 This function is marked as part of the XSI STREAMS Option Group.

35986 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

35987 **NAME**

35988 puts — put a string on standard output

35989 **SYNOPSIS**

35990 #include <stdio.h>

35991 int puts(const char \*s);

35992 **DESCRIPTION**

35993 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
35994 conflict between the requirements described here and the ISO C standard is unintentional. This  
35995 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

35996 The *puts()* function shall write the string pointed to by *s*, followed by a <newline> character, to  
35997 the standard output stream *stdout*. The terminating null byte shall not be written.

35998 cx The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the successful  
35999 execution of *puts()* and the next successful completion of a call to *fflush()* or *fclose()* on the same  
36000 stream or a call to *exit()* or *abort()*.

36001 **RETURN VALUE**

36002 Upon successful completion, *puts()* shall return a non-negative number. Otherwise, it shall  
36003 cx return EOF, shall set an error indicator for the stream, and *errno* shall be set to indicate the error.

36004 **ERRORS**

36005 Refer to *fputc()*.

36006 **EXAMPLES**36007 **Printing to Standard Output**

36008 The following example gets the current time, converts it to a string using *localtime()* and  
36009 *asctime()*, and prints it to standard output using *puts()*. It then prints the number of minutes to  
36010 an event for which it is waiting.

```
36011 #include <time.h>
36012 #include <stdio.h>
36013 ...
36014 time_t now;
36015 int minutes_to_event;
36016 ...
36017 time(&now);
36018 printf("The time is ");
36019 puts(asctime(localtime(&now)));
36020 printf("There are %d minutes to the event.\n",
36021 minutes_to_event);
36022 ...
```

36023 **APPLICATION USAGE**

36024 The *puts()* function appends a <newline> character, while *fputs()* does not.

36025 **RATIONALE**

36026 None.

36027 **FUTURE DIRECTIONS**

36028 None.

36029 **SEE ALSO**

36030 *fopen()*, *fputs()*, *putc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>

36031 **CHANGE HISTORY**

36032 First released in Issue 1. Derived from Issue 1 of the SVID.

36033 **Issue 4**

36034 In the DESCRIPTION, the words “null character” are replaced by “null byte”.

36035 The following change is incorporated for alignment with the ISO C standard:

- 36036 • The type of argument *s* is changed from **char\*** to **const char\***.

36037 **Issue 6**

36038 Extensions beyond the ISO C standard are now marked.

36039 **NAME**

36040 pututxline — put an entry into user accounting database

36041 **SYNOPSIS**

36042 xSI `#include <utmpx.h>`

36043 `struct utmpx *pututxline(const struct utmpx *utmpx);`

36044

36045 **DESCRIPTION**

36046 Refer to *endutxent()*.



36047 **NAME**

36048 putwc — put a wide character on a stream

36049 **SYNOPSIS**

36050 #include &lt;stdio.h&gt;

36051 #include &lt;wchar.h&gt;

36052 wint\_t putwc(wchar\_t *wc*, FILE \**stream*);36053 **DESCRIPTION**

36054 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
36055 conflict between the requirements described here and the ISO C standard is unintentional. This  
36056 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

36057 The *putwc()* function shall be equivalent to *fputwc()*, except that if it is implemented as a macro  
36058 it may evaluate *stream* more than once, so the argument should never be an expression with side  
36059 effects.

36060 **RETURN VALUE**36061 Refer to *fputwc()*.36062 **ERRORS**36063 Refer to *fputwc()*.36064 **EXAMPLES**

36065 None.

36066 **APPLICATION USAGE**

36067 Because it may be implemented as a macro, *putwc()* may treat a *stream* argument with side  
36068 effects incorrectly. In particular, *putwc(wc,\*f++)* need not work correctly. Therefore, use of this  
36069 function is not recommended; *fputwc()* should be used instead.

36070 **RATIONALE**

36071 None.

36072 **FUTURE DIRECTIONS**

36073 None.

36074 **SEE ALSO**36075 *fputwc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>, <wchar.h>36076 **CHANGE HISTORY**

36077 First released as a World-wide Portability Interface in Issue 4.

36078 **Issue 5**

36079 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
36080 is changed from **wint\_t** to **wchar\_t**.

36081 The Optional Header (OH) marking is removed from &lt;stdio.h&gt;.

36082 **NAME**

36083 putwchar — put a wide character on stdout stream

36084 **SYNOPSIS**

36085 #include <wchar.h>

36086 wint\_t putwchar(wchar\_t wc);

36087 **DESCRIPTION**

36088 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
36089 conflict between the requirements described here and the ISO C standard is unintentional. This  
36090 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

36091 The function call *putwchar(wc)* shall be equivalent to *putwc(wc,stdout)*.

36092 **RETURN VALUE**

36093 Refer to *fputwc()*.

36094 **ERRORS**

36095 Refer to *fputwc()*.

36096 **EXAMPLES**

36097 None.

36098 **APPLICATION USAGE**

36099 None.

36100 **RATIONALE**

36101 None.

36102 **FUTURE DIRECTIONS**

36103 None.

36104 **SEE ALSO**

36105 *fputwc()*, *putwc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <wchar.h>

36106 **CHANGE HISTORY**

36107 First released in Issue 4.

36108 **Issue 5**

36109 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
36110 is changed from **wint\_t** to **wchar\_t**.

36111 **NAME**

36112 pwrite, — write on a file

36113 **SYNOPSIS**

36114 #include &lt;unistd.h&gt;

36115 XSI ssize\_t pwrite(int *fildev*, const void \**buf*, size\_t *nbyte*,  
36116 off\_t *offset*);

36117

36118 **DESCRIPTION**36119 Refer to *write()*.

36120 **NAME**

36121 qsort — sort a table of data

36122 **SYNOPSIS**

36123 #include &lt;stdlib.h&gt;

36124 void qsort(void \*base, size\_t nel, size\_t width  
36125 int (\*compar)(const void \*, const void \*));36126 **DESCRIPTION**36127 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
36128 conflict between the requirements described here and the ISO C standard is unintentional. This  
36129 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.36130 The *qsort()* function sorts an array of *nel* objects, the initial element of which is pointed to by  
36131 *base*. The size of each object, in bytes, is specified by the *width* argument.36132 The contents of the array are sorted in ascending order according to a comparison function. The  
36133 *compar* argument is a pointer to the comparison function, which is called with two arguments  
36134 that point to the elements being compared. The application shall ensure that the function returns  
36135 an integer less than, equal to, or greater than 0, if the first argument is considered respectively  
36136 less than, equal to, or greater than the second. If two members compare as equal, their order in  
36137 the sorted array is unspecified.36138 **RETURN VALUE**36139 The *qsort()* function shall return no value.36140 **ERRORS**

36141 No errors are defined.

36142 **EXAMPLES**

36143 None.

36144 **APPLICATION USAGE**36145 The comparison function need not compare every byte, so arbitrary data may be contained in  
36146 the elements in addition to the values being compared.36147 **RATIONALE**

36148 None.

36149 **FUTURE DIRECTIONS**

36150 None.

36151 **SEE ALSO**

36152 The Base Definitions volume of IEEE Std. 1003.1-200x, &lt;stdlib.h&gt;

36153 **CHANGE HISTORY**

36154 First released in Issue 1. Derived from Issue 1 of the SVID.

36155 **Issue 4**

36156 The following change is incorporated for alignment with the ISO C standard:

- 36157
- The arguments to *compar* are formally defined in the SYNOPSIS section.

36158 **Issue 6**

36159 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

36160 **NAME**

36161 raise — send a signal to the executing process

36162 **SYNOPSIS**

36163 #include &lt;signal.h&gt;

36164 int raise(int *sig*);36165 **DESCRIPTION**

36166 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 36167 conflict between the requirements described here and the ISO C standard is unintentional. This  
 36168 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

36169 CX The *raise()* function shall send the signal *sig* to the executing thread or process. If a signal  
 36170 handler is called, the *raise()* function shall not return until after the signal handler does.

36171 THR If the implementation supports the Threads option, the effect of the *raise()* function is equivalent  
 36172 to calling:

36173 pthread\_kill(pthread\_self(), sig);

36174

36175 CX Otherwise, the effect of the *raise()* function is equivalent to calling:

36176 kill(getpid(), sig);

36177

36178 **RETURN VALUE**

36179 CX Upon successful completion, 0 shall be returned. Otherwise, a non-zero value shall be returned  
 36180 and *errno* shall be set to indicate the error.

36181 **ERRORS**36182 The *raise()* function shall fail if:36183 CX [EINVAL] The value of the *sig* argument is an invalid signal number.36184 **EXAMPLES**

36185 None.

36186 **APPLICATION USAGE**

36187 None.

36188 **RATIONALE**

36189 The term “thread” is an extension to the ISO C standard.

36190 **FUTURE DIRECTIONS**

36191 None.

36192 **SEE ALSO**

36193 *kill()*, *sigaction()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <signal.h>,  
 36194 <sys/types.h>

36195 **CHANGE HISTORY**

36196 First released in Issue 4. Derived from the ANSI C standard.

36197 **Issue 5**

36198 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

36199 **Issue 6**

36200 Extensions beyond the ISO C standard are now marked.

36201 The following new requirements on POSIX implementations derive from alignment with the  
36202 Single UNIX Specification:

- 36203 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
- 36204 • The [EINVAL] error condition is added.

36205 **NAME**

36206 rand, rand\_r, srand — pseudo-random number generator

36207 **SYNOPSIS**

```
36208 #include <stdlib.h>
36209 int rand(void);
36210 TSF int rand_r(unsigned *seed);
36211 void srand(unsigned seed);
```

36212 **DESCRIPTION**

36213 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 36214 conflict between the requirements described here and the ISO C standard is unintentional. This  
 36215 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

36216 The *rand()* function shall compute a sequence of pseudo-random integers in the range 0 to  
 36217 XSI {RAND\_MAX} with a period of at least  $2^{32}$ .

36218 CX The *rand()* function need not be reentrant. A function that is not required to be reentrant is not  
 36219 required to be thread-safe.

36220 TSF The *rand\_r()* function shall compute a sequence of pseudo-random integers in the range 0 to  
 36221 {RAND\_MAX}. (The value of the {RAND\_MAX} macro shall be at least 32 767.)

36222 If *rand\_r()* is called with the same initial value for the object pointed to by *seed* and that object is  
 36223 not modified between successive returns and calls to *rand\_r()*, the same sequence shall be  
 36224 generated.

36225 The *srand()* function uses the argument as a seed for a new sequence of pseudo-random  
 36226 numbers to be returned by subsequent calls to *rand()*. If *srand()* is then called with the same  
 36227 seed value, the sequence of pseudo-random numbers shall be repeated. If *rand()* is called before  
 36228 any calls to *srand()* are made, the same sequence shall be generated as when *srand()* is first  
 36229 called with a seed value of 1.

36230 The implementation shall behave as if no function defined in this volume of  
 36231 IEEE Std. 1003.1-200x calls *rand()* or *srand()*.

36232 **RETURN VALUE**

36233 The *rand()* function shall return the next pseudo-random number in the sequence.

36234 TSF The *rand\_r()* function shall return a pseudo-random integer.

36235 The *srand()* function shall return no value.

36236 **ERRORS**

36237 No errors are defined.

36238 **EXAMPLES**36239 **Generating a Pseudo-Random Number Sequence**

36240 The following example demonstrates how to generate a sequence of pseudo-random numbers.

```
36241 #include <stdio.h>
36242 #include <stdlib.h>
36243 ...
36244 long count, i;
36245 char *keystri;
36246 int elementlen, len;
36247 char c;
```

```

36248 ...
36249 /* Initial random number generator. */
36250 srand(1);

36251 /* Create keys using only lower case characters */
36252 len = 0;
36253 for (i=0; i<count; i++) {
36254 while (len < elementlen) {
36255 c = (char) (rand() % 128);
36256 if (islower(c))
36257 keystr[len++] = c;
36258 }

36259 keystr[len] = '\0';
36260 printf("%s Element%0*ld\n", keystr, elementlen, i);
36261 len = 0;
36262 }

```

### 36263 **Generating the Same Sequence on Different Machines**

36264 The following code defines a pair of functions that could be incorporated into applications  
 36265 wishing to ensure that the same sequence of numbers is generated across different machines.

```

36266 static unsigned long next = 1;
36267 int myrand(void) /* RAND_MAX assumed to be 32767. */
36268 {
36269 next = next * 1103515245 + 12345;
36270 return((unsigned)(next/65536) % 32768);
36271 }

36272 void mysrand(unsigned seed)
36273 {
36274 next = seed;
36275 }

```

### 36276 **APPLICATION USAGE**

36277 The *drand48()* function provides a much more elaborate random number generator.

### 36278 **RATIONALE**

36279 The ISO C standard *rand()* and *srand()* functions allow per-process pseudo-random streams  
 36280 shared by all threads. Those two functions need not change, but there has to be mutual-  
 36281 exclusion that prevents interference between two threads concurrently accessing the random  
 36282 number generator.

36283 With regard to *rand()*, there are two different behaviors that may be wanted in a multi-threaded  
 36284 program:

- 36285 1. A single per-process sequence of pseudo-random numbers that is shared by all threads  
 36286 that call *rand()*
- 36287 2. A different sequence of pseudo-random numbers for each thread that calls *rand()*

36288 This is provided by the modified thread-safe function based on whether the seed value is global  
 36289 to the entire process or local to each thread.

36290 This does not address the known deficiencies of the *rand()* function implementations, which  
 36291 have been approached by maintaining more state. In effect, this specifies new thread-safe forms  
 36292 of a deficient function. Since alternatives to *rand()* are not standardized, they are not modified as



- 36293 part of this volume of IEEE Std. 1003.1-200x.
- 36294 **FUTURE DIRECTIONS**
- 36295 None.
- 36296 **SEE ALSO**
- 36297 *drand48()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>
- 36298 **CHANGE HISTORY**
- 36299 First released in Issue 1. Derived from Issue 1 of the SVID.
- 36300 **Issue 4**
- 36301 The definition of *srand()* is added to the SYNOPSIS section.
- 36302 In the DESCRIPTION, the text referring to the period of pseudo-random numbers is marked as  
36303 an extension.
- 36304 The example in the APPLICATION USAGE section is updated as follows:
- 36305 • To use ISO C standard syntax.
- 36306 • To avoid name clashes with standard functions.
- 36307 The following changes are incorporated for alignment with the ISO C standard:
- 36308 • The argument list of *rand()* is explicitly defined as **void**.
- 36309 • The argument *seed* is explicitly defined as **unsigned**.
- 36310 **Issue 5**
- 36311 The *rand\_r()* function is included for alignment with the POSIX Threads Extension.
- 36312 A note indicating that the *rand()* function need not be reentrant is added to the DESCRIPTION.
- 36313 **Issue 6**
- 36314 Extensions beyond the ISO C standard are now marked.
- 36315 The *rand\_r()* function is marked as part of the Thread-Safe Functions option.

36316 **NAME**

36317           random — generate pseudorandom number

36318 **SYNOPSIS**

36319 xSI       #include <stdlib.h>

36320           long random(void);

36321

36322 **DESCRIPTION**

36323           Refer to *initstate()*.

36324 **NAME**36325 `pread, read, readv` — read from a file36326 **SYNOPSIS**36327 `#include <unistd.h>`36328 XSI `ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);`36329 `ssize_t read(int fildes, void *buf, size_t nbyte);`36330 XSI `#include <sys/uio.h>`36331 `ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);`

36332

36333 **DESCRIPTION**

36334 The `read()` function attempts to read *nbyte* bytes from the file associated with the open file  
 36335 descriptor, *fildes*, into the buffer pointed to by *buf*. The behavior of multiple concurrent reads on  
 36336 the same pipe, FIFO, or terminal device is unspecified.

36337 If *nbyte* is zero, the `read()` function may detect and return errors as described below. In the  
 36338 absence of errors, or if error detection is not performed, the `read()` function shall return zero and  
 36339 have no other results.

36340 On files that support seeking (for example, a regular file), the `read()` starts at a position in the file  
 36341 given by the file offset associated with *fildes*. The file offset is incremented by the number of  
 36342 bytes actually read.

36343 Files that do not support seeking—for example, terminals—always read from the current  
 36344 position. The value of a file offset associated with such a file is undefined.

36345 No data transfer shall occur past the current end-of-file. If the starting position is at or after the  
 36346 end-of-file, 0 shall be returned. If the file refers to a device special file, the result of subsequent  
 36347 `read()` requests is implementation-defined.

36348 If the value of *nbyte* is greater than {SSIZE\_MAX}, the result is implementation-defined.

36349 When attempting to read from an empty pipe or FIFO:

- 36350 • If no process has the pipe open for writing, `read()` shall return 0 to indicate end-of-file.
- 36351 • If some process has the pipe open for writing and O\_NONBLOCK is set, `read()` shall return  
 36352 -1 and set *errno* to [EAGAIN].
- 36353 • If some process has the pipe open for writing and O\_NONBLOCK is clear, `read()` shall block  
 36354 the calling thread until some data is written or the pipe is closed by all processes that had the  
 36355 pipe open for writing.

36356 When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and  
 36357 has no data currently available:

- 36358 • If O\_NONBLOCK is set, `read()` shall return -1 and set *errno* to [EAGAIN].
- 36359 • If O\_NONBLOCK is clear, `read()` shall block the calling thread until some data becomes  
 36360 available.
- 36361 • The use of the O\_NONBLOCK flag has no effect if there is some data available.

36362 The `read()` function reads data previously written to a file. If any portion of a regular file prior to  
 36363 the end-of-file has not been written, `read()` shall return bytes with value 0. For example, `lseek()`  
 36364 allows the file offset to be set beyond the end of existing data in the file. If data is later written at  
 36365 this point, subsequent reads in the gap between the previous end of data and the newly written  
 36366 data shall return bytes with value 0 until data is written into the gap.

36367 Upon successful completion, where *nbyte* is greater than 0, *read()* shall mark for update the  
36368 *st\_atime* field of the file, and shall return the number of bytes read. This number shall never be  
36369 greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes left in the  
36370 file is less than *nbyte*, if the *read()* request was interrupted by a signal, or if the file is a pipe or  
36371 FIFO or special file and has fewer than *nbyte* bytes immediately available for reading. For  
36372 example, a *read()* from a file associated with a terminal may return one typed line of data.

36373 If a *read()* is interrupted by a signal before it reads any data, it shall return  $-1$  with *errno* set to  
36374 [EINTR].

36375 If a *read()* is interrupted by a signal after it has successfully read some data, it shall return the  
36376 number of bytes read.

36377 XSR A *read()* from a STREAMS file can read data in three different modes: *byte-stream* mode,  
36378 *message-nondiscard* mode, and *message-discard* mode. The default is byte-stream mode. This can  
36379 be changed using the *I\_SRDOPT ioctl()* request, and can be tested with the *I\_GRDOPT ioctl()*. In  
36380 byte-stream mode, *read()* retrieves data from the STREAM until as many bytes as were  
36381 requested are transferred, or until there is no more data to be retrieved. Byte-stream mode  
36382 ignores message boundaries.

36383 In STREAMS message-nondiscard mode, *read()* retrieves data until as many bytes as were  
36384 requested are transferred, or until a message boundary is reached. If *read()* does not retrieve all  
36385 the data in a message, the remaining data is left on the STREAM, and can be retrieved by the  
36386 next *read()* call. Message-discard mode also retrieves data until as many bytes as were requested  
36387 are transferred, or a message boundary is reached. However, unread data remaining in a  
36388 message after the *read()* returns is discarded, and is not available for a subsequent *read()*,  
36389 *readv()*, or *getmsg()* call.

36390 How *read()* handles zero-byte STREAMS messages is determined by the current read mode  
36391 setting. In byte-stream mode, *read()* accepts data until it has read *nbyte* bytes, or until there is no  
36392 more data to read, or until a zero-byte message block is encountered. The *read()* function shall  
36393 then return the number of bytes read, and place the zero-byte message back on the STREAM to  
36394 be retrieved by the next *read()*, *readv()*, or *getmsg()*. In message-nondiscard mode or message-  
36395 discard mode, a zero-byte message shall return 0 and the message shall be removed from the  
36396 STREAM. When a zero-byte message is read as the first message on a STREAM, the message  
36397 shall be removed from the STREAM and 0 shall be returned, regardless of the read mode.

36398 A *read()* from a STREAMS file shall return the data in the message at the front of the STREAM  
36399 head read queue, regardless of the priority band of the message.

36400 By default, STREAMS are in control-normal mode, in which a *read()* from a STREAMS file can  
36401 only process messages that contain a data part but do not contain a control part. The *read()* shall  
36402 fail if a message containing a control part is encountered at the STREAM head. This default  
36403 action can be changed by placing the STREAM in either control-data mode or control-discard  
36404 mode with the *I\_SRDOPT ioctl()* command. In control-data mode, *read()* converts any control  
36405 part to data and passes it to the application before passing any data part originally present in the  
36406 same message. In control-discard mode, *read()* shall discard message control parts but return to  
36407 the process any data part in the message.

36408 In addition, *read()* and *readv()* fail if the STREAM head had processed an asynchronous error  
36409 before the call. In this case, the value of *errno* does not reflect the result of *read()* or *readv()*, but  
36410 reflects the prior error. If a hangup occurs on the STREAM being read, *read()* continues to  
36411 operate normally until the STREAM head read queue is empty. Thereafter, it shall return 0.

36412 XSI The *readv()* function is equivalent to *read()*, but places the input data into the *iovcnt* buffers  
36413 specified by the members of the *iov* array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*-1]. The *iovcnt* argument is  
36414 valid if greater than 0 and less than or equal to {IOV\_MAX}.

- 36415 Each *iovec* entry specifies the base address and length of an area in memory where data should  
36416 be placed. The *readv()* function always fills an area completely before proceeding to the next.
- 36417 Upon successful completion, *readv()* shall mark for update the *st\_atime* field of the file.
- 36418 SIO If the O\_DSYNC and O\_RSYNC bits have been set, read I/O operations on the file descriptor  
36419 complete as defined by synchronized I/O data integrity completion. If the O\_SYNC and  
36420 O\_RSYNC bits have been set, read I/O operations on the file descriptor complete as defined by  
36421 synchronized I/O file integrity completion.
- 36422 SHM If *fildev* refers to a shared memory object, the result of the *read()* function is unspecified.
- 36423 TYM If *fildev* refers to a typed memory object, the result of the *read()* function is unspecified.
- 36424 For regular files, no data transfer shall occur past the offset maximum established in the open  
36425 file description associated with *fildev*.
- 36426 XSI The *pread()* function performs the same action as *read()*, except that it reads from a given  
36427 position in the file without changing the file pointer. The first three arguments to *pread()* are the  
36428 same as *read()* with the addition of a fourth argument offset for the desired position inside the  
36429 file. An attempt to perform a *pread()* on a file that is incapable of seeking results in an error.
- 36430 If *fildev* refers to a socket, *read()* is equivalent to *recv()* with no flags set.
- 36431 **RETURN VALUE**
- 36432 XSI Upon successful completion, *read()*, *pread()* and *readv()* shall return a non-negative integer  
36433 indicating the number of bytes actually read. Otherwise, the functions shall return -1 and set  
36434 *errno* to indicate the error.
- 36435 **ERRORS**
- 36436 XSI The *read()*, *pread()* and *readv()* functions shall fail if:
- 36437 [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor and the process would be  
36438 delayed.
- 36439 [EBADF] The *fildev* argument is not a valid file descriptor open for reading.
- 36440 XSR [EBADMSG] The file is a STREAM file that is set to control-normal mode and the message  
36441 waiting to be read includes a control part.
- 36442 [EINTR] The read operation was terminated due to the receipt of a signal, and no data  
36443 was transferred.
- 36444 XSR [EINVAL] The STREAM or multiplexer referenced by *fildev* is linked (directly or  
36445 indirectly) downstream from a multiplexer.
- 36446 [EIO] The process is a member of a background process attempting to read from its  
36447 controlling terminal, the process is ignoring or blocking the SIGTTIN signal,  
36448 or the process group is orphaned. This error may also be generated for  
36449 implementation-defined reasons.
- 36450 XSI [EISDIR] The *fildev* argument refers to a directory and the implementation does not  
36451 allow the directory to be read using *read()*, *pread()*, or *readv()*. The *readdir()*  
36452 function should be used instead.
- 36453 [EOVERFLOW] The file is a regular file, *nbyte* is greater than 0, the starting position is before  
36454 the end-of-file, and the starting position is greater than or equal to the offset  
36455 maximum established in the open file description associated with *fildev*.
- 36456 MAN The *read()* function shall fail if:

|       |                           |                                                                                              |
|-------|---------------------------|----------------------------------------------------------------------------------------------|
| 36457 | [EAGAIN] or [EWOULDBLOCK] |                                                                                              |
| 36458 |                           | The file descriptor is for a connection-made socket, is marked                               |
| 36459 |                           | O_NONBLOCK, and no data is waiting to be received.                                           |
| 36460 | [ECONNRESET]              | A read was attempted on a connection-mode socket and the connection was                      |
| 36461 |                           | forcibly closed by its peer.                                                                 |
| 36462 | [ENOTCONN]                | A read was attempted on a connection-made socket that is not connected.                      |
| 36463 | [ETIMEDOUT]               | A read was attempted on a connection-mode socket and a transmission                          |
| 36464 |                           | timeout occurred.                                                                            |
| 36465 |                           |                                                                                              |
| 36466 |                           | The <i>readv()</i> function shall fail if:                                                   |
| 36467 | [EINVAL]                  | The sum of the <i>iov_len</i> values in the <i>iov</i> array overflowed an <i>ssize_t</i> .  |
| 36468 | XSI                       | The <i>read()</i> , <i>pread()</i> and <i>readv()</i> functions may fail if:                 |
| 36469 | MAN                       | [EIO] A physical I/O error has occurred.                                                     |
| 36470 | MAN                       | [ENOBUFS] Insufficient resources were available in the system to perform the operation.      |
| 36471 | MAN                       | [ENOMEM] Insufficient memory was available to fulfill the request.                           |
| 36472 |                           | [ENXIO] A request was made of a nonexistent device, or the request was outside the           |
| 36473 |                           | capabilities of the device.                                                                  |
| 36474 |                           | The <i>readv()</i> function may fail if:                                                     |
| 36475 | XSI                       | [EINVAL] The <i>iovcnt</i> argument was less than or equal to 0, or greater than {IOV_MAX}.  |
| 36476 | XSI                       | The <i>pread()</i> function shall fail, and the file pointer shall remain unchanged, if:     |
| 36477 | XSI                       | [EINVAL] The <i>offset</i> argument is invalid. The value is negative.                       |
| 36478 | XSI                       | [EOVERFLOW] The file is a regular file and an attempt was made to read or write at or beyond |
| 36479 |                           | the offset maximum associated with the file.                                                 |
| 36480 | XSI                       | [ENXIO] A request was outside the capabilities of the device.                                |
| 36481 | XSI                       | [ESPIPE] <i>fdes</i> is associated with a pipe or FIFO.                                      |

36482 **EXAMPLES**36483 **Reading Data into a Buffer**

36484 The following example reads data from the file associated with the file descriptor *fd* into the  
 36485 buffer pointed to by *buf*.

```

36486 #include <sys/types.h>
36487 #include <unistd.h>
36488 ...
36489 char buf[20];
36490 size_t nbytes;
36491 ssize_t bytes_read;
36492 int fd;
36493 ...
36494 nbytes = sizeof(buf);
36495 bytes_read = read(fd, buf, nbytes);
36496 ...

```

36497 **Reading Data into an Array**

36498 The following example reads data from the file associated with the file descriptor *fd* into the  
 36499 buffers specified by members of the *iov* array.

```

36500 #include <sys/types.h>
36501 #include <sys/uio.h>
36502 #include <unistd.h>
36503 ...
36504 ssize_t bytes_read;
36505 int fd;
36506 char buf0[20];
36507 char buf1[30];
36508 char buf2[40];
36509 int iovcnt;
36510 struct iovec iov[3];

36511 iov[0].iov_base = buf0;
36512 iov[0].iov_len = sizeof(buf0);
36513 iov[1].iov_base = buf1;
36514 iov[1].iov_len = sizeof(buf1);
36515 iov[2].iov_base = buf2;
36516 iov[2].iov_len = sizeof(buf2);
36517 ...
36518 iovcnt = sizeof(iov) / sizeof(struct iovec);

36519 bytes_read = readv(fd, iov, iovcnt);
36520 ...

```

36521 **APPLICATION USAGE**

36522 None.

36523 **RATIONALE**

36524 This volume of IEEE Std. 1003.1-200x does not specify the value of the file offset after an error is  
 36525 returned; there are too many cases. For programming errors, such as [EBADF], the concept is  
 36526 meaningless since no file is involved. For errors that are detected immediately, such as  
 36527 [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however,  
 36528 an updated value would be very useful and is the behavior of many implementations.

36529 Note that a *read()* of zero bytes does not modify *st\_atime*. A *read()* that requests more than zero  
 36530 bytes, but returns zero, shall modify *st\_atime*.

36531 Implementations are allowed, but not required, to perform error checking for *read()* requests of  
 36532 zero bytes.

36533 **Input and Output**

36534 The use of I/O with large byte counts has always presented problems. Ideas such as *lread()* and  
 36535 *lwrite()* (using and returning **long**s) were considered at one time. The current solution is to use  
 36536 abstract types on the ISO C standard function to *read()* and *write()*. The abstract types can be  
 36537 declared so that existing functions work, but can also be declared so that larger types can be  
 36538 represented in future implementations. It is presumed that whatever constraints limit the  
 36539 maximum range of **size\_t** also limit portable I/O requests to the same range. This volume of  
 36540 IEEE Std. 1003.1-200x also limits the range further by requiring that the byte count be limited so  
 36541 that a signed return value remains meaningful. Since the return type is also a (signed) abstract  
 36542 type, the byte count can be defined by the implementation to be larger than an **int** can hold.

36543 The standard developers considered adding atomicity requirements to a pipe or FIFO, but  
36544 recognized that due to the nature of pipes and FIFOs there could be no guarantee of atomicity of  
36545 reads of {PIPE\_BUF} or any other size that would be an aid to applications portability.

36546 This volume of IEEE Std. 1003.1-200x requires that no action be taken when *nbyte* is zero. This is  
36547 not intended to take precedence over detection of errors (such as invalid buffer pointers or file  
36548 descriptors). This is consistent with the rest of this volume of IEEE Std. 1003.1-200x, but the  
36549 phrasing here could be misread to require detection of the zero case before any other errors. A  
36550 value of zero is to be considered a correct value, for which the semantics are a no-op.

36551 I/O is intended to be atomic to ordinary files and pipes and FIFOs. Atomic means that all the  
36552 bytes from a single operation that started out together end up together, without interleaving  
36553 from other I/O operations. It is a known attribute of terminals that this is not honored, and  
36554 terminals are explicitly (and implicitly permanently) excepted, making the behavior unspecified.  
36555 The behavior for other device types is also left unspecified, but the wording is intended to imply  
36556 that future standards might choose to specify atomicity (or not).

36557 There were recommendations to add format parameters to *read()* and *write()* in order to handle  
36558 networked transfers among heterogeneous file system and base hardware types. Such a facility  
36559 may be required for support by the OSI presentation of layer services. However, it was  
36560 determined that this should correspond with similar C-language facilities, and that is beyond the  
36561 scope of this volume of IEEE Std. 1003.1-200x. The concept was suggested to the developers of  
36562 the ISO C standard for their consideration as a possible area for future work.

36563 In 4.3 BSD, a *read()* or *write()* that is interrupted by a signal before transferring any data does not  
36564 by default return an [EINTR] error, but is restarted. In 4.2 BSD, 4.3 BSD, and the Eighth Edition,  
36565 there is an additional function, *select()*, whose purpose is to pause until specified activity (data  
36566 to read, space to write, and so on) is detected on specified file descriptors. It is common in  
36567 applications written for those systems for *select()* to be used before *read()* in situations (such as  
36568 keyboard input) where interruption of I/O due to a signal is desired.

36569 The issue of which files or file types are interruptible is considered an implementation design  
36570 issue. This is often affected primarily by hardware and reliability issues.

36571 There are no references to actions taken following an “unrecoverable error”. It is considered  
36572 beyond the scope of this volume of IEEE Std. 1003.1-200x to describe what happens in the case of  
36573 hardware errors.

36574 Previous versions of IEEE Std. 1003.1-200x allowed two very different behaviors with regard to  
36575 the handling of interrupts. In order to minimize the resulting confusion, it was decided that  
36576 IEEE Std. 1003.1-200x should support only one of these behaviors. Historical practice on AT&T-  
36577 derived systems was to have *read()* and *write()* return  $-1$  and set *errno* to [EINTR] when  
36578 interrupted after some, but not all, of the data requested had been transferred. However, the U.S.  
36579 Department of Commerce FIPS 151-1 and FIPS 151-2 require the historical BSD behavior, in  
36580 which *read()* and *write()* return the number of bytes actually transferred before the interrupt. If  
36581  $-1$  is returned when any data is transferred, it is difficult to recover from the error on a seekable  
36582 device and impossible on a non-seekable device. Most new implementations support this  
36583 behavior. The behavior required by IEEE Std. 1003.1-200x is to return the number of bytes  
36584 transferred.

36585 IEEE Std. 1003.1-200x does not specify when an implementation that buffers *read()*s actually  
36586 moves the data into the user-supplied buffer, so an implementation may chose to do this at the  
36587 latest possible moment. Therefore, an interrupt arriving earlier may not cause *read()* to return a  
36588 partial byte count, but rather to return  $-1$  and set *errno* to [EINTR].

36589 Consideration was also given to combining the two previous options, and setting *errno* to  
36590 [EINTR] while returning a short count. However, not only is there no existing practice that



36591 implements this, it is also contradictory to the idea that when *errno* is set, the function  
36592 responsible shall return  $-1$ .

#### 36593 FUTURE DIRECTIONS

36594 None.

#### 36595 SEE ALSO

36596 *fcntl()*, *ioctl()*, *lseek()*, *open()*, *pipe()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
36597 **<stropts.h>**, **<sys/uio.h>**, **<unistd.h>**, the Base Definitions volume of IEEE Std. 1003.1-200x,  
36598 Chapter 11, General Terminal Interface

#### 36599 CHANGE HISTORY

36600 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 36601 Issue 4

36602 The **<unistd.h>** header is added to the SYNOPSIS section.

36603 The DESCRIPTION is rearranged for clarity and to align more closely with the ISO POSIX-1  
36604 standard. No functional changes are made other than as noted elsewhere in this CHANGE  
36605 HISTORY section.

36606 In the ERRORS section in previous issues, generation of the [EIO] error depended on whether or  
36607 not an implementation supported Job Control. This functionality is now defined as mandatory.

36608 The [ENXIO] error is marked as an extension.

36609 The APPLICATION USAGE section is removed.

36610 The description of [EINTR] is amended.

36611 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 36612 • The type of the argument *buf* is changed from **char\*** to **void\***, and the type of the argument  
36613 *nbyte* is changed from **unsigned** to **size\_t**.

- 36614 • The DESCRIPTION now states that the result is implementation-defined if *nbyte* is greater  
36615 than {SSIZE\_MAX}. This limit was defined by the constant {INT\_MAX} in Issue 3.

36616 The following change is incorporated for alignment with the FIPS requirements:

- 36617 • The last paragraph of the DESCRIPTION now states that if *read()* is interrupted by a signal  
36618 after it has successfully read some data, it returns the number of bytes read. In Issue 3, it was  
36619 optional whether *read()* returned the number of bytes read, or whether it returned  $-1$  with  
36620 *errno* set to [EINTR].

#### 36621 Issue 4, Version 2

36622 The following changes are incorporated for X/OPEN UNIX conformance:

- 36623 • The *readv()* function is added to the SYNOPSIS.

- 36624 • The DESCRIPTION is updated to describe the reading of data from STREAMS files. An  
36625 operational description of the *readv()* function is also added.

- 36626 • References to the *readv()* function are added to the RETURN VALUE and ERRORS sections  
36627 in appropriate places.

- 36628 • The ERRORS section has been restructured to describe errors that apply generally (that is, to  
36629 both *read()* and *readv()*), and to describe those that apply to *readv()* specifically. The  
36630 [EBADMSG], [EINVAL], and [EISDIR] errors are also added.

36631 **Issue 5**

36632 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
36633 Threads Extension.

36634 Large File Summit extensions are added.

36635 The *pread()* function is added.

36636 **Issue 6**

36637 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
36638 marked as part of the XSI STREAMS Option Group.

36639 The following new requirements on POSIX implementations derive from alignment with the  
36640 Single UNIX Specification:

36641 • The DESCRIPTION now states that if *read()* is interrupted by a signal after it has successfully  
36642 read some data, it returns the number of bytes read. In Issue 3, it was optional whether *read()*  
36643 returned the number of bytes read, or whether it returned  $-1$  with *errno* set to [EINTR]. This  
36644 is a FIPS requirement.

36645 • In the DESCRIPTION, text is added to indicate that for regular files, no data transfer occurs  
36646 past the offset maximum established in the open file description associated with *files*. This  
36647 change is to support large files.

36648 • The [EOVERFLOW] mandatory error condition is added.

36649 • The [ENXIO] optional error condition is added.

36650 Text referring to sockets is added to the DESCRIPTION.

36651 The following changes were made to align with the IEEE P1003.1a draft standard:

36652 • The effect of reading zero bytes is clarified.

36653 The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by specifying that  
36654 *read()* results are unspecified for typed memory objects.

36655 New RATIONALE is added to explain the atomicity requirements for input and output  
36656 operations.

36657 The following error conditions are added for operations on sockets: [EAGAIN],  
36658 [ECONNRESET], [ENOTCONN], and [ETIMEDOUT].

36659 The [EIO] error is changed to “may fail”.

36660 The following error conditions are added for operations on sockets: [ENOBUFFS] and  
36661 [ENOMEM].

## 36662 NAME

36663 readdir, readdir\_r — read directory

## 36664 SYNOPSIS

36665 #include &lt;dirent.h&gt;

36666 struct dirent \*readdir(DIR \*dirp);

36667 TSF int readdir\_r(DIR \*restrict dirp, struct dirent \*restrict entry,

36668 struct dirent \*\*restrict result);

36669

## 36670 DESCRIPTION

36671 The type **DIR**, which is defined in the header <dirent.h>, represents a *directory stream*, which is  
 36672 an ordered sequence of all the directory entries in a particular directory. Directory entries  
 36673 represent files; files may be removed from a directory or added to a directory asynchronously to  
 36674 the operation of *readdir()*.

36675 The *readdir()* function shall return a pointer to a structure representing the directory entry at the  
 36676 current position in the directory stream specified by the argument *dirp*, and position the  
 36677 directory stream at the next entry. It shall return a null pointer upon reaching the end of the  
 36678 directory stream. The structure **dirent** defined by the <dirent.h> header describes a directory  
 36679 entry.

36680 The *readdir()* function shall not return directory entries containing empty names. If entries for  
 36681 dot or dot-dot exist, one entry shall be returned for dot and one entry shall be returned for dot-  
 36682 dot; otherwise, they shall not be returned.

36683 The pointer returned by *readdir()* points to data which may be overwritten by another call to  
 36684 *readdir()* on the same directory stream. This data is not overwritten by another call to *readdir()*  
 36685 on a different directory stream.

36686 If a file is removed from or added to the directory after the most recent call to *opendir()* or  
 36687 *rewinddir()*, whether a subsequent call to *readdir()* returns an entry for that file is unspecified.

36688 The *readdir()* function may buffer several directory entries per actual read operation; *readdir()*  
 36689 shall mark for update the *st\_atime* field of the directory each time the directory is actually read.

36690 After a call to *fork()*, either the parent or child (but not both) may continue processing the  
 36691 XSI directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and child processes  
 36692 use these functions, the result is undefined.

36693 If the entry names a symbolic link, the value of the *d\_ino* member is unspecified.

36694 The *readdir()* function need not be reentrant. A function that is not required to be reentrant is not  
 36695 required to be thread-safe.

36696 TSF The *readdir\_r()* function initializes the **dirent** structure referenced by *entry* to represent the  
 36697 directory entry at the current position in the directory stream referred to by *dirp*, store a pointer  
 36698 to this structure at the location referenced by *result*, and position the directory stream at the next  
 36699 entry.

36700 The storage pointed to by *entry* shall be large enough for a **dirent** with an array of **char** *d\_name*  
 36701 members containing at least {NAME\_MAX} plus one elements.

36702 Upon successful return, the pointer returned at *\*result* shall have the same value as the argument  
 36703 *entry*. Upon reaching the end of the directory stream, this pointer shall have the value NULL.

36704 The *readdir\_r()* function shall not return directory entries containing empty names.

36705 If a file is removed from or added to the directory after the most recent call to *opendir()* or  
 36706 *rewinddir()*, whether a subsequent call to *readdir\_r()* returns an entry for that file is unspecified.

36707 The *readdir\_r()* function may buffer several directory entries per actual read operation; the  
 36708 *readdir\_r()* function shall mark for update the *st\_atime* field of the directory each time the  
 36709 directory is actually read.

36710 Applications wishing to check for error situations should set *errno* to 0 before calling *readdir()*. If  
 36711 *errno* is set to non-zero on return, an error occurred.

#### 36712 RETURN VALUE

36713 Upon successful completion, *readdir()* shall return a pointer to an object of type **struct dirent**.  
 36714 When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate  
 36715 the error. When the end of the directory is encountered, a null pointer shall be returned and *errno*  
 36716 is not changed.

36717 TSF If successful, the *readdir\_r()* function shall return zero; otherwise, an error number shall be  
 36718 returned to indicate the error.

#### 36719 ERRORS

36720 The *readdir()* function shall fail if:

36721 [EOverflow] One of the values in the structure to be returned cannot be represented  
 36722 correctly.

36723 The *readdir()* function may fail if:

36724 [EBADF] The *dirp* argument does not refer to an open directory stream.

36725 [ENOENT] The current position of the directory stream is invalid.

36726 The *readdir\_r()* function may fail if:

36727 [EBADF] The *dirp* argument does not refer to an open directory stream.

#### 36728 EXAMPLES

36729 The following sample code searches the current directory for the entry *name*:

```
36730 dirp = opendir(".");
36731 while (dirp) {
36732 errno = 0;
36733 if ((dp = readdir(dirp)) != NULL) {
36734 if (strcmp(dp->d_name, name) == 0) {
36735 closedir(dirp);
36736 return FOUND;
36737 }
36738 } else {
36739 if (errno == 0) {
36740 closedir(dirp);
36741 return NOT_FOUND;
36742 }
36743 closedir(dirp);
36744 return READ_ERROR;
36745 }
36746 }
36747 return OPEN_ERROR;
```

36748 **APPLICATION USAGE**

36749 The *readdir()* function should be used in conjunction with *opendir()*, *closedir()*, and *rewinddir()* to  
 36750 examine the contents of the directory.

36751 The *readdir\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
 36752 of possibly using a static data area that may be overwritten by each call.

36753 **RATIONALE**

36754 The returned value of *readdir()* merely *represents* a directory entry. No equivalence should be  
 36755 inferred.

36756 Historical implementations of *readdir()* obtain multiple directory entries on a single read  
 36757 operation, which permits subsequent *readdir()* operations to operate from the buffered  
 36758 information. Any wording that required each successful *readdir()* operation to mark the  
 36759 directory *st\_atime* field for update would militate against the historical performance-oriented  
 36760 implementations.

36761 Since *readdir()* returns NULL when it detects an error and when the end of the directory is  
 36762 encountered, an application that needs to tell the difference must set *errno* to zero before the call  
 36763 and check it if NULL is returned. Because the function must not change *errno* in the second case  
 36764 and must set it to a non-zero value in the first case, a zero *errno* after a call returning NULL  
 36765 indicates end of directory; otherwise, an error.

36766 Routines to deal with this problem more directly were proposed:

```
36767 int derror (dirp)
36768 DIR *dirp;
```

```
36769 void clearerr (dirp)
36770 DIR *dirp;
```

36771 The first would indicate whether an error had occurred, and the second would clear the error  
 36772 indication. The simpler method involving *errno* was adopted instead by requiring that *readdir()*  
 36773 not change *errno* when end-of-directory is encountered.

36774 An error or signal indicating that a directory has changed while open was considered but  
 36775 rejected.

36776 The thread-safe version of the directory reading function shall return values in a user-supplied  
 36777 buffer instead of possibly using a static data area that may be overwritten by each call. Either the  
 36778 {NAME\_MAX} compile-time constant or the corresponding *pathconf()* option can be used to  
 36779 determine the maximum sizes of returned path names.

36780 **FUTURE DIRECTIONS**

36781 None.

36782 **SEE ALSO**

36783 *closedir()*, *lstat()*, *opendir()*, *rewinddir()*, *symlink()*, the Base Definitions volume of  
 36784 IEEE Std. 1003.1-200x, <*dirent.h*>, <*sys/types.h*>

36785 **CHANGE HISTORY**

36786 First released in Issue 2.

36787 **Issue 4**

36788 The <*sys/types.h*> header is now marked as optional (OH); this header need not be included on  
 36789 XSI-conformant systems.

36790 In the DESCRIPTION, the fact that XSI-conformant systems return entries for dot and dot-dot is  
 36791 marked as an extension. This functionality is not specified in the ISO POSIX-1 standard.

- 36792 There is some rewording of the DESCRIPTION and RETURN VALUE sections. No functional  
36793 changes are made other than as noted elsewhere in this CHANGE HISTORY section.
- 36794 The following change is incorporated for alignment with the ISO POSIX-1 standard:
- 36795
- The last paragraph of the DESCRIPTION describing a restriction after *fork()* is added.
- 36796 **Issue 4, Version 2**
- 36797 The following changes are incorporated for X/OPEN UNIX conformance:
- 36798
- A statement is added to the DESCRIPTION indicating the disposition of certain fields in  
36799 **struct dirent** when an entry refers to a symbolic link.
- 36800
- The [ENOENT] optional error condition is added.
- 36801 **Issue 5**
- 36802 Large File Summit extensions are added.
- 36803 The *readdir\_r()* function is included for alignment with the POSIX Threads Extension.
- 36804 A note indicating that the *readdir()* function need not be reentrant is added to the  
36805 DESCRIPTION.
- 36806 **Issue 6**
- 36807 The *readdir\_r()* function is marked as part of the Thread-Safe Functions option.
- 36808 The Open Group corrigenda item U026/7 has been applied, correcting the prototype for  
36809 *readdir\_r()*.
- 36810 The Open Group corrigenda item U026/8 has been applied, clarifying the wording of the  
36811 successful return for the *readdir\_r()* function.
- 36812 The following new requirements on POSIX implementations derive from alignment with the  
36813 Single UNIX Specification:
- 36814
- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
36815 required for conforming implementations of previous POSIX specifications, it was not  
36816 required for UNIX applications.
- 36817
- A statement is added to the DESCRIPTION indicating the disposition of certain fields in  
36818 **struct dirent** when an entry refers to a symbolic link.
- 36819
- The [EOVERFLOW] mandatory error condition is added. This change is to support large  
36820 files.
- 36821
- The [ENOENT] optional error condition is added.
- 36822 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
36823 its avoidance of possibly using a static data area.
- 36824 The **restrict** keyword is added to the *readdir\_r()* prototype for alignment with the  
36825 ISO/IEC 9899:1999 standard.

36826 **NAME**

36827 readlink — read the contents of a symbolic link

36828 **SYNOPSIS**

36829 #include &lt;unistd.h&gt;

36830 ssize\_t readlink(const char \*restrict path, char \*restrict buf,  
36831 size\_t bufsize);36832 **DESCRIPTION**36833 The *readlink()* function shall place the contents of the symbolic link referred to by *path* in the  
36834 buffer *buf* which has size *bufsize*. If the number of bytes in the symbolic link is less than *bufsize*,  
36835 the contents of the remainder of *buf* are unspecified. If the *buf* argument is not large enough to  
36836 contain the link content, the first *bufsize* bytes shall be placed in *buf*.36837 If the value of *bufsize* is greater than {SSIZE\_MAX}, the result is implementation-defined.36838 **RETURN VALUE**36839 Upon successful completion, *readlink()* shall return the count of bytes placed in the buffer.  
36840 Otherwise, it shall return a value of -1, leave the buffer unchanged, and set *errno* to indicate the  
36841 error.36842 **ERRORS**36843 The *readlink()* function shall fail if:36844 [EACCES] Search permission is denied for a component of the path prefix of *path*.36845 [EINVAL] The *path* argument names a file that is not a symbolic link.

36846 [EIO] An I/O error occurred while reading from the file system.

36847 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
36848 argument.

36849 [ENAMETOOLONG]

36850 The length of the *path* argument exceeds {PATH\_MAX} or a path name  
36851 component is longer than {NAME\_MAX}.36852 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

36853 [ENOTDIR] A component of the path prefix is not a directory.

36854 The *readlink()* function may fail if:

36855 [EACCES] Read permission is denied for the directory.

36856 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
36857 resolution of the *path* argument.

36858 [ENAMETOOLONG]

36859 As a result of encountering a symbolic link in resolution of the *path* argument,  
36860 the length of the substituted path name string exceeded {PATH\_MAX}.

36861 **EXAMPLES**36862 **Reading the Name of a Symbolic Link**

36863 The following example shows how to read the name of a symbolic link named `/modules/pass1`.

```
36864 #include <unistd.h>
36865 char buf[1024];
36866 int len;
36867 ...
36868 if ((len = readlink("/modules/pass1", buf, sizeof(buf)-1)) != -1);
36869 buf[len] = '\0';
```

36870 **APPLICATION USAGE**

36871 Portable applications should not assume that the returned contents of the symbolic link are  
36872 null-terminated.

36873 **RATIONALE**

36874 Since IEEE Std. 1003.1-200x does not require any association of file times with symbolic links,  
36875 there is no requirement that file times be updated by `readlink()`. The type associated with `bufsiz`  
36876 is a `size_t` in order to be consistent with both the ISO C standard and the definition of `read()`.  
36877 The behavior specified for `readlink()` when `bufsiz` is zero represents historical practice. For this  
36878 case, the standard developers considered a change whereby `readlink()` would return the number  
36879 of non-null bytes contained in the symbolic link with the buffer `buf` remaining unchanged;  
36880 however, since the `stat` structure member `st_size` value can be used to determine the size of  
36881 buffer necessary to contain the contents of the symbolic link as returned by `readlink()`, this  
36882 proposal was rejected, and the historical practice retained.

36883 **FUTURE DIRECTIONS**

36884 None.

36885 **SEE ALSO**

36886 `lstat()`, `stat()`, `symlink()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<unistd.h>`

36887 **CHANGE HISTORY**

36888 First released in Issue 4, Version 2.

36889 **Issue 5**

36890 Moved from X/OPEN UNIX extension to BASE.

36891 **Issue 6**

36892 The return type is changed to `ssize_t`, to align with the IEEE P1003.1a draft standard.

36893 The following new requirements on POSIX implementations derive from alignment with the  
36894 Single UNIX Specification:

- 36895 • This function is made mandatory.
- 36896 • In this function it is possible for the return value to exceed the range of the type `ssize_t` (since  
36897 `size_t` has a larger range of positive values than `ssize_t`). A sentence restricting the size of  
36898 the `size_t` object is added to the description to resolve this conflict.

36899 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 36900 • The `[ENAMETOOLONG]` error is restored as an error dependent on `_POSIX_NO_TRUNC`.  
36901 This is since behavior may vary from one file system to another.
- 36902 • The **FUTURE DIRECTIONS** section is changed to None.



- 36903 The following changes were made to align with the IEEE P1003.1a draft standard:
- 36904 • The [ELOOP] optional error condition is added.
- 36905 The **restrict** keyword is added to the *readlink()* prototype for alignment with the
- 36906 ISO/IEC 9899:1999 standard.

36907 **NAME**

36908           readv — vectored read from file

36909 **SYNOPSIS**

36910 xSI       #include <sys/uio.h>

36911           ssize\_t readv(int *fd*, const struct iovec \**iov*, int *iovcnt*);

36912

36913 **DESCRIPTION**

36914           Refer to *read()*.

36915 **NAME**

36916           realloc — memory reallocator

36917 **SYNOPSIS**

36918           #include &lt;stdlib.h&gt;

36919           void \*realloc(void \*ptr, size\_t size);

36920 **DESCRIPTION**

36921 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
36922       conflict between the requirements described here and the ISO C standard is unintentional. This  
36923       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

36924       The *realloc()* function shall change the size of the memory object pointed to by *ptr* to the size  
36925       specified by *size*. The contents of the object shall remain unchanged up to the lesser of the new  
36926       and old sizes. If the new size of the memory object would require movement of the object, the  
36927       space for the previous instantiation of the object is freed. If the new size is larger, the contents of  
36928       the newly allocated portion of the object are unspecified. If *size* is 0 and *ptr* is not a null pointer,  
36929       the object pointed to is freed. If the space cannot be allocated, the object remains unchanged.

36930       If *ptr* is a null pointer, *realloc()* shall behave like *malloc()* for the specified size.

36931       If *ptr* does not match a pointer returned earlier by *calloc()*, *malloc()*, or *realloc()* or if the space has  
36932       previously been deallocated by a call to *free()* or *realloc()*, the behavior is undefined.

36933       The order and contiguity of storage allocated by successive calls to *realloc()* is unspecified. The  
36934       pointer returned if the allocation succeeds is suitably aligned so that it may be assigned to a  
36935       pointer to any type of object and then used to access such an object in the space allocated (until  
36936       the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object  
36937       disjoint from any other object. The pointer returned points to the start (lowest byte address) of  
36938       the allocated space. If the space cannot be allocated, a null pointer shall be returned.

36939 **RETURN VALUE**

36940       Upon successful completion with a *size* not equal to 0, *realloc()* shall return a pointer to the  
36941       (possibly moved) allocated space. If *size* is 0, either a null pointer or a unique pointer that can be  
36942       successfully passed to *free()* shall be returned. If there is not enough available memory, *realloc()*  
36943 cx       shall return a null pointer and set *errno* to [ENOMEM].

36944 **ERRORS**

36945       The *realloc()* function shall fail if:

36946 cx       [ENOMEM]       Insufficient memory is available.

36947 **EXAMPLES**

36948       None.

36949 **APPLICATION USAGE**

36950       None.

36951 **RATIONALE**

36952       None.

36953 **FUTURE DIRECTIONS**

36954       None.

36955 **SEE ALSO**

36956       *calloc()*, *free()*, *malloc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>

36957 **CHANGE HISTORY**

36958 First released in Issue 1. Derived from Issue 1 of the SVID.

36959 **Issue 4**36960 The setting of *errno* and the [ENOMEM] error are marked as extensions.

36961 The APPLICATION USAGE section is removed.

36962 The following changes are incorporated for alignment with the ISO C standard:

- 36963 • The DESCRIPTION is updated to indicate as follows:
  - 36964 — The order and contiguity of storage allocated by successive calls to this function is
  - 36965 unspecified.
  - 36966 — Each allocation yields a pointer to an object disjoint from any other object.
  - 36967 — The returned pointer points to the lowest byte address of the allocation.
- 36968 • The RETURN VALUE section is updated to indicate what is returned if *size* is 0.

36969 **Issue 6**

36970 Extensions beyond the ISO C standard are now marked.

36971 The following new requirements on POSIX implementations derive from alignment with the  
36972 Single UNIX Specification:

- 36973 • In the RETURN VALUE section, if there is not enough available memory, the setting of *errno*
- 36974 to [ENOMEM] is added.
- 36975 • The [ENOMEM] error condition is added.

36976 **NAME**

36977           realpath — resolve a path name

36978 **SYNOPSIS**

36979 XSI       #include &lt;stdlib.h&gt;

36980           char \*realpath(const char \*restrict *file\_name*,  
36981                        char \*restrict *resolved\_name*);

36982

36983 **DESCRIPTION**

36984       The *realpath()* function derives, from the path name pointed to by *file\_name*, an absolute path  
 36985       name that names the same file, whose resolution does not involve '.', '..', or symbolic links.  
 36986       The generated path name is stored as a null-terminated string, up to a maximum of  
 36987       {PATH\_MAX} bytes, in the buffer pointed to by *resolved\_name*.

36988 **RETURN VALUE**

36989       Upon successful completion, *realpath()* shall return a pointer to the resolved name. Otherwise,  
 36990       *realpath()* shall return a null pointer and set *errno* to indicate the error, and the contents of the  
 36991       buffer pointed to by *resolved\_name* are undefined.

36992 **ERRORS**36993       The *realpath()* function shall fail if:

- |       |                |                                                                                              |
|-------|----------------|----------------------------------------------------------------------------------------------|
| 36994 | [EACCES]       | Read or search permission was denied for a component of <i>file_name</i> .                   |
| 36995 | [EINVAL]       | Either the <i>file_name</i> or <i>resolved_name</i> argument is a null pointer.              |
| 36996 | [EIO]          | An error occurred while reading from the file system.                                        |
| 36997 | [ELOOP]        | A loop exists in symbolic links encountered during resolution of the <i>path</i>             |
| 36998 |                | argument.                                                                                    |
| 36999 | [ENAMETOOLONG] |                                                                                              |
| 37000 |                | The length of the <i>file_name</i> argument exceeds {PATH_MAX} or a path name                |
| 37001 |                | component is longer than {NAME_MAX}.                                                         |
| 37002 | [ENOENT]       | A component of <i>file_name</i> does not name an existing file or <i>file_name</i> points to |
| 37003 |                | an empty string.                                                                             |
| 37004 | [ENOTDIR]      | A component of the path prefix is not a directory.                                           |
| 37005 |                | The <i>realpath()</i> function may fail if:                                                  |
| 37006 | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during                               |
| 37007 |                | resolution of the <i>path</i> argument.                                                      |
| 37008 | [ENAMETOOLONG] |                                                                                              |
| 37009 |                | Path name resolution of a symbolic link produced an intermediate result                      |
| 37010 |                | whose length exceeds {PATH_MAX}.                                                             |
| 37011 | [ENOMEM]       | Insufficient storage space is available.                                                     |

37012 **EXAMPLES**37013 **Generating an Absolute Path Name**

37014 The following example generates an absolute path name for the file identified by the *symlinkpath*  
37015 argument. The generated path name is stored in the *actualpath* array.

```
37016 #include <stdlib.h>
37017 ...
37018 char *symlinkpath = "/tmp/symlink/file";
37019 char actualpath [PATH_MAX+1];
37020 char *ptr;

37021 ptr = realpath(symlinkpath, actualpath);
```

37022 **APPLICATION USAGE**

37023 None.

37024 **RATIONALE**

37025 None.

37026 **FUTURE DIRECTIONS**

37027 None.

37028 **SEE ALSO**

37029 *getcwd()*, *sysconf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**stdlib.h**>

37030 **CHANGE HISTORY**

37031 First released in Issue 4, Version 2.

37032 **Issue 5**

37033 Moved from X/OPEN UNIX extension to BASE.

37034 **Issue 6**

37035 The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`. This  
37036 is since behavior may vary from one file system to another.

37037 The **restrict** keyword is added to the *realpath()* prototype for alignment with the  
37038 ISO/IEC 9899:1999 standard.

37039 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
37040 [ELOOP] error condition is added.

37041 **NAME**

37042           recv — receive a message from a connected socket

37043 **SYNOPSIS**

37044           #include &lt;sys/socket.h&gt;

37045           ssize\_t recv(int *socket*, void \**buffer*, size\_t *length*, int *flags*);37046 **DESCRIPTION**

37047           The *recv()* function receives a message from a connection-mode or connectionless-mode socket.  
 37048           It is normally used with connected sockets because it does not permit the application to retrieve  
 37049           the source address of received data.

37050           The *recv()* function takes the following arguments:37051           *socket*           Specifies the socket file descriptor.37052           *buffer*           Points to a buffer where the message should be stored.37053           *length*           Specifies the length in bytes of the buffer pointed to by the *buffer* argument.

37054           *flags*            Specifies the type of message reception. Values of this argument are formed by  
 37055           logically OR'ing zero or more of the following values:

37056                   MSG\_PEEK       Peeks at an incoming message. The data is treated as unread and  
 37057                   the next *recv()* or similar function shall still return this data.

37058                   MSG\_OOB       Requests out-of-band data. The significance and semantics of  
 37059                   out-of-band data are protocol-specific.

37060                   MSG\_WAITALL Requests that the function block until the full amount of data  
 37061                   requested can be returned. The function may return a smaller  
 37062                   amount of data if a signal is caught, if the connection is  
 37063                   terminated, if MSG\_PEEK was specified, or if an error is pending  
 37064                   for the socket.

37065           The *recv()* function shall return the length of the message written to the buffer pointed to by the  
 37066           *buffer* argument. For message-based sockets, such as SOCK\_DGRAM and SOCK\_SEQPACKET,  
 37067           the entire message shall be read in a single operation. If a message is too long to fit in the  
 37068           supplied buffer, and MSG\_PEEK is not set in the *flags* argument, the excess bytes shall be  
 37069           discarded. For stream-based sockets, such as SOCK\_STREAM, message boundaries shall be  
 37070           ignored. In this case, data is returned to the user as soon as it becomes available, and no data  
 37071           shall be discarded.

37072           If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
 37073           message.

37074           If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
 37075           descriptor, *recv()* shall block until a message arrives. If no messages are available at the socket  
 37076           and O\_NONBLOCK is set on the socket's file descriptor, *recv()* shall fail and set *errno* to  
 37077           [EAGAIN] or [EWOULDBLOCK].

37078 **RETURN VALUE**

37079           Upon successful completion, *recv()* shall return the length of the message in bytes. If no  
 37080           messages are available to be received and the peer has performed an orderly shutdown, *recv()*  
 37081           shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

37082 **ERRORS**

- 37083       The *recv()* function shall fail if:
- 37084       [EAGAIN] or [EWOULDBLOCK]
- 37085               The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
37086               to be received; or MSG\_OOB is set and no out-of-band data is available and  
37087               either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
37088               not support blocking to await out-of-band data.
- 37089       [EBADF]       The *socket* argument is not a valid file descriptor.
- 37090       [ECONNRESET] A connection was forcibly closed by a peer.
- 37091       [EINTR]       The *recv()* function was interrupted by a signal that was caught, before any  
37092               data was available.
- 37093       [EINVAL]       The MSG\_OOB flag is set and no out-of-band data is available.
- 37094       [ENOTCONN]    A receive is attempted on a connection-mode socket that is not connected.
- 37095       [ENOTSOCK]    The *socket* argument does not refer to a socket.
- 37096       [EOPNOTSUPP] The specified flags are not supported for this socket type or protocol.
- 37097       [ETIMEDOUT]    The connection timed out during connection establishment, or due to a  
37098               transmission timeout on active connection.
- 37099       The *recv()* function may fail if:
- 37100       [EIO]         An I/O error occurred while reading from or writing to the file system.
- 37101       [ENOBUFS]     Insufficient resources were available in the system to perform the operation.
- 37102       [ENOMEM]     Insufficient memory was available to fulfill the request.

37103 **EXAMPLES**

37104       None.

37105 **APPLICATION USAGE**

37106       The *recv()* function is identical to *recvfrom()* with a zero *address\_len* argument, and to *read()* if no  
37107       flags are used.

37108       The *select()* and *poll()* functions can be used to determine when data is available to be received.

37109 **RATIONALE**

37110       None.

37111 **FUTURE DIRECTIONS**

37112       None.

37113 **SEE ALSO**

37114       *poll()*, *read()*, *recvmsg()*, *recvfrom()*, *select()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*,  
37115       *write()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/socket.h>

37116 **CHANGE HISTORY**

37117       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.



37118 **NAME**

37119           recvfrom — receive a message from a socket

37120 **SYNOPSIS**

```
37121 #include <sys/socket.h>
37122 ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
37123 int flags, struct sockaddr *restrict address,
37124 socklen_t *restrict address_len);
```

37125 **DESCRIPTION**

37126           The *recvfrom()* function receives a message from a connection-mode or connectionless-mode  
 37127           socket. It is normally used with connectionless-mode sockets because it permits the application  
 37128           to retrieve the source address of received data.

37129           The *recvfrom()* function takes the following arguments:

|       |                    |                                                                                                                                                                                                                                                                         |
|-------|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 37130 | <i>socket</i>      | Specifies the socket file descriptor.                                                                                                                                                                                                                                   |
| 37131 | <i>buffer</i>      | Points to the buffer where the message should be stored.                                                                                                                                                                                                                |
| 37132 | <i>length</i>      | Specifies the length in bytes of the buffer pointed to by the <i>buffer</i> argument.                                                                                                                                                                                   |
| 37133 | <i>flags</i>       | Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:                                                                                                                                   |
| 37134 |                    |                                                                                                                                                                                                                                                                         |
| 37135 | MSG_PEEK           | Peeks at an incoming message. The data is treated as unread and the next <i>recvfrom()</i> or similar function shall still return this data.                                                                                                                            |
| 37136 |                    |                                                                                                                                                                                                                                                                         |
| 37137 |                    |                                                                                                                                                                                                                                                                         |
| 37138 | MSG_OOB            | Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.                                                                                                                                                                    |
| 37139 |                    |                                                                                                                                                                                                                                                                         |
| 37140 | MSG_WAITALL        | Requests that the function block until the full amount of data requested can be returned. The function may return a smaller amount of data if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket. |
| 37141 |                    |                                                                                                                                                                                                                                                                         |
| 37142 |                    |                                                                                                                                                                                                                                                                         |
| 37143 |                    |                                                                                                                                                                                                                                                                         |
| 37144 |                    |                                                                                                                                                                                                                                                                         |
| 37145 | <i>address</i>     | A null pointer, or points to a <b>sockaddr</b> structure in which the sending address is to be stored. The length and format of the address depend on the address family of the socket.                                                                                 |
| 37146 |                    |                                                                                                                                                                                                                                                                         |
| 37147 |                    |                                                                                                                                                                                                                                                                         |
| 37148 | <i>address_len</i> | Specifies the length of the <b>sockaddr</b> structure pointed to by the <i>address</i> argument.                                                                                                                                                                        |
| 37149 |                    |                                                                                                                                                                                                                                                                         |

37150           The *recvfrom()* function shall return the length of the message written to the buffer pointed to by  
 37151           the *buffer* argument. For message-based sockets, such as SOCK\_DGRAM and SOCK\_SEQPACKET, the entire message shall be read in a single operation. If a message is too  
 37152           long to fit in the supplied buffer, and MSG\_PEEK is not set in the *flags* argument, the excess  
 37153           bytes shall be discarded. For stream-based sockets, such as SOCK\_STREAM, message  
 37154           boundaries shall be ignored. In this case, data is returned to the user as soon as it becomes  
 37155           available, and no data shall be discarded.

37157           If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
 37158           message.

37159           Not all protocols provide the source address for messages. If the *address* argument is not a null  
 37160           pointer and the protocol provides the source address of messages, the source address of the  
 37161           received message is stored in the **sockaddr** structure pointed to by the *address* argument, and the

- 37162 length of this address is stored in the object pointed to by the *address\_len* argument.
- 37163 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
37164 the stored address shall be truncated.
- 37165 If the *address* argument is not a null pointer and the protocol does not provide the source address  
37166 of messages, the value stored in the object pointed to by *address* is unspecified.
- 37167 If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
37168 descriptor, *recvfrom()* blocks until a message arrives. If no messages are available at the socket  
37169 and O\_NONBLOCK is set on the socket's file descriptor, *recvfrom()* shall fail and set *errno* to  
37170 [EAGAIN] or [EWOULDBLOCK].
- 37171 **RETURN VALUE**
- 37172 Upon successful completion, *recvfrom()* shall return the length of the message in bytes. If no  
37173 messages are available to be received and the peer has performed an orderly shutdown,  
37174 *recvfrom()* shall return 0. Otherwise, the function shall return -1 and set *errno* to indicate the  
37175 error.
- 37176 **ERRORS**
- 37177 The *recvfrom()* function shall fail if:
- 37178 [EAGAIN] or [EWOULDBLOCK] The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
37179 to be received; or MSG\_OOB is set and no out-of-band data is available and  
37180 either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
37181 not support blocking to await out-of-band data.  
37182
- 37183 [EBADF] The *socket* argument is not a valid file descriptor.
- 37184 [ECONNRESET] A connection was forcibly closed by a peer.
- 37185 [EINTR] A signal interrupted *recvfrom()* before any data was available.
- 37186 [EINVAL] The MSG\_OOB flag is set and no out-of-band data is available.
- 37187 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.
- 37188 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 37189 [EOPNOTSUPP] The specified flags are not supported for this socket type.
- 37190 [ETIMEDOUT] The connection timed out during connection establishment, or due to a  
37191 transmission timeout on active connection.
- 37192 The *recvfrom()* function may fail if:
- 37193 [EIO] An I/O error occurred while reading from or writing to the file system.
- 37194 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 37195 [ENOMEM] Insufficient memory was available to fulfill the request.

37196 **EXAMPLES**

37197 None.

37198 **APPLICATION USAGE**37199 The *select()* and *poll()* functions can be used to determine when data is available to be received.37200 **RATIONALE**

37201 None.

37202 **FUTURE DIRECTIONS**

37203 None.

37204 **SEE ALSO**37205 *poll()*, *read()*, *recv()*, *recvmsg()*, *select()* (on page 1753)1 *send()*, *sendmsg()*, *sendto()*, *shutdown()*,  
37206 *socket()*, *write()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/socket.h>37207 **CHANGE HISTORY**

37208 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

## 37209 NAME

37210 recvmsg — receive a message from a socket

## 37211 SYNOPSIS

37212 #include &lt;sys/socket.h&gt;

37213 ssize\_t recvmsg(int *socket*, struct msghdr \**message*, int *flags*);

## 37214 DESCRIPTION

37215 The *recvmsg()* function receives a message from a connection-mode or connectionless-mode  
 37216 socket. It is normally used with connectionless-mode sockets because it permits the application  
 37217 to retrieve the source address of received data.

37218 The *recvmsg()* function takes the following arguments:

|       |                |                                                                                                                                                                                                                                                                                                                      |
|-------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 37219 | <i>socket</i>  | Specifies the socket file descriptor.                                                                                                                                                                                                                                                                                |
| 37220 | <i>message</i> | Points to a <b>msghdr</b> structure, containing both the buffer to store the source address and the buffers for the incoming message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored on input, but may contain meaningful values on output. |
| 37221 |                |                                                                                                                                                                                                                                                                                                                      |
| 37222 |                |                                                                                                                                                                                                                                                                                                                      |
| 37223 |                |                                                                                                                                                                                                                                                                                                                      |
| 37224 | <i>flags</i>   | Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:                                                                                                                                                                                |
| 37225 |                |                                                                                                                                                                                                                                                                                                                      |
| 37226 | MSG_OOB        | Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.                                                                                                                                                                                                                 |
| 37227 |                |                                                                                                                                                                                                                                                                                                                      |
| 37228 | MSG_PEEK       | Peeks at the incoming message.                                                                                                                                                                                                                                                                                       |
| 37229 | MSG_WAITALL    | Requests that the function block until the full amount of data requested can be returned. The function may return a smaller amount of data if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.                                              |
| 37230 |                |                                                                                                                                                                                                                                                                                                                      |
| 37231 |                |                                                                                                                                                                                                                                                                                                                      |
| 37232 |                |                                                                                                                                                                                                                                                                                                                      |
| 37233 |                |                                                                                                                                                                                                                                                                                                                      |

37234 The *recvmsg()* function receives messages from unconnected or connected sockets and shall  
 37235 return the length of the message.

37236 The *recvmsg()* function shall return the total length of the message. For message-based sockets,  
 37237 such as SOCK\_DGRAM and SOCK\_SEQPACKET, the entire message shall be read in a single  
 37238 operation. If a message is too long to fit in the supplied buffers, and MSG\_PEEK is not set in the  
 37239 *flags* argument, the excess bytes shall be discarded, and MSG\_TRUNC is set in the *msg\_flags*  
 37240 member of the **msghdr** structure. For stream-based sockets, such as SOCK\_STREAM, message  
 37241 boundaries shall be ignored. In this case, data is returned to the user as soon as it becomes  
 37242 available, and no data shall be discarded.

37243 If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
 37244 message.

37245 If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
 37246 descriptor, *recvmsg()* shall block until a message arrives. If no messages are available at the  
 37247 socket and O\_NONBLOCK is set on the socket's file descriptor, *recvmsg()* function shall fail and  
 37248 set *errno* to [EAGAIN] or [EWOULDBLOCK].

37249 In the **msghdr** structure, the *msg\_name* and *msg\_namelen* members specify the source address if  
 37250 the socket is unconnected. If the socket is connected, the *msg\_name* and *msg\_namelen* members  
 37251 are ignored. The *msg\_name* member may be a null pointer if no names are desired or required.  
 37252 The *msg\_iov* and *msg\_iovlen* fields are used to specify where the received data shall be stored.  
 37253 *msg\_iov* points to an array of **iovec** structures; *msg\_iovlen* shall be set to the dimension of this

37254 array. In each **iovec** structure, the *iov\_base* field specifies a storage area and the *iov\_len* field gives  
 37255 its size in bytes. Each storage area indicated by *msg\_iov* is filled with received data in turn until  
 37256 all of the received data is stored or all of the areas have been filled.

37257 Upon successful completion, the *msg\_flags* member of the message header is the bitwise-  
 37258 inclusive OR of all of the following flags that indicate conditions detected for the received  
 37259 message:

37260 MSG\_EOR End of record was received (if supported by the protocol).

37261 MSG\_OOB Out-of-band data was received.

37262 MSG\_TRUNC Normal data was truncated.

37263 MSG\_CTRUNC Control data was truncated.

#### 37264 RETURN VALUE

37265 Upon successful completion, *recvmsg()* shall return the length of the message in bytes. If no  
 37266 messages are available to be received and the peer has performed an orderly shutdown,  
 37267 *recvmsg()* shall return 0. Otherwise,  $-1$  shall be returned and *errno* set to indicate the error.

#### 37268 ERRORS

37269 The *recvmsg()* function shall fail if:

37270 [EAGAIN] or [EWOULDBLOCK]

37271 The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
 37272 to be received; or MSG\_OOB is set and no out-of-band data is available and  
 37273 either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
 37274 not support blocking to await out-of-band data.

37275 [EBADF] The *socket* argument is not a valid open file descriptor.

37276 [ECONNRESET] A connection was forcibly closed by a peer.

37277 [EINTR] This function was interrupted by a signal before any data was available.

37278 [EINVAL] The sum of the *iov\_len* values overflows a **ssize\_t**, or the MSG\_OOB flag is set  
 37279 and no out-of-band data is available.

37280 [EMSGSIZE] The *msg\_iovlen* member of the **msg\_hdr** structure pointed to by *message* is less  
 37281 than or equal to 0, or is greater than {IOV\_MAX}.

37282 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

37283 [ENOTSOCK] The *socket* argument does not refer to a socket.

37284 [EOPNOTSUPP] The specified flags are not supported for this socket type.

37285 [ETIMEDOUT] The connection timed out during connection establishment, or due to a  
 37286 transmission timeout on active connection.

37287 The *recvmsg()* function may fail if:

37288 [EIO] An I/O error occurred while reading from or writing to the file system.

37289 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

37290 [ENOMEM] Insufficient memory was available to fulfill the request.

37291 **EXAMPLES**

37292           None.

37293 **APPLICATION USAGE**37294           The *select()* and *poll()* functions can be used to determine when data is available to be received.37295 **RATIONALE**

37296           None.

37297 **FUTURE DIRECTIONS**

37298           None.

37299 **SEE ALSO**37300           *poll()*, *recv()*, *recvfrom()*, *select()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*, the Base |

37301           Definitions volume of IEEE Std. 1003.1-200x, &lt;sys/socket.h&gt; |

37302 **CHANGE HISTORY**

37303           First released in Issue 6. Derived from the XNS, Issue 5.2 specification. |

## 37304 NAME

37305 regcomp, regerror, regex, regfree — regular expression matching

## 37306 SYNOPSIS

37307 #include &lt;regex.h&gt;

```

37308 int regcomp(regex_t *restrict preg, const char *restrict pattern, int cflags);
37309 size_t regerror(int errcode, const regex_t *restrict preg,
37310 char *restrict errbuf, size_t errbuf_size);
37311 int regex(const regex_t *restrict preg, const char *restrict string,
37312 size_t nmatch, regmatch_t pmatch[restrict], int eflags);
37313 void regfree(regex_t *preg);

```

## 37314 DESCRIPTION

37315 These functions interpret *basic* and *extended* regular expressions as described in the Base  
 37316 Definitions volume of IEEE Std. 1003.1-200x, Chapter 9, Regular Expressions.

37317 The `regex_t` structure contains at least the following member:

37318

37319

37320

| Member Type | Member Name | Description                             |
|-------------|-------------|-----------------------------------------|
| size_t      | re_nsub     | Number of parenthesized subexpressions. |

37321 The `regmatch_t` structure contains at least the following members:

37322

37323

37324

37325

37326

| Member Type | Member Name | Description                                                                                |
|-------------|-------------|--------------------------------------------------------------------------------------------|
| regoff_t    | rm_so       | Byte offset from start of <i>string</i> to start of substring.                             |
| regoff_t    | rm_eo       | Byte offset from start of <i>string</i> of the first character after the end of substring. |

37327 The `regcomp()` function shall compile the regular expression contained in the string pointed to by  
 37328 the *pattern* argument and places the results in the structure pointed to by *preg*. The *cflags*  
 37329 argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in  
 37330 the header <regex.h>:

37331 REG\_EXTENDED Use Extended Regular Expressions.

37332 REG\_ICASE Ignore case in match. (See the Base Definitions volume of  
 37333 IEEE Std. 1003.1-200x, Chapter 9, Regular Expressions.)

37334 REG\_NOSUB Report only success/fail in `regex()`.

37335 REG\_NEWLINE Change the handling of <newline> characters, as described in the text.

37336 The default regular expression type for *pattern* is a Basic Regular Expression. The application can  
 37337 specify Extended Regular Expressions using the REG\_EXTENDED *cflags* flag.

37338 If the REG\_NOSUB flag was not set in *cflags*, then `regcomp()` shall set *re\_nsub* to the number of  
 37339 parenthesized subexpressions (delimited by "\(\)" in basic regular expressions or "( )" in  
 37340 extended regular expressions) found in *pattern*.

37341 The `regex()` function compares the null-terminated string specified by *string* with the compiled  
 37342 regular expression *preg* initialized by a previous call to `regcomp()`. If it finds a match, `regex()`  
 37343 shall return 0; otherwise, it shall return non-zero indicating either no match or an error. The  
 37344 *eflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are  
 37345 defined in the header <regex.h>:

37346 REG\_NOTBOL The first character of the string pointed to by *string* is not the beginning of the  
 37347 line. Therefore, the circumflex character ('^'), when taken as a special  
 37348 character, shall not match the beginning of *string*.

37349 REG\_NOTEOL The last character of the string pointed to by *string* is not the end of the line.  
 37350 Therefore, the dollar sign ('\$'), when taken as a special character, shall not  
 37351 match the end of *string*.

37352 If *nmatch* is 0 or REG\_NOSUB was set in the *flags* argument to *regcomp()*, then *regexec()* shall  
 37353 ignore the *pmatch* argument. Otherwise, the application shall ensure that the *pmatch* argument  
 37354 points to an array with at least *nmatch* elements, and *regexec()* shall fill in the elements of that  
 37355 array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions  
 37356 of *pattern*: *pmatch[i].rm\_so* shall be the byte offset of the beginning and *pmatch[i].rm\_eo* shall be  
 37357 one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th  
 37358 matched open parenthesis, counting from 1.) Offsets in *pmatch[0]* identify the substring that  
 37359 corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch[nmatch-1]*  
 37360 shall be filled with -1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself  
 37361 counts as a subexpression), then *regexec()* shall still do the match, but shall record only the first  
 37362 *nmatch* substrings.

37363 When matching a basic or extended regular expression, any given parenthesized subexpression  
 37364 of *pattern* might participate in the match of several different substrings of *string*, or it might not  
 37365 match any substring even though the pattern as a whole did match. The following rules are used  
 37366 to determine which substrings to report in *pmatch* when matching regular expressions:

37367 1. If subexpression *i* in a regular expression is not contained within another subexpression,  
 37368 and it participated in the match several times, then the byte offsets in *pmatch[i]* shall  
 37369 delimit the last such match.

37370 2. If subexpression *i* is not contained within another subexpression, and it did not participate  
 37371 in an otherwise successful match, the byte offsets in *pmatch[i]* shall be -1. A subexpression  
 37372 does not participate in the match when:

37373 ' \* ' or "\{\}" appears immediately after the subexpression in a basic regular  
 37374 expression, or ' \* ', ' ? ', or "{ }" appears immediately after the subexpression in an  
 37375 extended regular expression, and the subexpression did not match (matched 0 times)

37376 or:

37377 ' | ' is used in an extended regular expression to select this subexpression or another,  
 37378 and the other subexpression matched.

37379 3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained  
 37380 within any other subexpression that is contained within *j*, and a match of subexpression *j*  
 37381 is reported in *pmatch[j]*, then the match or non-match of subexpression *i* reported in  
 37382 *pmatch[i]* shall be as described in 1. and 2. above, but within the substring reported in  
 37383 *pmatch[j]* rather than the whole string. The offsets in *pmatch[i]* are still relative to the start  
 37384 of string.



37385 **Notes to Reviewers**37386 *This section with side shading will not appear in the final copy. - Ed.*37387 D1, XSH, ERN 283 proposes changing “but within” above to “but describing the substring  
37388 found within the substring reported in *pmatch[j]* rather than the whole string. The byte  
37389 offsets are relative to the whole string.”37390 4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch[j]* are  $-1$ ,  
37391 then the pointers in *pmatch[i]* shall also be  $-1$ .37392 5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch[i]* shall be  
37393 the byte offset of the character or null terminator immediately following the zero-length  
37394 string.37395 If, when *regexec()* is called, the locale is different from when the regular expression was  
37396 compiled, the result is undefined.37397 If REG\_NEWLINE is not set in *cflags*, then a <newline> character in *pattern* or *string* shall be  
37398 treated as an ordinary character. If REG\_NEWLINE is set, then <newline> shall be treated as an  
37399 ordinary character except as follows:37400 1. A <newline> character in *string* shall not be matched by a period outside a bracket  
37401 expression or by any form of a non-matching list (see the Base Definitions volume of  
37402 IEEE Std. 1003.1-200x, Chapter 9, Regular Expressions).37403 2. A circumflex ('^') in *pattern*, when used to specify expression anchoring (see the Base  
37404 Definitions volume of IEEE Std. 1003.1-200x, Section 9.3.8, BRE Expression Anchoring),  
37405 shall match the zero-length string immediately after a <newline> in *string*, regardless of  
37406 the setting of REG\_NOTBOL.37407 3. A dollar sign ('\$') in *pattern*, when used to specify expression anchoring, shall match the  
37408 zero-length string immediately before a <newline> in *string*, regardless of the setting of  
37409 REG\_NOTEOL.37410 The *regfree()* function frees any memory allocated by *regcomp()* associated with *preg*.

37411 The following constants are defined as error return values:

37412 REG\_NOMATCH *regexec()* failed to match.

37413 REG\_BADPAT Invalid regular expression.

37414 REG\_ECOLLATE Invalid collating element referenced.

37415 REG\_ECTYPE Invalid character class type referenced.

37416 REG\_EESCAPE Trailing '\\' in pattern.

37417 REG\_ESUBREG Number in "\digit" invalid or in error.

37418 REG\_EBRACK "[ ]" imbalance.

37419 REG\_EPAREN "\(\)" or "()" imbalance.

37420 REG\_EBRACE "\{\}" imbalance.

37421 REG\_BADBR Content of "\{\}" invalid: not a number, number too large, more than  
37422 two numbers, first larger than second.

37423 REG\_ERANGE Invalid endpoint in range expression.

37424 REG\_ESPACE Out of memory.

37425 REG\_BADRPT '?', '\*', or '+' not preceded by valid regular expression.

37426 The *regerror()* function provides a mapping from error codes returned by *regcomp()* and  
 37427 *regexec()* to unspecified printable strings. It generates a string corresponding to the value of the  
 37428 *errcode* argument, which the application shall ensure is the last non-zero value returned by  
 37429 *regcomp()* or *regexec()* with the given value of *preg*. If *errcode* is not such a value, the content of  
 37430 the generated string is unspecified.

37431 If *preg* is a null pointer, but *errcode* is a value returned by a previous call to *regexec()* or *regcomp()*,  
 37432 the *regerror()* still generates an error string corresponding to the value of *errcode*, but it might not  
 37433 be as detailed under some implementations.

37434 If the *errbuf\_size* argument is not 0, *regerror()* shall place the generated string into the buffer of  
 37435 size *errbuf\_size* bytes pointed to by *errbuf*. If the string (including the terminating null) cannot fit  
 37436 in the buffer, *regerror()* shall truncate the string and null-terminates the result.

37437 If *errbuf\_size* is 0, *regerror()* shall ignore the *errbuf* argument, and return the size of the buffer  
 37438 needed to hold the generated string.

37439 If the *preg* argument to *regexec()* or *regfree()* is not a compiled regular expression returned by  
 37440 *regcomp()*, the result is undefined. A *preg* is no longer treated as a compiled regular expression  
 37441 after it is given to *regfree()*.

37442 **RETURN VALUE**

37443 Upon successful completion, the *regcomp()* function shall return 0. Otherwise, it shall return an  
 37444 integer value indicating an error as described in <regex.h>, and the content of *preg* is undefined.  
 37445 If a code is returned, the interpretation shall be as given in <regex.h>.

37446 If *regcomp()* detects an invalid RE, it may return REG\_BADPAT, or it may return one of the error  
 37447 codes that more precisely describes the error.

37448 Upon successful completion, the *regexec()* function shall return 0. Otherwise, it shall return  
 37449 REG\_NOMATCH to indicate no match.

37450 Upon successful completion, the *regerror()* function shall return the number of bytes needed to  
 37451 hold the entire generated string, including the null termination. If the return value is greater than  
 37452 *errbuf\_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

37453 The *regfree()* function shall return no value.

37454 **ERRORS**

37455 No errors are defined.

37456 **EXAMPLES**

```

37457 #include <regex.h>
37458 /*
37459 * Match string against the extended regular expression in
37460 * pattern, treating errors as no match.
37461 *
37462 * Return 1 for match, 0 for no match.
37463 */
37464 int
37465 match(const char *string, char *pattern)
37466 {
37467 int status;
37468 regex_t re;
```

```

37469 if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
37470 return(0); /* Report error. */
37471 }
37472 status = regexec(&re, string, (size_t) 0, NULL, 0);
37473 regfree(&re);
37474 if (status != 0) {
37475 return(0); /* Report error. */
37476 }
37477 return(1);
37478 }

```

37479 The following demonstrates how the REG\_NOTBOL flag could be used with *regexec()* to find all  
37480 substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very  
37481 little error checking is done.)

```

37482 (void) regcomp (&re, pattern, 0);
37483 /* This call to regexec() finds the first match on the line. */
37484 error = regexec (&re, &buffer[0], 1, &pm, 0);
37485 while (error == 0) { /* While matches found. */
37486 /* Substring found between pm.rm_so and pm.rm_eo. */
37487 /* This call to regexec() finds the next match. */
37488 error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
37489 }

```

#### 37490 APPLICATION USAGE

37491 An application could use:

```

37492 regerror(code, preg, (char *)NULL, (size_t)0)

```

37493 to find out how big a buffer is needed for the generated string, *malloc()* a buffer to hold the  
37494 string, and then call *regerror()* again to get the string. Alternatively, it could allocate a fixed,  
37495 static buffer that is big enough to hold most strings, and then use *malloc()* to allocate a larger  
37496 buffer if it finds that this is too small.

37497 To match a pattern as described in the Shell and Utilities volume of IEEE Std. 1003.1-200x,  
37498 Section 2.14, Pattern Matching Notation, use the *fnmatch()* function.

#### 37499 RATIONALE

37500 The *regmatch()* function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are  
37501 supplied by the application, even if some elements of *pmatch* do not correspond to  
37502 subexpressions in *pattern*. The application writer should note that there is probably no reason  
37503 for using a value of *nmatch* that is larger than *preg->re\_nsub+1*.

37504 The REG\_NEWLINE flag supports a use of RE matching that is needed in some applications like  
37505 text editors. In such applications, the user supplies an RE asking the application to find a line  
37506 that matches the given expression. An anchor in such an RE anchors at the beginning or end of  
37507 any line. Such an application can pass a sequence of <newline>-separated lines to *regexec()* as a  
37508 single long string and specify REG\_NEWLINE to *regcomp()* to get the desired behavior. The  
37509 application must ensure that there are no explicit <newline>s in *pattern* if it wants to ensure that  
37510 any match occurs entirely within a single line.

37511 The REG\_NEWLINE flag affects the behavior of *regexec()*, but it is in the *cflags* parameter to  
37512 *regcomp()* to allow flexibility of implementation. Some implementations will want to generate  
37513 the same compiled RE in *regcomp()* regardless of the setting of REG\_NEWLINE and have  
37514 *regexec()* handle anchors differently based on the setting of the flag. Other implementations will  
37515 generate different compiled REs based on the REG\_NEWLINE.

37516 The REG\_ICASE flag supports the operations taken by the *grep -i* option and the historical  
37517 implementations of *ex* and *vi*. Including this flag will make it easier for application code to be  
37518 written that does the same thing as these utilities.

37519 The substrings reported in *pmatch*[] are defined using offsets from the start of the string rather  
37520 than pointers. Since this is a new interface, there should be no impact on historical  
37521 implementations or applications, and offsets should be just as easy to use as pointers. The  
37522 change to offsets was made to facilitate future extensions in which the string to be searched is  
37523 presented to *regexexec()* in blocks, allowing a string to be searched that is not all in memory at  
37524 once.

37525 A new type **regoff\_t** is used for the elements of *pmatch*[] to ensure that the application can  
37526 represent either the largest possible array in memory (important for an application conforming  
37527 to the Shell and Utilities volume of IEEE Std. 1003.1-200x) or the largest possible file (important  
37528 for an application using the extension where a file is searched in chunks).

37529 The standard developers rejected the inclusion of a *regsub()* function that would be used to do  
37530 substitutions for a matched RE. While such a routine would be useful to some applications, its  
37531 utility would be much more limited than the matching function described here. Both RE parsing  
37532 and substitution are possible to implement without support other than that required by the  
37533 ISO C standard, but matching is much more complex than substituting. The only difficult part of  
37534 substitution, given the information supplied by *regexexec()*, is finding the next character in a string  
37535 when there can be multibyte characters. That is a much larger issue, and one that needs a more  
37536 general solution.

37537 The *errno* variable has not been used for error returns to avoid filling the *errno* name space for  
37538 this feature.

37539 The interface is defined so that the matched substrings *rm\_sp* and *rm\_ep* are in a separate  
37540 **regmatch\_t** structure instead of in **regex\_t**. This allows a single compiled RE to be used  
37541 simultaneously in several contexts; in *main()* and a signal handler, perhaps, or in multiple  
37542 threads of lightweight processes. (The *preg* argument to *regexexec()* is declared with type **const**, so  
37543 the implementation is not permitted to use the structure to store intermediate results.) It also  
37544 allows an application to request an arbitrary number of substrings from an RE. The number of  
37545 subexpressions in the RE is reported in *re\_nsub* in *preg*. With this change to *regexexec()*,  
37546 consideration was given to dropping the REG\_NOSUB flag since the user can now specify this  
37547 with a zero *nmatch* argument to *regexexec()*. However, keeping REG\_NOSUB allows an  
37548 implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp()*  
37549 that no subexpressions need be reported. The implementation is only required to fill in *pmatch* if  
37550 *nmatch* is not zero and if REG\_NOSUB is not specified. Note that the **size\_t** type, as defined in  
37551 the ISO C standard, is unsigned, so the description of *regexexec()* does not need to address  
37552 negative values of *nmatch*.

37553 REG\_NOTBOL was added to allow an application to do repeated searches for the same pattern  
37554 in a line. If the pattern contains a circumflex character that should match the beginning of a line,  
37555 then the pattern should only match when matched against the beginning of the line. Without  
37556 the REG\_NOTBOL flag, the application could rewrite the expression for subsequent matches,  
37557 but in the general case this would require parsing the expression. The need for REG\_NOTEOL is  
37558 not as clear; it was added for symmetry.

37559 The addition of the *regerror()* function addresses the historical need for portable application  
37560 programs to have access to error information more than “Function failed to compile/match your  
37561 RE for unknown reasons”.

37562 This interface provides for two different methods of dealing with error conditions. The specific  
37563 error codes (REG\_EBRACE, for example), defined in **<regex.h>**, allow an application to recover

37564 from an error if it is so able. Many applications, especially those that use patterns supplied by a  
 37565 user, will not try to deal with specific error cases, but will just use *regerror()* to obtain a human-  
 37566 readable error message to present to the user.

37567 The *regerror()* function uses a scheme similar to *confstr()* to deal with the problem of allocating  
 37568 memory to hold the generated string. The scheme used by *strerror()* in the ISO C standard was  
 37569 considered unacceptable since it creates difficulties for multi-threaded applications.

37570 The *preg* argument is provided to *regerror()* to allow an implementation to generate a more  
 37571 descriptive message than would be possible with *errcode* alone. An implementation might, for  
 37572 example, save the character offset of the offending character of the pattern in a field of *preg*, and  
 37573 then include that in the generated message string. The implementation may also ignore *preg*.

37574 A REG\_FILENAME flag was considered, but omitted. This flag caused *regexec()* to match  
 37575 patterns as described in the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.14,  
 37576 Pattern Matching Notation instead of REs. This service is now provided by the *fnmatch()*  
 37577 function.

37578 Notice that there is a difference in philosophy between the ISO POSIX-2:1993 standard and  
 37579 IEEE Std. 1003.1-200x in how to handle a bad regular expression. The ISO POSIX-2:1993  
 37580 standard says that many bad constructs produce undefined results, or that the interpretation is  
 37581 undefined. IEEE Std. 1003.1-200x, however, says that the interpretation of such REs is  
 37582 unspecified. The term “undefined” means that the action by the application is an error, of  
 37583 similar severity to passing a bad pointer to a function.

37584 The *regcomp()* and *regexec()* functions are required to accept any null-terminated string as the  
 37585 *pattern* argument. If the meaning of the string is undefined, the behavior of the function is  
 37586 unspecified. IEEE Std. 1003.1-200x does not specify how the functions will interpret the pattern;  
 37587 they might return error codes, or they might do pattern matching in some completely  
 37588 unexpected way, but they should not do something like abort the process.

#### 37589 FUTURE DIRECTIONS

37590 None.

#### 37591 SEE ALSO

37592 *fnmatch()*, *glob()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<regex.h>`,  
 37593 `<sys/types.h>`

#### 37594 CHANGE HISTORY

37595 First released in Issue 4. Derived from the ISO POSIX-2 standard.

#### 37596 Issue 5

37597 Moved from POSIX2 C-language Binding to BASE.

#### 37598 Issue 6

37599 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

37600 The following new requirements on POSIX implementations derive from alignment with the  
 37601 Single UNIX Specification:

- 37602 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 37603 required for conforming implementations of previous POSIX specifications, it was not  
 37604 required for UNIX applications.

37605 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

37606 The REG\_ENOSYS constant is removed.

37607 The **restrict** keyword is added to the *regcomp()*, *regerror()*, and *regexec()* prototypes for  
 37608 alignment with the ISO/IEC 9899:1999 standard.



37609 **NAME**

37610 remainder, remainderf, remainderl — remainder function

37611 **SYNOPSIS**37612 XSI `#include <math.h>`37613 `double remainder(double x, double y);`37614 `float remainderf(float x, float y);`37615 `long double remainderl(long double x, long double y);`

37616

37617 **DESCRIPTION**

37618 These functions shall return the floating point remainder  $r=x-ny$  when  $y$  is non-zero. The value  
 37619  $n$  is the integral value nearest the exact value  $x/y$ . When  $|n-x/y|=1/2$ , the value  $n$  is chosen to be  
 37620 even.

37621 The behavior of *remainder()* is independent of the rounding mode.37622 **RETURN VALUE**37623 These functions shall return the floating point remainder  $r=x-ny$  when  $y$  is non-zero.37624 When  $y$  is 0, *remainder()* shall return NaN (or equivalent if available) and set *errno* to [EDOM].37625 If the value of  $x$  is  $\pm\text{Inf}$ , *remainder()* shall return NaN and set *errno* to [EDOM].37626 If  $x$  or  $y$  is NaN, then the function shall return NaN and *errno* may be set to [EDOM].37627 **ERRORS**

37628 These functions shall fail if:

37629 [EDOM] The  $y$  argument is 0 or the  $x$  argument is positive or negative infinity.

37630 These functions may fail if:

37631 [EDOM] The  $x$  or  $y$  argument is NaN.37632 **EXAMPLES**

37633 None.

37634 **APPLICATION USAGE**

37635 None.

37636 **RATIONALE**

37637 None.

37638 **FUTURE DIRECTIONS**

37639 None.

37640 **SEE ALSO**37641 *abs()*, *div()*, *ldiv()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<math.h>`37642 **CHANGE HISTORY**

37643 First released in Issue 4, Version 2.

37644 **Issue 5**

37645 Moved from X/OPEN UNIX extension to BASE.

37646 **Issue 6**37647 The *remainderf()* and *remainderl()* functions are added for alignment with the ISO/IEC 9899:1999  
 37648 standard.

37649 **NAME**

37650 remove — remove a file

37651 **SYNOPSIS**

37652 #include &lt;stdio.h&gt;

37653 int remove(const char \*path);

37654 **DESCRIPTION**

37655 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
37656 conflict between the requirements described here and the ISO C standard is unintentional. This  
37657 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

37658 The *remove()* function shall cause the file named by the path name pointed to by path to be no  
37659 longer accessible by that name. A subsequent attempt to open that file using that name shall fail,  
37660 unless it is created anew.

37661 CX If path does not name a directory, *remove(path)* shall be equivalent to *unlink(path)*.

37662 If path names a directory, *remove(path)* shall be equivalent to *rmdir(path)*.

37663 **RETURN VALUE**37664 CX Refer to *rmdir()* or *unlink()*.37665 **ERRORS**37666 CX Refer to *rmdir()* or *unlink()*.37667 **EXAMPLES**37668 **Removing Access to a File**37669 The following example shows how to remove access to a file named `/home/cnd/old_mods`.

37670 #include &lt;stdio.h&gt;

37671 int status;

37672 ...

37673 status = remove("/home/cnd/old\_mods");

37674 **APPLICATION USAGE**

37675 None.

37676 **RATIONALE**

37677 None.

37678 **FUTURE DIRECTIONS**

37679 None.

37680 **SEE ALSO**37681 *rmdir()*, *unlink()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>37682 **CHANGE HISTORY**

37683 First released in Issue 3.

37684 Entry included for alignment with the POSIX.1-1988 standard and the ISO C standard.

37685 **Issue 4**

37686 All statements containing references to *unlink()* and *rmdir()* in the DESCRIPTION, RETURN  
37687 VALUE, and ERRORS sections are marked as extensions.

37688 The following changes are incorporated for alignment with the ISO C standard:



- 37689           • The type of argument *path* is changed from **char\*** to **const char\***.
- 37690           • The DESCRIPTION is expanded to describe the operation of *remove()* more completely.
- 37691 **Issue 6**
- 37692           Extensions beyond the ISO C standard are now marked.
- 37693           The following new requirements on POSIX implementations derive from alignment with the
- 37694           Single UNIX Specification:
- 37695           • The DESCRIPTION, RETURN VALUE, and ERRORS sections are updated so that if *path* is
- 37696           not a directory, *remove()* is equivalent to *unlink()*, and if it is a directory, it is equivalent to
- 37697           *rmdir()*.

37698 **NAME**

37699           remque — remove an element from a queue

37700 **SYNOPSIS**

37701 xSI       #include <search.h>

37702           void remque(void \*element);

37703

37704 **DESCRIPTION**

37705           Refer to *insque()*.

37706 **NAME**

37707 remquo, remquof, remquol — remainder functions

37708 **SYNOPSIS**

37709 #include &lt;math.h&gt;

37710 double remquo(double x, double y, int \*quo);

37711 float remquof(float x, float y, int \*quo);

37712 long double remquol(long double x, long double y, int \*quo);

37713 **DESCRIPTION**

37714 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 37715 conflict between the requirements described here and the ISO C standard is unintentional. This  
 37716 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

37717 These functions shall compute the same remainder as the *remainder()*, *remainderf()*, and  
 37718 *remainderl()* functions, respectively. In the object pointed to by *quo* they store a value whose sign  
 37719 is the sign of  $x/y$  and whose magnitude is congruent modulo  $2^n$  to the magnitude of the integral  
 37720 quotient of  $x/y$ , where  $n$  is an implementation-defined integer greater than or equal to 3.

37721 An application wishing to check for error situations should set *errno* to 0 before calling these  
 37722 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

37723 **RETURN VALUE**37724 These functions shall return  $x \text{ REM } y$ .

37725 When  $y$  is 0, these functions shall return NaN (or equivalent if available) and set *errno* to  
 37726 [EDOM].

37727 If the value of  $x$  is  $\pm\text{Inf}$ , these functions shall return NaN and set *errno* to [EDOM].

37728 If  $x$  or  $y$  is NaN, then these functions shall return NaN and *errno* may be set to [EDOM].

37729 **ERRORS**

37730 These functions shall fail if:

37731 [EDOM] The  $y$  argument is 0 or the  $x$  argument is positive or negative infinity.

37732 These functions may fail if:

37733 [EDOM] The  $x$  or  $y$  argument is NaN.

37734 **EXAMPLES**

37735 None.

37736 **APPLICATION USAGE**

37737 None.

37738 **RATIONALE**

37739 These functions are intended for implementing argument reductions which can exploit a few  
 37740 low-order bits of the quotient. Note that  $x$  may be so large in magnitude relative to  $y$  that an  
 37741 exact representation of the quotient is not practical.

37742 **FUTURE DIRECTIONS**

37743 None.

37744 **SEE ALSO**37745 *remainder()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

37746 **CHANGE HISTORY**

37747 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

37748 **NAME**

37749 rename — rename a file

37750 **SYNOPSIS**

37751 #include &lt;stdio.h&gt;

37752 int rename(const char \*old, const char \*new);

37753 **DESCRIPTION**

37754 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 37755 conflict between the requirements described here and the ISO C standard is unintentional. This  
 37756 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

37757 The *rename()* function shall change the name of a file. The *old* argument points to the path name  
 37758 of the file to be renamed. The *new* argument points to the new path name of the file.

37759 cx If the *old* argument and the *new* argument both refer to, and both link to, the same existing file,  
 37760 *rename()* shall return successfully and perform no other action.

37761 If the *old* argument points to the path name of a file that is not a directory, the application shall  
 37762 ensure that the *new* argument does not point to the path name of a directory. If the link named  
 37763 by the *new* argument exists, it shall be removed and *old* renamed to *new*. In this case, a link  
 37764 named *new* shall remain visible to other processes throughout the renaming operation and refer  
 37765 either to the file referred to by *new* or *old* before the operation began. Write access permission is  
 37766 required for both the directory containing *old* and the directory containing *new*.

37767 If the *old* argument points to the path name of a directory, the application shall ensure that the  
 37768 *new* argument does not point to the path name of a file that is not a directory. If the directory  
 37769 named by the *new* argument exists, it shall be removed and *old* renamed to *new*. In this case, a  
 37770 link named *new* shall exist throughout the renaming operation and shall refer either to the  
 37771 directory referred to by *new* or *old* before the operation began. If *new* names an existing directory,  
 37772 the application shall ensure that it is an empty directory.

37773 If either the *old* or the *new* arguments name a symbolic link, *rename()* shall operate on the  
 37774 symbolic link itself, and shall not resolve the last component of the argument. If *old* points to a  
 37775 path name that names a symbolic link, the symbolic link shall be renamed. If *new* points to a  
 37776 path name that names a symbolic link, the symbolic link shall be removed.

37777 The application shall ensure that the *new* path name does not contain a path prefix that names  
 37778 *old*. Write access permission is required for the directory containing *old* and the directory  
 37779 containing *new*. If the *old* argument points to the path name of a directory, write access  
 37780 permission may be required for the directory named by *old*, and, if it exists, the directory named  
 37781 by *new*.

37782 If the link named by the *new* argument exists and the file's link count becomes 0 when it is  
 37783 removed and no process has the file open, the space occupied by the file shall be freed and the  
 37784 file shall no longer be accessible. If one or more processes have the file open when the last link is  
 37785 removed, the link shall be removed before *rename()* returns, but the removal of the file contents  
 37786 shall be postponed until all references to the file are closed.

37787 Upon successful completion, *rename()* shall mark for update the *st\_ctime* and *st\_mtime* fields of  
 37788 the parent directory of each file.

37789 If the *rename()* function fails for any reason other than [EIO], any file named by *new* shall be  
 37790 unaffected.

## 37791 RETURN VALUE

37792 CX Upon successful completion, *rename()* shall return 0; otherwise, -1 shall be returned, *errno* shall  
 37793 be set to indicate the error, and neither the file named by *old* nor the file named by *new* shall be  
 37794 changed or created.

## 37795 ERRORS

37796 The *rename()* function shall fail if:

37797 CX [EACCES] A component of either path prefix denies search permission; or one of the  
 37798 directories containing *old* or *new* denies write permissions; or, write  
 37799 permission is required and is denied for a directory pointed to by the *old* or  
 37800 *new* arguments.

37801 CX [EBUSY] The directory named by *old* or *new* is currently in use by the system or another  
 37802 process, and the implementation considers this an error.

37803 CX [EEXIST] or [ENOTEMPTY]  
 37804 The link named by *new* is a directory that is not an empty directory.

37805 CX [EINVAL] The *new* directory path name contains a path prefix that names the *old*  
 37806 directory.

37807 CX [EIO] A physical I/O error has occurred.

37808 CX [EISDIR] The *new* argument points to a directory and the *old* argument points to a file  
 37809 that is not a directory.

37810 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 37811 argument.

37812 CX [EMLINK] The file named by *old* is a directory, and the link count of the parent directory  
 37813 of *new* would exceed {LINK\_MAX}.

37814 CX [ENAMETOOLONG]  
 37815 The length of the *old* or *new* argument exceeds {PATH\_MAX} or a path name  
 37816 component is longer than {NAME\_MAX}.

37817 CX [ENOENT] The link named by *old* does not name an existing file, or either *old* or *new*  
 37818 points to an empty string.

37819 CX [ENOSPC] The directory that would contain *new* cannot be extended.

37820 CX [ENOTDIR] A component of either path prefix is not a directory; or the *old* argument  
 37821 names a directory and *new* argument names a non-directory file.

37822 XSI [EPERM] or [EACCES]  
 37823 The S\_ISVTX flag is set on the directory containing the file referred to by *old*  
 37824 and the caller is not the file owner, nor is the caller the directory owner, nor  
 37825 does the caller have appropriate privileges; or *new* refers to an existing file, the  
 37826 S\_ISVTX flag is set on the directory containing this file, and the caller is not  
 37827 the file owner, nor is the caller the directory owner, nor does the caller have  
 37828 appropriate privileges.

37829 CX [EROFS] The requested operation requires writing in a directory on a read-only file  
 37830 system.

37831 CX [EXDEV] The links named by *new* and *old* are on different file systems and the  
 37832 implementation does not support links between file systems.

37833 The *rename()* function may fail if:

|                            |                |                                                                                                                                                            |
|----------------------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 37834 XSI                  | [EBUSY]        | The file named by the <i>old</i> or <i>new</i> arguments is a named STREAM.                                                                                |
| 37835<br>37836             | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.                                                     |
| 37837 CX<br>37838<br>37839 | [ENAMETOOLONG] | As a result of encountering a symbolic link in resolution of the <i>path</i> argument, the length of the substituted path name string exceeded {PATH_MAX}. |
| 37840 CX<br>37841          | [ETXTBSY]      | The file to be renamed is a pure procedure (shared text) file that is being executed.                                                                      |

37842 **EXAMPLES**37843 **Renaming a File**

37844 The following example shows how to rename a file named `/home/cnd/mod1` to  
37845 `/home/cnd/mod2`.

```
37846 #include <stdio.h>
37847 int status;
37848 ...
37849 status = rename("/home/cnd/mod1", "/home/cnd/mod2");
```

37850 **APPLICATION USAGE**

37851 None.

37852 **RATIONALE**

37853 This *rename()* function is equivalent for regular files to that defined by the ISO C standard. Its  
37854 inclusion here expands that definition to include actions on directories and specifies behavior  
37855 when the *new* parameter names a file that already exists. That specification requires that the  
37856 action of the function be atomic.

37857 One of the reasons for introducing this function was to have a means of renaming directories  
37858 while permitting implementations to prohibit the use of *link()* and *unlink()* with directories,  
37859 thus constraining links to directories to those made by *mkdir()*.

37860 The specification that if *old* and *new* refer to the same file is intended to guarantee that:

```
37861 rename("x", "x");
37862 does not remove the file.
```

37863 Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

37864 See also the descriptions of [ENOTEMPTY] and [ENAMETOOLONG] in *rmdir()* and [EBUSY] in  
37865 *unlink()*. For a discussion of [EXDEV], see *link()*.

37866 **FUTURE DIRECTIONS**

37867 None.

37868 **SEE ALSO**

37869 *link()*, *rmdir()*, *symlink()*, *unlink()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
37870 `<stdio.h>`

37871 **CHANGE HISTORY**

37872 First released in Issue 3.

37873 Entry included for alignment with the POSIX.1-1988 standard.

37874 **Issue 4**

37875 The [EMLINK] error is added to the ERRORS section.

37876 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 37877
- The type of arguments *old* and *new* are changed from **char\*** to **const char\***.
  - The RETURN VALUE section now states that if an error occurs, neither file is changed or created.
- 37878  
37879

37880 The following change is incorporated for alignment with the FIPS requirements:

- 37881
- In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path name component is larger than {NAME\_MAX}, is now defined as mandatory and marked as an extension.
- 37882  
37883

37884 **Issue 4, Version 2**

37885 The following changes are made for X/OPEN UNIX conformance:

- 37886
- The DESCRIPTION is updated to indicate the results of naming a symbolic link in either *old* or *new*.
  - In the ERRORS section, [EIO] is added to indicate that a physical I/O error has occurred, [ELOOP] to indicate that too many symbolic links were encountered during path name resolution, and [EPERM] or [EACCES] to indicate a permission check failure when operating on directories with S\_ISVTX set.
  - In the ERRORS section, a second [ENAMETOOLONG] condition is defined that may report excessive length of an intermediate result of path name resolution of a symbolic link.
- 37887  
37888  
37889  
37890  
37891  
37892  
37893

37894 **Issue 5**

37895 The [EBUSY] error is added to the “may fail” part of the ERRORS section.

37896 **Issue 6**

37897 Extensions beyond the ISO C standard are now marked.

37898 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 37899
- The [ENAMETOOLONG] error is restored as an error dependent on \_POSIX\_NO\_TRUNC. This is since behavior may vary from one file system to another.
- 37900

37901 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 37902
- The [EIO] mandatory error condition is added.
  - The [ELOOP] mandatory error condition is added.
  - A second [ENAMETOOLONG] is added as an optional error condition.
  - The [ETXTBSY] optional error condition is added.
- 37903  
37904  
37905  
37906

37907 The following changes were made to align with the IEEE P1003.1a draft standard:

- 37908
- Details are added regarding the treatment of symbolic links.
  - The [ELOOP] optional error condition is added.
- 37909

37910 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.



37911 **NAME**

37912           rewind — reset file position indicator in a stream

37913 **SYNOPSIS**

37914           #include <stdio.h>

37915           void rewind(FILE \**stream*);

37916 **DESCRIPTION**

37917 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
37918           conflict between the requirements described here and the ISO C standard is unintentional. This  
37919           volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

37920           The call:

37921           rewind(*stream*)

37922           shall be equivalent to:

37923           (void) fseek(*stream*, 0L, SEEK\_SET)

37924           except that *rewind()* also clears the error indicator.

37925           Because *rewind()* does not return a value, an application wishing to detect errors should clear  
37926           *errno*, then call *rewind()*, and if *errno* is non-zero, assume an error has occurred.

37927 **RETURN VALUE**

37928           The *rewind()* function shall return no value.

37929 **ERRORS**

37930 **CX**       Refer to *fseek()* with the exception of [EINVAL] which does not apply.

37931 **EXAMPLES**

37932           None.

37933 **APPLICATION USAGE**

37934           None.

37935 **RATIONALE**

37936           None.

37937 **FUTURE DIRECTIONS**

37938           None.

37939 **SEE ALSO**

37940           *fseek()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>

37941 **CHANGE HISTORY**

37942           First released in Issue 1. Derived from Issue 1 of the SVID.

37943 **Issue 6**

37944           Extensions beyond the ISO C standard are now marked.

37945 **NAME**

37946           rewinddir — reset position of directory stream to the beginning of a directory

37947 **SYNOPSIS**

37948           #include &lt;dirent.h&gt;

37949           void rewinddir(DIR \*dirp);

37950 **DESCRIPTION**

37951           The *rewinddir()* function resets the position of the directory stream to which *dirp* refers to the  
 37952           beginning of the directory. It shall also cause the directory stream to refer to the current state of  
 37953           the corresponding directory, as a call to *opendir()* would have done. If *dirp* does not refer to a  
 37954           directory stream, the effect is undefined.

37955           After a call to the *fork()* function, either the parent or child (but not both) may continue  
 37956 XSI          processing the directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and  
 37957           child processes use these functions, the result is undefined.

37958 **RETURN VALUE**37959           The *rewinddir()* function shall not return a value.37960 **ERRORS**

37961           No errors are defined.

37962 **EXAMPLES**

37963           None.

37964 **APPLICATION USAGE**

37965           The *rewinddir()* function should be used in conjunction with *opendir()*, *readdir()*, and *closedir()* to  
 37966           examine the contents of the directory. This method is recommended for portability.

37967 **RATIONALE**

37968           None.

37969 **FUTURE DIRECTIONS**

37970           None.

37971 **SEE ALSO**

37972           *closedir()*, *opendir()*, *readdir()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <dirent.h>  
 37973           <sys/types.h>

37974 **CHANGE HISTORY**

37975           First released in Issue 2.

37976 **Issue 4**

37977           The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
 37978           XSI-conformant systems.

37979           The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 37980           • The last paragraph of the DESCRIPTION, describing a restriction after a *fork()* function, is  
 37981           added.

37982 **Issue 6**

37983           In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

37984           The following new requirements on POSIX implementations derive from alignment with the  
 37985           Single UNIX Specification:

- 37986           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
 37987           required for conforming implementations of previous POSIX specifications, it was not

37988

required for UNIX applications.

37989 **NAME**37990 rindex — character string operations (**LEGACY**)37991 **SYNOPSIS**37992 XSI `#include <strings.h>`37993 `char *rindex(const char *s, int c);`

37994

37995 **DESCRIPTION**37996 The *rindex()* function is identical to *strchr()*.37997 **RETURN VALUE**37998 Refer to *strchr()*.37999 **ERRORS**38000 Refer to *strchr()*.38001 **EXAMPLES**

38002 None.

38003 **APPLICATION USAGE**38004 *strchr()* is preferred over this function.38005 For maximum portability, it is recommended to replace the function call to *rindex()* as follows:38006 `#define rindex(a,b) strchr((a),(b))`38007 **RATIONALE**

38008 None.

38009 **FUTURE DIRECTIONS**

38010 This function may be withdrawn in a future version.

38011 **SEE ALSO**38012 *strchr()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<strings.h>`38013 **CHANGE HISTORY**

38014 First released in Issue 4, Version 2.

38015 **Issue 5**

38016 Moved from X/OPEN UNIX extension to BASE.

38017 **Issue 6**

38018 This function is marked LEGACY.

38019 **NAME**

38020 rint, rintf, rintl — round-to-nearest integral value

38021 **SYNOPSIS**

38022 #include &lt;math.h&gt;

38023 double rint(double x);

38024 float rintf(float x);

38025 long double rintl(long double x);

38026 **DESCRIPTION**

38027 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
38028 conflict between the requirements described here and the ISO C standard is unintentional. This  
38029 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

38030 These functions shall return the integral value (represented as a **double**) nearest  $x$  in the  
38031 direction of the current rounding mode. The current rounding mode is implementation-defined.

38032 If the current rounding mode rounds toward negative infinity, then *rint()* is identical to *floor()*.  
38033 If the current rounding mode rounds toward positive infinity, then *rint()* is identical to *ceil()*.

38034 These functions differ from the *nearbyint()*, *nearbyintf()*, and *nearbyintl()* functions only in that  
38035 they may raise the inexact floating-point exception if the result differs in value from the  
38036 argument.

38037 **RETURN VALUE**

38038 Upon successful completion, these functions shall return the integer (represented as a double  
38039 precision number) nearest  $x$  in the direction of the current rounding mode.

38040 When  $x$  is  $\pm\text{Inf}$ , *rint()* shall return  $x$ .

38041 If the value of  $x$  is NaN, NaN shall be returned and *errno* may be set to [EDOM].

38042 **ERRORS**

38043 These functions may fail if:

38044 [EDOM] The  $x$  argument is NaN.

38045 **EXAMPLES**

38046 None.

38047 **APPLICATION USAGE**

38048 None.

38049 **RATIONALE**

38050 None.

38051 **FUTURE DIRECTIONS**

38052 None.

38053 **SEE ALSO**

38054 *abs()*, *isnan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

38055 **CHANGE HISTORY**

38056 First released in Issue 4, Version 2.

38057 **Issue 5**

38058 Moved from X/OPEN UNIX extension to BASE.

38059 **Issue 6**

38060

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

38061

- The *rintf()* and *rintl()* functions are added.

38062

- The *rint()* function is no longer marked XSI as it is part of the ISO/IEC 9899:1999 standard.

38063 **NAME**

38064 rmdir — remove a directory

38065 **SYNOPSIS**

38066 #include &lt;unistd.h&gt;

38067 int rmdir(const char \*path);

38068 **DESCRIPTION**38069 The *rmdir()* function shall remove a directory whose name is given by *path*. The directory is  
38070 removed only if it is an empty directory.38071 If the directory is the root directory or the current working directory of any process, it is  
38072 unspecified whether the function succeeds, or whether it shall fail and set *errno* to [EBUSY].38073 If *path* names a symbolic link, then *rmdir()* shall fail and set *errno* to [ENOTDIR].38074 If the *path* argument refers to a path whose final component is either dot or dot-dot, *rmdir()* shall  
38075 fail.38076 If the directory's link count becomes 0 and no process has the directory open, the space occupied  
38077 by the directory shall be freed and the directory shall no longer be accessible. If one or more  
38078 processes have the directory open when the last link is removed, the dot and dot-dot entries, if  
38079 present, are removed before *rmdir()* returns and no new entries may be created in the directory,  
38080 but the directory is not removed until all references to the directory are closed.38081 If the directory is not an empty directory, *rmdir()* shall fail and set *errno* to [EEXIST] or  
38082 [ENOTEMPTY].38083 Upon successful completion, the *rmdir()* function shall mark for update the *st\_ctime* and  
38084 *st\_mtime* fields of the parent directory.38085 **RETURN VALUE**38086 Upon successful completion, the function *rmdir()* shall return 0. Otherwise, -1 shall be returned,  
38087 and *errno* set to indicate the error. If -1 is returned, the named directory shall not be changed.38088 **ERRORS**38089 The *rmdir()* function shall fail if:38090 [EACCES] Search permission is denied on a component of the path prefix, or write  
38091 permission is denied on the parent directory of the directory to be removed.38092 [EBUSY] The directory to be removed is currently in use by the system or some process  
38093 and the implementation considers this to be an error.

38094 [EEXIST] or [ENOTEMPTY]

38095 The *path* argument names a directory that is not an empty directory, or there  
38096 are hard links to the directory other than dot or a single entry in dot-dot.38097 [EINVAL] The *path* argument contains a last component that is dot.

38098 [EIO] A physical I/O error has occurred.

38099 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
38100 argument.

38101 [ENAMETOOLONG]

38102 The length of the *path* argument exceeds {PATH\_MAX} or a path name  
38103 component is longer than

38104 NAME\_MAX

|           |                     |                                                                                        |
|-----------|---------------------|----------------------------------------------------------------------------------------|
| 38105     | [ENOENT]            | A component of <i>path</i> does not name an existing file, or the <i>path</i> argument |
| 38106     |                     | names a nonexistent directory or points to an empty string.                            |
| 38107     | [ENOTDIR]           | A component of <i>path</i> is not a directory.                                         |
| 38108 XSI | [EPERM] or [EACCES] |                                                                                        |
| 38109     |                     | The S_ISVTX flag is set on the parent directory of the directory to be removed         |
| 38110     |                     | and the caller is not the owner of the directory to be removed, nor is the caller      |
| 38111     |                     | the owner of the parent directory, nor does the caller have the appropriate            |
| 38112     |                     | privileges.                                                                            |
| 38113     | [EROFS]             | The directory entry to be removed resides on a read-only file system.                  |
| 38114     |                     | The <i>rmdir()</i> function may fail if:                                               |
| 38115     | [ELOOP]             | More than {SYMLOOP_MAX} symbolic links were encountered during                         |
| 38116     |                     | resolution of the <i>path</i> argument.                                                |
| 38117     | [ENAMETOOLONG]      |                                                                                        |
| 38118     |                     | As a result of encountering a symbolic link in resolution of the <i>path</i> argument, |
| 38119     |                     | the length of the substituted path name string exceeded {PATH_MAX}.                    |

### 38120 EXAMPLES

#### 38121 Removing a Directory

38122 The following example shows how to remove a directory named `/home/cnd/mod1`.

```
38123 #include <unistd.h>
38124 int status;
38125 ...
38126 status = rmdir("/home/cnd/mod1");
```

### 38127 APPLICATION USAGE

38128 None.

### 38129 RATIONALE

38130 The *rmdir()* and *rename()* functions originated in 4.2 BSD, and they used [ENOTEMPTY] for the  
 38131 condition when the directory to be removed does not exist or *new* already exists. When the 1984  
 38132 /usr/group standard was published, it contained [EEXIST] instead. When these functions were  
 38133 adopted into System V, the 1984 /usr/group standard was used as a reference. Therefore, several  
 38134 existing applications and implementations support/use both forms, and no agreement could be  
 38135 reached on either value. All implementations are required to supply both [EEXIST] and  
 38136 [ENOTEMPTY] in `<errno.h>` with distinct values, so that applications can use both values in C-  
 38137 language **case** statements.

38138 The meaning of deleting *pathname/dot* is unclear, because the name of the file (directory) in the  
 38139 parent directory to be removed is not clear, particularly in the presence of multiple links to a  
 38140 directory.

38141 IEEE Std. 1003.1-200x was silent with regard to the behavior of *rmdir()* when there are multiple  
 38142 hard links to the directory being removed. The requirement to set *errno* to [EEXIST] or  
 38143 [ENOTEMPTY] clarifies the behavior in this case.

38144 If the process's home directory is being removed, that should be an allowed error.

38145 Virtually all existing implementations detect [ENOTEMPTY] or the case of dot-dot. The text in  
 38146 Section 2.3 (on page 521) about returning any one of the possible errors permits that behavior to  
 38147 continue. The [ELOOP] error may be returned if more than {SYMLOOP\_MAX} symbolic links



38148 are encountered during resolution of the *path* argument.

#### 38149 FUTURE DIRECTIONS

38150 None.

#### 38151 SEE ALSO

38152 *mkdir()*, *remove()*, *unlink()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>

#### 38153 CHANGE HISTORY

38154 First released in Issue 3.

38155 Entry included for alignment with the POSIX.1-1988 standard.

#### 38156 Issue 4

38157 The <**unistd.h**> header is added to the SYNOPSIS section.

38158 The [ENAMETOOLONG] description is amended.

38159 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 38160 • The type of argument *path* is changed from **char\*** to **const char\***.
- 38161 • The DESCRIPTION is expanded to indicate that, if the directory is a root directory or a
- 38162 current working directory, it is unspecified whether the function succeeds, or whether it fails
- 38163 and sets *errno* to [EBUSY]. In Issue 3, the behavior under these circumstances was defined as
- 38164 implementation-defined.
- 38165 • The RETURN VALUE section is expanded to direct that if  $-1$  is returned, the directory is not
- 38166 changed.

38167 The following change is incorporated for alignment with the FIPS requirements:

- 38168 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path
- 38169 name component is larger than {NAME\_MAX} is now defined as mandatory and marked as
- 38170 an extension.

#### 38171 Issue 4, Version 2

38172 The following changes are made for X/OPEN UNIX conformance:

- 38173 • The DESCRIPTION is updated to indicate the results of naming a symbolic link in *path*.
- 38174 • In the ERRORS section, [EIO] is added to indicate that a physical I/O error has occurred,
- 38175 [ELOOP] to indicate that too many symbolic links were encountered during path name
- 38176 resolution, and [EPERM] or [EACCES] to indicate a permission check failure when operating
- 38177 on directories with S\_ISVTX set.
- 38178 • In the ERRORS section, a second [ENAMETOOLONG] condition is defined that may report
- 38179 excessive length of an intermediate result of path name resolution of a symbolic link.

#### 38180 Issue 6

38181 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 38182 • The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.
- 38183 This is since behavior may vary from one file system to another.

38184 The following new requirements on POSIX implementations derive from alignment with the

38185 Single UNIX Specification:

- 38186 • The DESCRIPTION is updated to indicate the results of naming a symbolic link in *path*.
- 38187 • The [EIO] mandatory error condition is added.

- 38188           • The [ELOOP] mandatory error condition is added.
- 38189           • A second [ENAMETOOLONG] is added as an optional error condition.
- 38190       The following changes were made to align with the IEEE P1003.1a draft standard:
- 38191           • The [ELOOP] optional error condition is added.

38192 **NAME**

38193 round, roundf, roundl — round to nearest integer value in floating-point format

38194 **SYNOPSIS**

38195 #include <math.h>

38196 double round(double x);

38197 float roundf(float x);

38198 long double roundl(long double x);

38199 **DESCRIPTION**

38200 *cx* The functionality described on this reference page is aligned with the ISO C standard. Any  
38201 conflict between the requirements described here and the ISO C standard is unintentional. This  
38202 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

38203 These functions shall round their argument to the nearest integer value in floating-point format,  
38204 rounding halfway cases away from zero, regardless of the current rounding direction.

38205 An application wishing to check for error situations should set *errno* to 0 before calling these  
38206 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

38207 **RETURN VALUE**

38208 Upon successful completion, these functions shall return the rounded integer value.

38209 If *x* is  $\pm\text{Inf}$ , these functions shall return *x*.

38210 If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

38211 **ERRORS**

38212 These functions may fail if:

38213 [EDOM] The value of *x* is NaN.

38214 **EXAMPLES**

38215 None.

38216 **APPLICATION USAGE**

38217 None.

38218 **RATIONALE**

38219 None.

38220 **FUTURE DIRECTIONS**

38221 None.

38222 **SEE ALSO**

38223 The Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

38224 **CHANGE HISTORY**

38225 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38226 **NAME**

38227 scalb — load exponent of a radix-independent floating-point number

38228 **SYNOPSIS**38229 XSI `#include <math.h>`38230 `double scalb(double x, double n);`

38231

38232 **DESCRIPTION**

38233 The *scalb()* function shall compute  $x \cdot r^n$ , where  $r$  is the radix of the machine's floating point  
 38234 arithmetic. When  $r$  is 2, *scalb()* is equivalent to *ldexp()*. The value of  $r$  is FLT\_RADIX which is  
 38235 defined in `<float.h>`.

38236 An application wishing to check for error situations should set *errno* to 0 before calling *scalb()*. If  
 38237 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

38238 **RETURN VALUE**38239 Upon successful completion, the *scalb()* function shall return  $x \cdot r^n$ .

38240 If the correct value would overflow, *scalb()* shall return  $\pm$ HUGE\_VAL (according to the sign of  $x$ )  
 38241 and set *errno* to [ERANGE].

38242 If the correct value would underflow, *scalb()* shall return 0 and set *errno* to [ERANGE].

38243 The *scalb()* function shall return  $x$  when  $x$  is  $\pm$ Inf.

38244 If  $x$  or  $n$  is NaN, then *scalb()* shall return NaN and may set *errno* to [EDOM].

38245 **ERRORS**38246 The *scalb()* function shall fail if:

38247 [ERANGE] The correct value would overflow or underflow.

38248 The *scalb()* function may fail if:

38249 [EDOM] The  $x$  or  $n$  argument is NaN.

38250 **EXAMPLES**

38251 None.

38252 **APPLICATION USAGE**

38253 None.

38254 **RATIONALE**

38255 None.

38256 **FUTURE DIRECTIONS**

38257 None.

38258 **SEE ALSO**

38259 *ilogb()*, *ldexp()*, *logb()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<float.h>`,  
 38260 `<math.h>`

38261 **CHANGE HISTORY**

38262 First released in Issue 4, Version 2.

38263 **Issue 5**

38264 Moved from X/OPEN UNIX extension to BASE.

38265 The DESCRIPTION is updated to indicate how an application should check for an error. This  
 38266 text was previously published in the APPLICATION USAGE section.

38267 **NAME**

38268 `scalbln, scalblnf, scalblnl` `scalbn, scalbnf, scalbnl`, — compute exponent using `FLT_RADIX`

38269 **SYNOPSIS**

38270 `#include <math.h>`

38271 `double scalbln(double x, long n);`  
 38272 `float scalblnf(float x, long n);`  
 38273 `long double scalblnl(long double x, long n);`  
 38274 `double scalbn(double x, int n);`  
 38275 `float scalbnf(float x, int n);`  
 38276 `long double scalbnl(long double x, int n);`

38277 **DESCRIPTION**

38278 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 38279 conflict between the requirements described here and the ISO C standard is unintentional. This  
 38280 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

38281 These functions shall compute  $x * FLT\_RADIX^n$  efficiently, not normally by computing  
 38282  $FLT\_RADIX^n$  explicitly.

38283 An application wishing to check for error situations should set *errno* to 0 before calling these  
 38284 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

38285 **RETURN VALUE**

38286 Upon successful completion, these functions shall return  $x * FLT\_RADIX^n$ .

38287 If the correct value would overflow, these functions shall return  $\pm HUGE\_VAL$  (according to the  
 38288 sign of *x*) and set *errno* to [ERANGE].

38289 If the correct value would underflow, these functions shall return 0 and set *errno* to [ERANGE].

38290 These functions shall return *x* when *x* is  $\pm Inf$ .

38291 If *x* or *n* is NaN, then these functions shall return NaN and may set *errno* to [EDOM].

38292 **ERRORS**

38293 These functions shall fail if:

38294 [ERANGE] The correct value would overflow or underflow.

38295 These functions may fail if:

38296 [EDOM] The *x* or *n* argument is NaN.

38297 **EXAMPLES**

38298 None.

38299 **APPLICATION USAGE**

38300 None.

38301 **RATIONALE**

38302 These functions are named so as to avoid conflicting with the Single UNIX Specification, which  
 38303 has a *scalb()* function whose second argument is **double** instead of **int**. The *scalb()* function is  
 38304 not part of ISO C standard. These functions, whose second parameter has type **long**, is provided  
 38305 because the factor required to scale from the smallest positive floating-point value to the largest  
 38306 finite one, on many implementations, is too large to represent in the minimum-width **int** format.

38307 **FUTURE DIRECTIONS**

38308           None.

38309 **SEE ALSO**

38310           *scalb()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

38311 **CHANGE HISTORY**

38312           First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

38313 **NAME**

38314           scanf — convert formatted input

38315 **SYNOPSIS**

38316           #include &lt;stdio.h&gt;

38317           int scanf(const char \*restrict *format*, ... );38318 **DESCRIPTION**38319           Refer to *fscanf()*.

38320 **NAME**

38321 sched\_get\_priority\_max, sched\_get\_priority\_min — get priority limits (**REALTIME**)

38322 **SYNOPSIS**

38323 PS #include <sched.h>

38324 int sched\_get\_priority\_max(int *policy*);

38325 int sched\_get\_priority\_min(int *policy*);

38326

38327 **DESCRIPTION**

38328 The *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions return the appropriate  
38329 maximum or minimum, respectively, for the scheduling policy specified by *policy*.

38330 The value of *policy* is one of the scheduling policy values defined in <**sched.h**>.

38331 **RETURN VALUE**

38332 If successful, the *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall return the  
38333 appropriate maximum or minimum values, respectively. If unsuccessful, they shall return a  
38334 value of  $-1$  and set *errno* to indicate the error.

38335 **ERRORS**

38336 The *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall fail if:

38337 [EINVAL] The value of the *policy* parameter does not represent a defined scheduling  
38338 policy.

38339 **EXAMPLES**

38340 None.

38341 **APPLICATION USAGE**

38342 None.

38343 **RATIONALE**

38344 None.

38345 **FUTURE DIRECTIONS**

38346 None.

38347 **SEE ALSO**

38348 *sched\_getparam()*, *sched\_setparam()*, *sched\_getscheduler()*, *sched\_rr\_get\_interval()*,  
38349 *sched\_setscheduler()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**sched.h**>

38350 **CHANGE HISTORY**

38351 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38352 **Issue 6**

38353 These functions are marked as part of the Process Scheduling option.

38354 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38355 implementation does not support the Process Scheduling option.

38356 The [ESRCH] error condition has been removed since these functions do not take a *pid*  
38357 argument.



38358 **NAME**38359 sched\_getparam — get scheduling parameters (**REALTIME**)38360 **SYNOPSIS**

38361 PS #include &lt;sched.h&gt;

38362 int sched\_getparam(pid\_t pid, struct sched\_param \*param);

38363

38364 **DESCRIPTION**38365 The *sched\_getparam()* function shall return the scheduling parameters of a process specified by  
38366 *pid* in the **sched\_param** structure pointed to by *param*.38367 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
38368 parameters for the process whose process ID is equal to *pid* shall be returned.38369 If *pid* is zero, the scheduling parameters for the calling process shall be returned. The behavior of  
38370 the *sched\_getparam()* function is unspecified if the value of *pid* is negative.38371 **RETURN VALUE**38372 Upon successful completion, the *sched\_getparam()* function shall return zero. If the call to  
38373 *sched\_getparam()* is unsuccessful, the function shall return a value of -1 and set *errno* to indicate  
38374 the error.38375 **ERRORS**38376 The *sched\_getparam()* function shall fail if:38377 [EPERM] The requesting process does not have permission to obtain the scheduling  
38378 parameters of the specified process.38379 [ESRCH] No process can be found corresponding to that specified by *pid*.38380 **EXAMPLES**

38381 None.

38382 **APPLICATION USAGE**

38383 None.

38384 **RATIONALE**

38385 None.

38386 **FUTURE DIRECTIONS**

38387 None.

38388 **SEE ALSO**38389 *sched\_getscheduler()*, *sched\_setparam()*, *sched\_setscheduler()*, the Base Definitions volume of  
38390 IEEE Std. 1003.1-200x, <**sched.h**>38391 **CHANGE HISTORY**

38392 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38393 **Issue 6**38394 The *sched\_getparam()* function is marked as part of the Process Scheduling option.38395 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38396 implementation does not support the Process Scheduling option.

38397 **NAME**

38398 sched\_getscheduler — get scheduling policy (**REALTIME**)

38399 **SYNOPSIS**

38400 PS #include <sched.h>

38401 int sched\_getscheduler(pid\_t pid);

38402

38403 **DESCRIPTION**

38404 The *sched\_getscheduler()* function shall return the scheduling policy of the process specified by  
 38405 *pid*. If the value of *pid* is negative, the behavior of the *sched\_getscheduler()* function is  
 38406 unspecified.

38407 The values that can be returned by *sched\_getscheduler()* are defined in the header file <**sched.h**>.

38408 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
 38409 policy shall be returned for the process whose process ID is equal to *pid*.

38410 If *pid* is zero, the scheduling policy shall be returned for the calling process.

38411 **RETURN VALUE**

38412 Upon successful completion, the *sched\_getscheduler()* function shall return the scheduling policy  
 38413 of the specified process. If unsuccessful, the function shall return  $-1$  and set *errno* to indicate the  
 38414 error.

38415 **ERRORS**

38416 The *sched\_getscheduler()* function shall fail if:

38417 [EPERM] The requesting process does not have permission to determine the scheduling  
 38418 policy of the specified process.

38419 [ESRCH] No process can be found corresponding to that specified by *pid*.

38420 **EXAMPLES**

38421 None.

38422 **APPLICATION USAGE**

38423 None.

38424 **RATIONALE**

38425 None.

38426 **FUTURE DIRECTIONS**

38427 None.

38428 **SEE ALSO**

38429 *sched\_getparam()*, *sched\_setparam()*, *sched\_setscheduler()*, the Base Definitions volume of  
 38430 IEEE Std. 1003.1-200x, <**sched.h**>

38431 **CHANGE HISTORY**

38432 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38433 **Issue 6**

38434 The *sched\_getscheduler()* function is marked as part of the Process Scheduling option.

38435 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 38436 implementation does not support the Process Scheduling option.

38437 **NAME**

38438 sched\_rr\_get\_interval — get execution time limits (**REALTIME**)

38439 **SYNOPSIS**

38440 PS #include <sched.h>

38441 int sched\_rr\_get\_interval(pid\_t pid, struct timespec \*interval);

38442

38443 **DESCRIPTION**

38444 The *sched\_rr\_get\_interval()* function updates the **timespec** structure referenced by the *interval*  
38445 argument to contain the current execution time limit (that is, time quantum) for the process  
38446 specified by *pid*. If *pid* is zero, the current execution time limit for the calling process shall be  
38447 returned.

38448 **RETURN VALUE**

38449 If successful, the *sched\_rr\_get\_interval()* function shall return zero. Otherwise, it shall return a  
38450 value of  $-1$  and set *errno* to indicate the error.

38451 **ERRORS**

38452 The *sched\_rr\_get\_interval()* function shall fail if:

38453 [ESRCH] No process can be found corresponding to that specified by *pid*.

38454 **EXAMPLES**

38455 None.

38456 **APPLICATION USAGE**

38457 None.

38458 **RATIONALE**

38459 None.

38460 **FUTURE DIRECTIONS**

38461 None.

38462 **SEE ALSO**

38463 *sched\_getparam()*, *sched\_get\_priority\_max()*, *sched\_getscheduler()*, *sched\_setparam()*,  
38464 *sched\_setscheduler()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**sched.h**>

38465 **CHANGE HISTORY**

38466 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38467 **Issue 6**

38468 The *sched\_rr\_get\_interval()* function is marked as part of the Process Scheduling option.

38469 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38470 implementation does not support the Process Scheduling option.

## 38471 NAME

38472 sched\_setparam — set scheduling parameters (**REALTIME**)

## 38473 SYNOPSIS

38474 PS #include &lt;sched.h&gt;

38475 int sched\_setparam(pid\_t pid, const struct sched\_param \*param);

38476

## 38477 DESCRIPTION

38478 The *sched\_setparam()* function sets the scheduling parameters of the process specified by *pid* to  
 38479 the values specified by the **sched\_param** structure pointed to by *param*. The value of the  
 38480 *sched\_priority* member in the **sched\_param** structure is any integer within the inclusive priority  
 38481 range for the current scheduling policy of the process specified by *pid*. Higher numerical values  
 38482 for the priority represent higher priorities. If the value of *pid* is negative, the behavior of the  
 38483 *sched\_setparam()* function is unspecified.

38484 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
 38485 parameters shall be set for the process whose process ID is equal to *pid*.

38486 If *pid* is zero, the scheduling parameters shall be set for the calling process.

38487 The conditions under which one process has permission to change the scheduling parameters of  
 38488 another process are implementation-defined.

38489 Implementations may require the requesting process to have the appropriate privilege to set its  
 38490 own scheduling parameters or those of another process.

38491 The target process, whether it is running or not running, resumes execution after all other  
 38492 runnable processes of equal or greater priority have been scheduled to run.

38493 If the priority of the process specified by the *pid* argument is set higher than that of the lowest  
 38494 priority running process and if the specified process is ready to run, the process specified by the  
 38495 *pid* argument preempts a lowest priority running process. Similarly, if the process calling  
 38496 *sched\_setparam()* sets its own priority lower than that of one or more other non-empty process  
 38497 lists, then the process that is the head of the highest priority list also preempts the calling  
 38498 process. Thus, in either case, the originating process might not receive notification of the  
 38499 completion of the requested priority change until the higher priority process has executed.

38500 ss If the scheduling policy of the target process is SCHED\_SPORADIC, the value specified by the  
 38501 *sched\_ss\_low\_priority* member of the *param* argument shall be any integer within the inclusive  
 38502 priority range for the sporadic server policy. The *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget*  
 38503 members of the *param* argument shall represent the time parameters to be used by the sporadic  
 38504 server scheduling policy for the target process. The *sched\_ss\_max\_repl* member of the *param*  
 38505 argument shall represent the maximum number of replenishments that are allowed to be  
 38506 pending simultaneously for the process scheduled under this scheduling policy.

38507 The specified *sched\_ss\_repl\_period* shall be greater than or equal to the specified  
 38508 *sched\_ss\_init\_budget* for the function to succeed; if it is not, then the function shall fail.

38509 The value of *sched\_ss\_max\_repl* shall be within the inclusive range [1,{SS\_REPL\_MAX}] for the  
 38510 function to succeed; if not, the function shall fail.

38511 If the scheduling policy of the target process is either SCHED\_FIFO or SCHED\_RR, the  
 38512 *sched\_ss\_low\_priority*, *sched\_ss\_repl\_period*, and *sched\_ss\_init\_budget* members of the *param*  
 38513 argument shall have no effect on the scheduling behavior. If the scheduling policy of this process  
 38514 is not SCHED\_FIFO, SCHED\_RR, or SCHED\_SPORADIC, including SCHED\_OTHER, the  
 38515 effects of these members shall be implementation-defined.

38516 **Notes to Reviewers**38517 *This section with side shading will not appear in the final copy. - Ed.*

38518 D3, XSH, ERN 502 questions "including SCHED\_OTHER" above. (Problem left over from D2.)

38519 If the current scheduling policy for the process specified by *pid* is not SCHED\_FIFO,  
38520 ss SCHED\_RR, or SCHED\_SPORADIC, the result is implementation-defined; this case includes the  
38521 SCHED\_OTHER policy.38522 The effect of this function on individual threads is dependent on the scheduling contention  
38523 scope of the threads:38524 • For threads with system scheduling contention scope, these functions have no effect on their  
38525 scheduling.38526 • For threads with process scheduling contention scope, the threads' scheduling parameters  
38527 shall not be affected. However, the scheduling of these threads with respect to threads in  
38528 other processes may be dependent on the scheduling parameters of their process, which are  
38529 governed using these functions.38530 If an implementation supports a two-level scheduling model in which library threads are  
38531 multiplexed on top of several kernel-scheduled entities, then the underlying kernel-scheduled  
38532 entities for the system contention scope threads shall not be affected by these functions.38533 The underlying kernel-scheduled entities for the process contention scope threads shall have  
38534 their scheduling parameters changed to the value specified in *param*. Kernel scheduled entities  
38535 for use by process contention scope threads that are created after this call completes inherit their  
38536 scheduling policy and associated scheduling parameters from the process.38537 This function is not atomic with respect to other threads in the process. Threads are allowed to  
38538 continue to execute while this function call is in the process of changing the scheduling policy  
38539 for the underlying kernel-scheduled entities used by the process contention scope threads.38540 **RETURN VALUE**38541 If successful, the *sched\_setparam()* function shall return zero.38542 If the call to *sched\_setparam()* is unsuccessful, the priority shall remain unchanged, and the  
38543 function shall return a value of -1 and set *errno* to indicate the error.38544 **ERRORS**38545 The *sched\_setparam()* function shall fail if:38546 [EINVAL] One or more of the requested scheduling parameters is outside the range  
38547 defined for the scheduling policy of the specified *pid*.38548 [EPERM] The requesting process does not have permission to set the scheduling  
38549 parameters for the specified process, or does not have the appropriate  
38550 privilege to invoke *sched\_setparam()*.38551 [ESRCH] No process can be found corresponding to that specified by *pid*.

38552 **EXAMPLES**

38553 None.

38554 **APPLICATION USAGE**

38555 None.

38556 **RATIONALE**

38557 None.

38558 **FUTURE DIRECTIONS**

38559 None.

38560 **SEE ALSO**

38561 *sched\_getparam()*, *sched\_getscheduler()*, *sched\_setscheduler()*, the Base Definitions volume of  
38562 IEEE Std. 1003.1-200x, <**sched.h**>

38563 **CHANGE HISTORY**

38564 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38565 **Issue 6**38566 The *sched\_setparam()* function is marked as part of the Process Scheduling option.

38567 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38568 implementation does not support the Process Scheduling option.

38569 The following new requirements on POSIX implementations derive from alignment with the  
38570 Single UNIX Specification:

38571 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is  
38572 added.

38573 • Sections describing two-level scheduling and atomicity of the function are added.

38574 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std. 1003.1d-1999.

38575 **NAME**38576 sched\_setscheduler — set scheduling policy and parameters (**REALTIME**)38577 **SYNOPSIS**38578 PS `#include <sched.h>`38579 `int sched_setscheduler(pid_t pid, int policy,`  
38580 `const struct sched_param *param);`

38581

38582 **DESCRIPTION**

38583 The `sched_setscheduler()` function sets the scheduling policy and scheduling parameters of the  
 38584 process specified by `pid` to `policy` and the parameters specified in the `sched_param` structure  
 38585 pointed to by `param`, respectively. The value of the `sched_priority` member in the `sched_param`  
 38586 structure is any integer within the inclusive priority range for the scheduling policy specified by  
 38587 `policy`. If the value of `pid` is negative, the behavior of the `sched_setscheduler()` function is  
 38588 unspecified.

38589 The possible values for the `policy` parameter are defined in the header file `<sched.h>`.

38590 If a process specified by `pid` exists, and if the calling process has permission, the scheduling  
 38591 policy and scheduling parameters shall be set for the process whose process ID is equal to `pid`.

38592 If `pid` is zero, the scheduling policy and scheduling parameters shall be set for the calling  
 38593 process.

38594 The conditions under which one process has the appropriate privilege to change the scheduling  
 38595 parameters of another process are implementation-defined.

38596 Implementations may require that the requesting process have permission to set its own  
 38597 scheduling parameters or those of another process. Additionally, implementation-defined  
 38598 restrictions may apply as to the appropriate privileges required to set a process' own scheduling  
 38599 policy, or another process' scheduling policy, to a particular value.

38600 The `sched_setscheduler()` function is considered successful if it succeeds in setting the scheduling  
 38601 policy and scheduling parameters of the process specified by `pid` to the values specified by `policy`  
 38602 and the structure pointed to by `param`, respectively.

38603 ss If the scheduling policy specified by `policy` is `SCHED_SPORADIC`, the value specified by the  
 38604 `sched_ss_low_priority` member of the `param` argument shall be any integer within the inclusive  
 38605 priority range for the sporadic server policy. The `sched_ss_repl_period` and `sched_ss_init_budget`  
 38606 members of the `param` argument shall represent the time parameters used by the sporadic server  
 38607 scheduling policy for the target process. The `sched_ss_max_repl` member of the `param` argument  
 38608 shall represent the maximum number of replenishments that are allowed to be pending  
 38609 simultaneously for the process scheduled under this scheduling policy.

38610 The specified `sched_ss_repl_period` shall be greater than or equal to the specified  
 38611 `sched_ss_init_budget` for the function to succeed; if it is not, then the function shall fail.

38612 The value of `sched_ss_max_repl` shall be within the inclusive range `[1, {SS_REPL_MAX}]` for the  
 38613 function to succeed; if not, the function shall fail.

38614 If the scheduling policy specified by `policy` is either `SCHED_FIFO` or `SCHED_RR`, the  
 38615 `sched_ss_low_priority`, `sched_ss_repl_period`, and `sched_ss_init_budget` members of the `param`  
 38616 argument shall have no effect on the scheduling behavior.

38617 The effect of this function on individual threads is dependent on the scheduling contention  
 38618 scope of the threads:

38619           • For threads with system scheduling contention scope, these functions have no effect on their  
38620 scheduling.

38621           • For threads with process scheduling contention scope, the threads' scheduling policy and  
38622 associated parameters shall not be affected. However, the scheduling of these threads with  
38623 respect to threads in other processes may be dependent on the scheduling parameters of their  
38624 process, which are governed using these functions.

38625           If an implementation supports a two-level scheduling model in which library threads are  
38626 multiplexed on top of several kernel-scheduled entities, then the underlying kernel-scheduled  
38627 entities for the system contention scope threads shall not be affected by these functions.

38628           The underlying kernel-scheduled entities for the process contention scope threads shall have  
38629 their scheduling policy and associated scheduling parameters changed to the values specified in  
38630 *policy* and *param*, respectively. Kernel scheduled entities for use by process contention scope  
38631 threads that are created after this call completes inherit their scheduling policy and associated  
38632 scheduling parameters from the process.

38633           This function is not atomic with respect to other threads in the process. Threads are allowed to  
38634 continue to execute while this function call is in the process of changing the scheduling policy  
38635 and associated scheduling parameters for the underlying kernel-scheduled entities used by the  
38636 process contention scope threads.

#### 38637 RETURN VALUE

38638           Upon successful completion, the function shall return the former scheduling policy of the  
38639 specified process. If the *sched\_setscheduler()* function fails to complete successfully, the policy  
38640 and scheduling parameters shall remain unchanged, and the function shall return a value of  $-1$   
38641 and set *errno* to indicate the error.

#### 38642 ERRORS

38643           The *sched\_setscheduler()* function shall fail if:

38644           [EINVAL]           The value of the *policy* parameter is invalid, or one or more of the parameters  
38645 contained in *param* is outside the valid range for the specified scheduling  
38646 policy.

38647           [EPERM]           The requesting process does not have permission to set either or both of the  
38648 scheduling parameters or the scheduling policy of the specified process.

38649           [ESRCH]           No process can be found corresponding to that specified by *pid*.

#### 38650 EXAMPLES

38651           None.

#### 38652 APPLICATION USAGE

38653           None.

#### 38654 RATIONALE

38655           None.

#### 38656 FUTURE DIRECTIONS

38657           None.

#### 38658 SEE ALSO

38659           *sched\_getparam()*, *sched\_getscheduler()*, *sched\_setparam()*, the Base Definitions volume of  
38660 IEEE Std. 1003.1-200x, <*sched.h*>



38661 **CHANGE HISTORY**

38662 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38663 **Issue 6**

38664 The *sched\_setscheduler()* function is marked as part of the Process Scheduling option.

38665 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38666 implementation does not support the Process Scheduling option.

38667 The following new requirements on POSIX implementations derive from alignment with the  
38668 Single UNIX Specification:

38669 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is  
38670 added.

38671 • Sections describing two-level scheduling and atomicity of the function are added.

38672 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std. 1003.1d-1999.

38673 **NAME**

38674 sched\_yield — yield processor

38675 **SYNOPSIS**

38676 PS|THR #include &lt;sched.h&gt;

38677 int sched\_yield(void);

38678

38679 **DESCRIPTION**38680 The *sched\_yield()* function forces the running thread to relinquish the processor until it again  
38681 becomes the head of its thread list. It takes no arguments.38682 **RETURN VALUE**38683 The *sched\_yield()* function shall return 0 if it completes successfully; otherwise, it shall return a  
38684 value of -1 and set *errno* to indicate the error.38685 **ERRORS**

38686 No errors are defined.

38687 **EXAMPLES**

38688 None.

38689 **APPLICATION USAGE**

38690 None.

38691 **RATIONALE**

38692 None.

38693 **FUTURE DIRECTIONS**

38694 None.

38695 **SEE ALSO**

38696 The Base Definitions volume of IEEE Std. 1003.1-200x, &lt;sched.h&gt;

38697 **CHANGE HISTORY**38698 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
38699 POSIX Threads Extension.38700 **Issue 6**38701 The *sched\_yield()* function is now marked as part of the Process Scheduling and Threads options.

38702 **NAME**

38703        seed48 — seed uniformly distributed pseudo-random non-negative long integer generator

38704 **SYNOPSIS**

38705 xSI        #include &lt;stdlib.h&gt;

38706        unsigned short \*seed48(unsigned short *seed16v*[3]);

38707

38708 **DESCRIPTION**38709        Refer to *drand48()*.

38710 **NAME**

38711 seekdir — set position of directory stream

38712 **SYNOPSIS**

38713 XSI #include &lt;dirent.h&gt;

38714 void seekdir(DIR \*dirp, long loc);

38715

38716 **DESCRIPTION**

38717 The *seekdir()* function sets the position of the next *readdir()* operation on the directory stream  
38718 specified by *dirp* to the position specified by *loc*. The value of *loc* should have been returned  
38719 from an earlier call to *telldir()*. The new position reverts to the one associated with the directory  
38720 stream when *telldir()* was performed.

38721 If the value of *loc* was not obtained from an earlier call to *telldir()*, or if a call to *rewinddir()*  
38722 occurred between the call to *telldir()* and the call to *seekdir()*, the results of subsequent calls to  
38723 *readdir()* are unspecified.

38724 **RETURN VALUE**38725 The *seekdir()* function shall return no value.38726 **ERRORS**

38727 No errors are defined.

38728 **EXAMPLES**

38729 None.

38730 **APPLICATION USAGE**

38731 None.

38732 **RATIONALE**

38733 None.

38734 **FUTURE DIRECTIONS**

38735 None.

38736 **SEE ALSO**38737 *opendir()*, *readdir()*, *telldir()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <dirent.h>,

38738 &lt;stdio.h&gt;, &lt;sys/types.h&gt;

38739 **CHANGE HISTORY**

38740 First released in Issue 2.

38741 **Issue 4**

38742 The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
38743 XSI-conformant systems.

38744 The type of argument *loc* is expanded to **long**.38745 **Issue 4, Version 2**

38746 The DESCRIPTION is updated for X/OPEN UNIX conformance to indicate that a call to  
38747 *readdir()* may produce unspecified results if either *loc* was not obtained by a previous call to  
38748 *telldir()*, or if there is an intervening call to *rewinddir()*.

38749 **Issue 6**

38750 In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

38751 **NAME**

38752       select — synchronous I/O multiplexing

38753 **SYNOPSIS**

38754       #include &lt;sys/time.h&gt;

38755       int select(int *nfds*, fd\_set \*restrict *readfds*, fd\_set \*restrict *writefds*,38756                fd\_set \*restrict *errorfds*, struct timeval \*restrict *timeout*);

38757

38758 **DESCRIPTION**38759       Refer to *pselect()*.

## 38760 NAME

38761 sem\_close — close a named semaphore (**REALTIME**)

## 38762 SYNOPSIS

```
38763 SEM #include <semaphore.h>
```

```
38764 int sem_close(sem_t *sem);
```

38765

## 38766 DESCRIPTION

38767 The *sem\_close()* function is used to indicate that the calling process is finished using the named  
38768 semaphore indicated by *sem*. The effects of calling *sem\_close()* for an unnamed semaphore (one  
38769 created by *sem\_init()*) are undefined. The *sem\_close()* function deallocates (that is, makes  
38770 available for reuse by a subsequent *sem\_open()* by this process) any system resources allocated  
38771 by the system for use by this process for this semaphore. The effect of subsequent use of the  
38772 semaphore indicated by *sem* by this process is undefined. If the semaphore has not been  
38773 removed with a successful call to *sem\_unlink()*, then *sem\_close()* has no effect on the state of the  
38774 semaphore. If the *sem\_unlink()* function has been successfully invoked for *name* after the most  
38775 recent call to *sem\_open()* with *O\_CREAT* for this semaphore, then when all processes that have  
38776 opened the semaphore close it, the semaphore is no longer accessible.

## 38777 RETURN VALUE

38778 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be  
38779 returned and *errno* set to indicate the error.

## 38780 ERRORS

38781 The *sem\_close()* function shall fail if:

38782 [EINVAL] The *sem* argument is not a valid semaphore descriptor.

## 38783 EXAMPLES

38784 None.

## 38785 APPLICATION USAGE

38786 The *sem\_close()* function is part of the Semaphores option and need not be available on all  
38787 implementations.

## 38788 RATIONALE

38789 None.

## 38790 FUTURE DIRECTIONS

38791 None.

## 38792 SEE ALSO

38793 *semctl()*, *semget()*, *semop()*, *sem\_init()*, *sem\_open()*, *sem\_unlink()*, the Base Definitions volume of  
38794 IEEE Std. 1003.1-200x, <**semaphore.h**>

## 38795 CHANGE HISTORY

38796 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

## 38797 Issue 6

38798 The *sem\_close()* function is marked as part of the Semaphores option.

38799 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38800 implementation does not support the Semaphores option.

38801 **NAME**38802 sem\_destroy — destroy an unnamed semaphore (**REALTIME**)38803 **SYNOPSIS**

38804 SEM #include &lt;semaphore.h&gt;

38805 int sem\_destroy(sem\_t \*sem);

38806

38807 **DESCRIPTION**

38808 The *sem\_destroy()* function is used to destroy the unnamed semaphore indicated by *sem*. Only a  
 38809 semaphore that was created using *sem\_init()* may be destroyed using *sem\_destroy()*; the effect of  
 38810 calling *sem\_destroy()* with a named semaphore is undefined. The effect of subsequent use of the  
 38811 semaphore *sem* is undefined until *sem* is re-initialized by another call to *sem\_init()*.

38812 It is safe to destroy an initialized semaphore upon which no threads are currently blocked. The  
 38813 effect of destroying a semaphore upon which other threads are currently blocked is undefined.

38814 **RETURN VALUE**

38815 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be  
 38816 returned and *errno* set to indicate the error.

38817 **ERRORS**38818 The *sem\_destroy()* function shall fail if:38819 [EINVAL] The *sem* argument is not a valid semaphore.38820 The *sem\_destroy()* function may fail if:

38821 [EBUSY] There are currently processes blocked on the semaphore.

38822 **EXAMPLES**

38823 None.

38824 **APPLICATION USAGE**

38825 The *sem\_destroy()* function is part of the Semaphores option and need not be available on all  
 38826 implementations.

38827 **RATIONALE**

38828 None.

38829 **FUTURE DIRECTIONS**

38830 None.

38831 **SEE ALSO**

38832 *semctl()*, *semget()*, *semop()*, *sem\_init()*, *sem\_open()*, the Base Definitions volume of  
 38833 IEEE Std. 1003.1-200x, <**semaphore.h**>

38834 **CHANGE HISTORY**

38835 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38836 **Issue 6**38837 The *sem\_destroy()* function is marked as part of the Semaphores option.

38838 The [ENOSYS] error condition has been removed as stubs need not be provided if an

38839 implementation does not support the Semaphores option.

38840 **NAME**

38841 sem\_getvalue — get the value of a semaphore (**REALTIME**)

38842 **SYNOPSIS**

```
38843 SEM #include <semaphore.h>
```

```
38844 int sem_getvalue(sem_t *restrict sem, int *restrict sval);
```

38845

38846 **DESCRIPTION**

38847 The *sem\_getvalue()* function updates the location referenced by the *sval* argument to have the  
38848 value of the semaphore referenced by *sem* without affecting the state of the semaphore. The  
38849 updated value represents an actual semaphore value that occurred at some unspecified time  
38850 during the call, but it need not be the actual value of the semaphore when it is returned to the  
38851 calling process.

38852 If *sem* is locked, then the value returned by *sem\_getvalue()* is either zero or a negative number  
38853 whose absolute value represents the number of processes waiting for the semaphore at some  
38854 unspecified time during the call.

38855 **RETURN VALUE**

38856 Upon successful completion, the *sem\_getvalue()* function shall return a value of zero. Otherwise,  
38857 it shall return a value of -1 and set *errno* to indicate the error.

38858 **ERRORS**

38859 The *sem\_getvalue()* function shall fail if:

38860 [EINVAL] The *sem* argument does not refer to a valid semaphore.

38861 **EXAMPLES**

38862 None.

38863 **APPLICATION USAGE**

38864 The *sem\_getvalue()* function is part of the Semaphores option and need not be available on all  
38865 implementations.

38866 **RATIONALE**

38867 None.

38868 **FUTURE DIRECTIONS**

38869 None.

38870 **SEE ALSO**

38871 *semctl()*, *semget()*, *semop()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_wait()*, the Base  
38872 Definitions volume of IEEE Std. 1003.1-200x, <**semaphore.h**>

38873 **CHANGE HISTORY**

38874 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38875 **Issue 6**

38876 The *sem\_getvalue()* function is marked as part of the Semaphores option.

38877 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38878 implementation does not support the Semaphores option.

38879 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
38880 IEEE Std. 1003.1d-1999.

38881 The **restrict** keyword is added to the *sem\_getvalue()* prototype for alignment with the  
38882 ISO/IEC 9899:1999 standard.



38883 **NAME**38884 sem\_init — initialize an unnamed semaphore (**REALTIME**)38885 **SYNOPSIS**

38886 SEM #include &lt;semaphore.h&gt;

38887 int sem\_init(sem\_t \*sem, int pshared, unsigned value);

38888

38889 **DESCRIPTION**

38890 The *sem\_init()* function is used to initialize the unnamed semaphore referred to by *sem*. The  
 38891 value of the initialized semaphore is *value*. Following a successful call to *sem\_init()*, the  
 38892 semaphore may be used in subsequent calls to *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and  
 38893 *sem\_destroy()*. This semaphore remains usable until the semaphore is destroyed.

38894 If the *pshared* argument has a non-zero value, then the semaphore is shared between processes;  
 38895 in this case, any process that can access the semaphore *sem* can use *sem* for performing  
 38896 *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and *sem\_destroy()* operations.

38897 Only *sem* itself may be used for performing synchronization. The result of referring to copies of  
 38898 *sem* in calls to *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and *sem\_destroy()*, is undefined.

38899 If the *pshared* argument is zero, then the semaphore is shared between threads of the process; any  
 38900 thread in this process can use *sem* for performing *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and  
 38901 *sem\_destroy()* operations. The use of the semaphore by threads other than those created in the  
 38902 same process is undefined.

38903 Attempting to initialize an already initialized semaphore results in undefined behavior.

38904 **RETURN VALUE**

38905 Upon successful completion, the *sem\_init()* function shall initialize the semaphore in *sem*.  
 38906 Otherwise, it shall return  $-1$  and set *errno* to indicate the error.

38907 **ERRORS**

38908 The *sem\_init()* function shall fail if:

38909 [EINVAL] The *value* argument exceeds {SEM\_VALUE\_MAX}.

38910 [ENOSPC] A resource required to initialize the semaphore has been exhausted, or the  
 38911 limit on semaphores ({SEM\_NSEMS\_MAX}) has been reached.

38912 [EPERM] The process lacks the appropriate privileges to initialize the semaphore.

38913 **EXAMPLES**

38914 None.

38915 **APPLICATION USAGE**

38916 The *sem\_init()* function is part of the Semaphores option and need not be available on all  
 38917 implementations.

38918 **RATIONALE**

38919 Although this volume of IEEE Std. 1003.1-200x fails to specify a successful return value, it is  
 38920 likely that a later version may require the implementation to return a value of zero if the call to  
 38921 *sem\_init()* is successful.

38922 **FUTURE DIRECTIONS**

38923 None.

38924 **SEE ALSO**

38925 *sem\_destroy()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_wait()*, the Base Definitions  
38926 volume of IEEE Std. 1003.1-200x, <semaphore.h>

38927 **CHANGE HISTORY**

38928 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38929 **Issue 6**

38930 The *sem\_init()* function is marked as part of the Semaphores option.

38931 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38932 implementation does not support the Semaphores option.

38933 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
38934 IEEE Std. 1003.1d-1999.

38935 **NAME**38936 sem\_open — initialize and open a named semaphore (**REALTIME**)38937 **SYNOPSIS**

38938 SEM #include &lt;semaphore.h&gt;

38939 sem\_t \*sem\_open(const char \*name, int oflag, ...);

38940

38941 **DESCRIPTION**

38942 The *sem\_open()* function establishes a connection between a named semaphore and a process.  
 38943 Following a call to *sem\_open()* with semaphore name *name*, the process may reference the  
 38944 semaphore associated with *name* using the address returned from the call. This semaphore may  
 38945 be used in subsequent calls to *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and *sem\_close()*. The  
 38946 semaphore remains usable by this process until the semaphore is closed by a successful call to  
 38947 *sem\_close()*, *\_exit()*, or one of the *exec* functions.

38948 The *oflag* argument controls whether the semaphore is created or merely accessed by the call to  
 38949 *sem\_open()*. The following flag bits may be set in *oflag*:

38950 **O\_CREAT** This flag is used to create a semaphore if it does not already exist. If **O\_CREAT**  
 38951 is set and the semaphore already exists, then **O\_CREAT** has no effect, except as noted  
 38952 under **O\_EXCL**. Otherwise, *sem\_open()* creates a named semaphore. The **O\_CREAT**  
 38953 flag requires a third and a fourth argument: *mode*, which is of type **mode\_t**, and  
 38954 *value*, which is of type **unsigned**. The semaphore is created with an initial value of  
 38955 *value*. Valid initial values for semaphores are less than or equal to  
 38956 {SEM\_VALUE\_MAX}.

38957 The user ID of the semaphore is set to the effective user ID of the process; the  
 38958 group ID of the semaphore is set to a system default group ID or to the effective  
 38959 group ID of the process. The permission bits of the semaphore are set to the value  
 38960 of the *mode* argument except those set in the file mode creation mask of the  
 38961 process. When bits in *mode* other than the file permission bits are specified, the  
 38962 effect is unspecified.

38963 After the semaphore named *name* has been created by *sem\_open()* with the  
 38964 **O\_CREAT** flag, other processes can connect to the semaphore by calling  
 38965 *sem\_open()* with the same value of *name*.

38966 **O\_EXCL** If **O\_EXCL** and **O\_CREAT** are set, *sem\_open()* fails if the semaphore *name* exists.  
 38967 The check for the existence of the semaphore and the creation of the semaphore if  
 38968 it does not exist are atomic with respect to other processes executing *sem\_open()*  
 38969 with **O\_EXCL** and **O\_CREAT** set. If **O\_EXCL** is set and **O\_CREAT** is not set, the  
 38970 effect is undefined.

38971 If flags other than **O\_CREAT** and **O\_EXCL** are specified in the *oflag* parameter, the  
 38972 effect is unspecified.

38973 The *name* argument points to a string naming a semaphore object. It is unspecified whether the  
 38974 name appears in the file system and is visible to functions that take path names as arguments.  
 38975 The *name* argument conforms to the construction rules for a path name. If *name* begins with the  
 38976 slash character, then processes calling *sem\_open()* with the same value of *name* shall refer to the  
 38977 same semaphore object, as long as that name has not been removed. If *name* does not begin with  
 38978 the slash character, the effect is implementation-defined. The interpretation of slash characters  
 38979 other than the leading slash character in *name* is implementation-defined.

38980 If a process makes multiple successful calls to *sem\_open()* with the same value for *name*, the  
 38981 same semaphore address is returned for each such successful call, provided that there have been

38982 no calls to *sem\_unlink()* for this semaphore.

38983 References to copies of the semaphore produce undefined results.

#### 38984 RETURN VALUE

38985 Upon successful completion, the *sem\_open()* function shall return the address of the semaphore.  
 38986 Otherwise, it shall return a value of SEM\_FAILED and set *errno* to indicate the error. The symbol  
 38987 SEM\_FAILED is defined in the header <semaphore.h>. No successful return from *sem\_open()*  
 38988 shall return the value SEM\_FAILED.

#### 38989 ERRORS

38990 If any of the following conditions occur, the *sem\_open()* function shall return SEM\_FAILED and  
 38991 set *errno* to the corresponding value:

38992 [EACCES] The named semaphore exists and the permissions specified by *oflag* are  
 38993 denied, or the named semaphore does not exist and permission to create the  
 38994 named semaphore is denied.

38995 [EEXIST] O\_CREAT and O\_EXCL are set and the named semaphore already exists.

38996 [EINTR] The *sem\_open()* operation was interrupted by a signal.

38997 [EINVAL] The *sem\_open()* operation is not supported for the given name, or O\_CREAT  
 38998 was specified in *oflag* and *value* was greater than {SEM\_VALUE\_MAX}.

38999 [EMFILE] Too many semaphore descriptors or file descriptors are currently in use by  
 39000 this process.

39001 [ENAMETOOLONG]

39002 The length of the *name* argument exceeds {PATH\_MAX} or a path name  
 39003 component is longer than {NAME\_MAX}.

39004 [ENFILE] Too many semaphores are currently open in the system.

39005 [ENOENT] O\_CREAT is not set and the named semaphore does not exist.

39006 [ENOSPC] There is insufficient space for the creation of the new named semaphore.

#### 39007 EXAMPLES

39008 None.

#### 39009 APPLICATION USAGE

39010 The *sem\_open()* function is part of the Semaphores option and need not be available on all  
 39011 implementations.

#### 39012 RATIONALE

39013 An earlier version of this volume of IEEE Std. 1003.1-200x required an error return value of -1  
 39014 with the type **sem\_t\*** for the *sem\_open()* function, which is not guaranteed to be portable across  
 39015 implementations. The revised text provides the symbolic error code SEM\_FAILED to eliminate  
 39016 the type conflict.

#### 39017 FUTURE DIRECTIONS

39018 None.

#### 39019 SEE ALSO

39020 *semctl()*, *semget()*, *semop()*, *sem\_close()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_unlink()*,  
 39021 *sem\_wait()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <semaphore.h>

39022 **CHANGE HISTORY**

39023 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39024 **Issue 6**

39025 The *sem\_open()* function is marked as part of the Semaphores option.

39026 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
39027 implementation does not support the Semaphores option.

39028 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
39029 IEEE Std. 1003.1d-1999.

## 39030 NAME

39031 sem\_post — unlock a semaphore (**REALTIME**)

## 39032 SYNOPSIS

39033 SEM #include <semaphore.h>

39034 int sem\_post(sem\_t \*sem);

39035

## 39036 DESCRIPTION

39037 The *sem\_post()* function unlocks the semaphore referenced by *sem* by performing a semaphore  
39038 unlock operation on that semaphore.

39039 If the semaphore value resulting from this operation is positive, then no threads were blocked  
39040 waiting for the semaphore to become unlocked; the semaphore value is simply incremented.

39041 If the value of the semaphore resulting from this operation is zero, then one of the threads  
39042 blocked waiting for the semaphore shall be allowed to return successfully from its call to  
39043 *sem\_wait()*. If the Process Scheduling option is supported, the thread to be unblocked shall be  
39044 chosen in a manner appropriate to the scheduling policies and parameters in effect for the  
39045 blocked threads. In the case of the schedulers SCHED\_FIFO and SCHED\_RR, the highest  
39046 priority waiting thread shall be unblocked, and if there is more than one highest priority thread  
39047 blocked waiting for the semaphore, then the highest priority thread that has been waiting the  
39048 longest shall be unblocked. If the Process Scheduling option is not defined, the choice of a thread  
39049 to unblock is unspecified.

39050 SS If the Process Sporadic Server option is supported, and the scheduling policy is  
39051 SCHED\_SPORADIC, the semantics are as per SCHED\_FIFO above.

39052 The *sem\_post()* function shall be reentrant with respect to signals and may be invoked from a  
39053 signal-catching function.

## 39054 RETURN VALUE

39055 If successful, the *sem\_post()* function shall return zero; otherwise, the function shall return  $-1$   
39056 and set *errno* to indicate the error.

## 39057 ERRORS

39058 The *sem\_post()* function shall fail if:

39059 [EINVAL] The *sem* argument does not refer to a valid semaphore.

## 39060 EXAMPLES

39061 None.

## 39062 APPLICATION USAGE

39063 The *sem\_post()* function is part of the Semaphores option and need not be available on all  
39064 implementations.

## 39065 RATIONALE

39066 None.

## 39067 FUTURE DIRECTIONS

39068 None.

## 39069 SEE ALSO

39070 *semctl()*, *semget()*, *semop()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_wait()*, the Base Definitions  
39071 volume of IEEE Std. 1003.1-200x, <semaphore.h>

39072 **CHANGE HISTORY**

39073 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39074 **Issue 6**

39075 The *sem\_post()* function is marked as part of the Semaphores option.

39076 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
39077 implementation does not support the Semaphores option.

39078 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
39079 IEEE Std. 1003.1d-1999.

39080 SCHED\_SPORADIC is added to the list of scheduling policies for which the thread that is to be  
39081 unblocked is specified for alignment with IEEE Std. 1003.1d-1999.

## 39082 NAME

39083 sem\_timedwait — lock a semaphore (**REALTIME**)

## 39084 SYNOPSIS

39085 SEM TMO #include &lt;semaphore.h&gt;

39086 #include &lt;time.h&gt;

39087 int sem\_timedwait(sem\_t \*restrict sem,

39088 const struct timespec \*restrict abs\_timeout);

39089

## 39090 DESCRIPTION

39091 The *sem\_timedwait()* function locks the semaphore referenced by *sem* as in the *sem\_wait()*  
39092 function. However, if the semaphore cannot be locked without waiting for another process or  
39093 thread to unlock the semaphore by performing a *sem\_post()* function, this wait shall be  
39094 terminated when the specified timeout expires.

39095 The timeout expires when the absolute time specified by *abs\_timeout* passes, as measured by the  
39096 clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
39097 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already been passed at the time  
39098 of the call. If the Timers option is supported, the timeout is based on the CLOCK\_REALTIME  
39099 clock; if the Timers option is not supported, the timeout is based on the system clock as returned  
39100 by the *time()* function.

## 39101 RETURN VALUE

39102 The *sem\_timedwait()* function shall return zero if the calling process successfully performed the  
39103 semaphore lock operation on the semaphore designated by *sem*. If the call was unsuccessful, the  
39104 state of the semaphore shall be unchanged, and the function shall return a value of -1 and set  
39105 *errno* to indicate the error.

## 39106 ERRORS

39107 The *sem\_timedwait()* function shall fail if:39108 [EINVAL] The *sem* argument does not refer to a valid semaphore.39109 [EINVAL] The process or thread would have blocked, and the *abs\_timeout* parameter  
39110 specified a nanoseconds field value less than zero or greater than or equal to  
39111 1 000 million.

39112 [ETIMEDOUT] The semaphore could not be locked before the specified timeout expired.

39113 The *sem\_timedwait()* function may fail if:

39114 [EDEADLK] A deadlock condition was detected.

39115 [EINTR] A signal interrupted this function.

## 39116 EXAMPLES

39117 None.

## 39118 APPLICATION USAGE

39119 Applications using these functions may be subject to priority inversion, as discussed in the Base  
39120 Definitions volume of IEEE Std. 1003.1-200x, Section 3.287, Priority Inversion.39121 The *sem\_timedwait()* function is part of the Semaphores and Timeouts options and need not be  
39122 provided on all implementations.



39123 **RATIONALE**

39124 None.

39125 **FUTURE DIRECTIONS**

39126 None.

39127 **SEE ALSO**39128 *sem\_post()*, *sem\_trywait()*, *sem\_wait()*, *semctl()*, *semget()*, *semop()*, *time()*, the Base Definitions |39129 volume of IEEE Std. 1003.1-200x, <**semaphore.h**>, <**time.h**> |39130 **CHANGE HISTORY**

39131 First released in Issue 6. Derived from IEEE Std. 1003.1d-1999.

39132 **NAME**39133 sem\_trywait, sem\_wait — lock a semaphore (**REALTIME**)39134 **SYNOPSIS**

39135 SEM #include &lt;semaphore.h&gt;

39136 int sem\_trywait(sem\_t \*sem);

39137 int sem\_wait(sem\_t \*sem);

39138

39139 **DESCRIPTION**

39140 The *sem\_trywait()* function locks the semaphore referenced by *sem* only if the semaphore is  
39141 currently not locked; that is, if the semaphore value is currently positive. Otherwise, it does not  
39142 lock the semaphore.

39143 The *sem\_wait()* function locks the semaphore referenced by *sem* by performing a semaphore lock  
39144 operation on that semaphore. If the semaphore value is currently zero, then the calling thread  
39145 shall not return from the call to *sem\_wait()* until it either locks the semaphore or the call is  
39146 interrupted by a signal.

39147 Upon successful return, the state of the semaphore shall be locked and shall remain locked until  
39148 the *sem\_post()* function is executed and returns successfully.

39149 The *sem\_wait()* function is interruptible by the delivery of a signal.

39150 **RETURN VALUE**

39151 The *sem\_trywait()* and *sem\_wait()* functions shall return zero if the calling process successfully  
39152 performed the semaphore lock operation on the semaphore designated by *sem*. If the call was  
39153 unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a  
39154 value of  $-1$  and set *errno* to indicate the error.

39155 **ERRORS**

39156 The *sem\_trywait()* and *sem\_wait()* functions shall fail if:

39157 [EAGAIN] The semaphore was already locked, so it cannot be immediately locked by the  
39158 *sem\_trywait()* operation (*sem\_trywait()* only).

39159 [EINVAL] The *sem* argument does not refer to a valid semaphore.

39160 The *sem\_trywait()* and *sem\_wait()* functions may fail if:

39161 [EDEADLK] A deadlock condition was detected.

39162 [EINTR] A signal interrupted this function.

39163 **EXAMPLES**

39164 None.

39165 **APPLICATION USAGE**

39166 Applications using these functions may be subject to priority inversion, as discussed in the Base  
39167 Definitions volume of IEEE Std. 1003.1-200x, Section 3.287, Priority Inversion.

39168 The *sem\_trywait()* and *sem\_wait()* functions are part of the Semaphores option and need not be  
39169 provided on all implementations.

39170 **RATIONALE**

39171 None.

39172 **FUTURE DIRECTIONS**

39173 None.

39174 **SEE ALSO**

39175 *semctl()*, *semget()*, *semop()*, *sem\_post()*, *sem\_timedwait()*, the Base Definitions volume of  
39176 IEEE Std. 1003.1-200x, <**semaphore.h**>

39177 **CHANGE HISTORY**

39178 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39179 **Issue 6**39180 The *sem\_trywait()* and *sem\_wait()* functions are marked as part of the Semaphores option.

39181 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
39182 implementation does not support the Semaphores option.

39183 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
39184 IEEE Std. 1003.1d-1999.

39185 **NAME**39186 sem\_unlink — remove a named semaphore (**REALTIME**)39187 **SYNOPSIS**

39188 SEM #include &lt;semaphore.h&gt;

39189 int sem\_unlink(const char \*name);

39190

39191 **DESCRIPTION**

39192 The *sem\_unlink()* function removes the semaphore named by the string *name*. If the semaphore  
39193 named by *name* is currently referenced by other processes, then *sem\_unlink()* has no effect on the  
39194 state of the semaphore. If one or more processes have the semaphore open when *sem\_unlink()* is  
39195 called, destruction of the semaphore is postponed until all references to the semaphore have  
39196 been destroyed by calls to *sem\_close()*, *\_exit()*, or *exec*. Calls to *sem\_open()* to recreate or  
39197 reconnect to the semaphore refer to a new semaphore after *sem\_unlink()* is called. The  
39198 *sem\_unlink()* call does not block until all references have been destroyed; it shall return  
39199 immediately.

39200 **RETURN VALUE**

39201 Upon successful completion, the *sem\_unlink()* function shall return a value of 0. Otherwise, the  
39202 semaphore shall not be changed and the function shall return a value of -1 and set *errno* to  
39203 indicate the error.

39204 **ERRORS**39205 The *sem\_unlink()* function shall fail if:

39206 [EACCES] Permission is denied to unlink the named semaphore.

39207 [ENAMETOOLONG]

39208 The length of the *name* argument exceeds {PATH\_MAX} or a path name  
39209 component is longer than {NAME\_MAX}.

39210 [ENOENT] The named semaphore does not exist.

39211 **EXAMPLES**

39212 None.

39213 **APPLICATION USAGE**

39214 The *sem\_unlink()* function is part of the Semaphores option and need not be available on all  
39215 implementations.

39216 **RATIONALE**

39217 None.

39218 **FUTURE DIRECTIONS**

39219 None.

39220 **SEE ALSO**

39221 *semctl()*, *semget()*, *semop()*, *sem\_close()*, *sem\_open()*, the Base Definitions volume of  
39222 IEEE Std. 1003.1-200x, <semaphore.h>

39223 **CHANGE HISTORY**

39224 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39225 **Issue 6**39226 The *sem\_unlink()* function is marked as part of the Semaphores option.

39227 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
39228 implementation does not support the Semaphores option.



39229 **NAME**

39230 sem\_wait — lock a semaphore (**REALTIME**)

39231 **SYNOPSIS**

39232 SEM #include <semaphore.h>

39233 int sem\_wait(sem\_t \*sem);

39234

39235 **DESCRIPTION**

39236 Refer to *sem\_trywait()*.

39237 **NAME**

39238 semctl — XSI semaphore control operations

39239 **SYNOPSIS**39240 XSI 

```
#include <sys/sem.h>
```

39241 

```
int semctl(int semid, int semnum, int cmd, ...);
```

39242

39243 **DESCRIPTION**

39244 The *semctl()* function operates on XSI semaphores (see the Base Definitions volume of  
 39245 IEEE Std. 1003.1-200x, Section 4.13, Semaphore). It is unspecified whether this function  
 39246 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 39247 page 543).

39248 The *semctl()* function provides a variety of semaphore control operations as specified by *cmd*.  
 39249 The fourth argument is optional and depends upon the operation requested. If required, it is of  
 39250 type **union semun**, which the application shall explicitly declare:

```
39251 union semun {
39252 int val;
39253 struct semid_ds *buf;
39254 unsigned short *array;
39255 } arg;
```

39256 The following semaphore control operations as specified by *cmd* are executed with respect to the  
 39257 semaphore specified by *semid* and *semnum*. The level of permission required for each operation  
 39258 is shown with each command; see Section 2.7 (on page 541). The symbolic names for the values  
 39259 of *cmd* are defined by the `<sys/sem.h>` header:

39260 **GETVAL** Return the value of *semval*; see `<sys/sem.h>`. Requires read permission.

39261 **SETVAL** Set the value of *semval* to *arg.val*, where *arg* is the value of the fourth argument  
 39262 to *semctl()*. When this command is successfully executed, the *semadj* value  
 39263 corresponding to the specified semaphore in all processes is cleared. Requires  
 39264 alter permission; see Section 2.7 (on page 541).

39265 **GETPID** Return the value of *sempid*. Requires read permission.

39266 **GETNCNT** Return the value of *semmcnt*. Requires read permission.

39267 **GETZCNT** Return the value of *semzcnt*. Requires read permission.

39268 The following values of *cmd* operate on each *semval* in the set of semaphores:

39269 **GETALL** Return the value of *semval* for each semaphore in the semaphore set and place  
 39270 into the array pointed to by *arg.array*, where *arg* is the fourth argument to  
 39271 *semctl()*. Requires read permission.

39272 **SETALL** Set the value of *semval* for each semaphore in the semaphore set according to  
 39273 the array pointed to by *arg.array*, where *arg* is the fourth argument to *semctl()*.  
 39274 When this command is successfully executed, the *semadj* values corresponding  
 39275 to each specified semaphore in all processes are cleared. Requires alter  
 39276 permission.

39277 The following values of *cmd* are also available:

39278 **IPC\_STAT** Place the current value of each member of the **semid\_ds** data structure  
 39279 associated with *semid* into the structure pointed to by *arg.buf*, where *arg* is the  
 39280 fourth argument to *semctl()*. The contents of this structure are defined in

|       |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 39281 |                     | <sys/sem.h>. Requires read permission.                                                                                                                                                                                                                                                                                                                                                                                                            |
| 39282 | IPC_SET             | Set the value of the following members of the <b>semid_ds</b> data structure associated with <i>semid</i> to the corresponding value found in the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> :                                                                                                                                                                                           |
| 39283 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39284 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39285 |                     | <i>sem_perm.uid</i>                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 39286 |                     | <i>sem_perm.gid</i>                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 39287 |                     | <i>sem_perm.mode</i>                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 39288 |                     | The mode bits specified in Section 2.7.1 (on page 541) are copied into the corresponding bits of the <i>sem_perm.mode</i> associated with <i>semid</i> . The stored values of any other bits are unspecified.                                                                                                                                                                                                                                     |
| 39289 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39290 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39291 |                     | This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the <b>semid_ds</b> data structure associated with <i>semid</i> .                                                                                                                                                                    |
| 39292 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39293 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39294 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39295 | IPC_RMID            | Remove the semaphore identifier specified by <i>semid</i> from the system and destroy the set of semaphores and <b>semid_ds</b> data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the <b>semid_ds</b> data structure associated with <i>semid</i> . |
| 39296 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39297 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39298 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39299 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39300 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39301 | <b>RETURN VALUE</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39302 |                     | If successful, the value returned by <i>semctl()</i> depends on <i>cmd</i> as follows:                                                                                                                                                                                                                                                                                                                                                            |
| 39303 | GETVAL              | The value of <i>semval</i> .                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 39304 | GETPID              | The value of <i>sempid</i> .                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 39305 | GETNCNT             | The value of <i>semmcnt</i> .                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 39306 | GETZCNT             | The value of <i>semzcnt</i> .                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 39307 | All others          | 0.                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 39308 |                     | Otherwise, <i>semctl()</i> shall return $-1$ and set <i>errno</i> to indicate the error.                                                                                                                                                                                                                                                                                                                                                          |
| 39309 | <b>ERRORS</b>       |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39310 |                     | The <i>semctl()</i> function shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                       |
| 39311 | [EACCES]            | Operation permission is denied to the calling process; see Section 2.7 (on page 541).                                                                                                                                                                                                                                                                                                                                                             |
| 39312 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39313 | [EINVAL]            | The value of <i>semid</i> is not a valid semaphore identifier, or the value of <i>semnum</i> is less than 0 or greater than or equal to <i>sem_nsems</i> , or the value of <i>cmd</i> is not a valid command.                                                                                                                                                                                                                                     |
| 39314 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39315 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39316 | [EPERM]             | The argument <i>cmd</i> is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the data structure associated with <i>semid</i> .                                                                                                                                       |
| 39317 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39318 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39319 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39320 | [ERANGE]            | The argument <i>cmd</i> is equal to SETVAL or SETALL and the value to which <i>semval</i> is to be set is greater than the system-imposed maximum.                                                                                                                                                                                                                                                                                                |
| 39321 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |



39322 **EXAMPLES**

39323 None.

39324 **APPLICATION USAGE**

39325 The fourth parameter in the SYNOPSIS section is now specified as ". . ." in order to avoid a  
 39326 clash with the ISO C standard when referring to the union *semun* (as defined in Issue 3) and for  
 39327 backward compatibility.

39328 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
 39329 Application developers who need to use IPC should design their applications so that modules  
 39330 using the IPC routines described in Section 2.7 (on page 541) can be easily modified to use the  
 39331 alternative interfaces.

39332 **RATIONALE**

39333 None.

39334 **FUTURE DIRECTIONS**

39335 None.

39336 **SEE ALSO**

39337 *semget()*, *semop()*, *sem\_close()*, *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*,  
 39338 *sem\_unlink()*, *sem\_wait()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<sys/sem.h>`,  
 39339 Section 2.7 (on page 541)

39340 **CHANGE HISTORY**

39341 First released in Issue 2. Derived from Issue 2 of the SVID.

39342 **Issue 4**

39343 The function is no longer marked as OPTIONAL FUNCTIONALITY.

39344 The `<sys/types.h>` and `<sys/ipc.h>` headers are removed from the SYNOPSIS section.

39345 The last argument is now defined by an ellipsis symbol. In previous issues it was defined as a  
 39346 union of the various types required by settings of *cmd*. These are now defined individually in  
 39347 each description of permitted *cmd* settings. The text of the description of SETALL in the  
 39348 DESCRIPTION now refers to the fourth argument instead of *arg.buf*.

39349 In the DESCRIPTION the type of the array is specified in the descriptions of GETALL and  
 39350 SETALL.

39351 The [ENOSYS] error is removed from the ERRORS section.

39352 A FUTURE DIRECTIONS section is added warning application developers about migration to  
 39353 IEEE 1003.4 interfaces for interprocess communication.

39354 **Issue 4, Version 2**

39355 The fourth argument to *semctl()*, formerly specified in the APPLICATION USAGE section, is  
 39356 moved to the DESCRIPTION, and references to its elements are made more precise.

39357 **Issue 5**

39358 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
 39359 DIRECTIONS to the APPLICATION USAGE section.

## 39360 NAME

39361 semget — get set of XSI semaphores

## 39362 SYNOPSIS

39363 XSI #include &lt;sys/sem.h&gt;

39364 int semget(key\_t key, int nsems, int semflg);

39365

## 39366 DESCRIPTION

39367 The *semget()* function operates on XSI semaphores (see the Base Definitions volume of  
 39368 IEEE Std. 1003.1-200x, Section 4.13, Semaphore). It is unspecified whether this function  
 39369 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 39370 page 543).

39371 The *semget()* function shall return the semaphore identifier associated with *key*.

39372 A semaphore identifier with its associated **semid\_ds** data structure and its associated set of  
 39373 *nsems* semaphores (see <sys/sem.h>) is created for *key* if one of the following is true:

- 39374 • The argument *key* is equal to `IPC_PRIVATE`.
- 39375 • The argument *key* does not already have a semaphore identifier associated with it and (*semflg*  
 39376 & `IPC_CREAT`) is non-zero.

39377 Upon creation, the **semid\_ds** data structure associated with the new semaphore identifier is  
 39378 initialized as follows:

- 39379 • In the operation permissions structure *sem\_perm.cuid*, *sem\_perm.uid*, *sem\_perm.cgid*, and  
 39380 *sem\_perm.gid* are set equal to the effective user ID and effective group ID, respectively, of the  
 39381 calling process.
- 39382 • The low-order 9 bits of *sem\_perm.mode* are set equal to the low-order 9 bits of *semflg*.
- 39383 • The variable *sem\_nsems* is set equal to the value of *nsems*.
- 39384 • The variable *sem\_otime* is set equal to 0 and *sem\_ctime* is set equal to the current time.
- 39385 • The data structure associated with each semaphore in the set is not initialized. The *semctl()*  
 39386 function with the command `SETVAL` or `SETALL` can be used to initialize each semaphore.

## 39387 RETURN VALUE

39388 Upon successful completion, *semget()* shall return a non-negative integer, namely a semaphore  
 39389 identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

## 39390 ERRORS

39391 The *semget()* function shall fail if:

- |                                  |          |                                                                                                                                                                                                                                                                                                         |
|----------------------------------|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 39392<br>39393<br>39394          | [EACCES] | A semaphore identifier exists for <i>key</i> , but operation permission as specified by<br>the low-order 9 bits of <i>semflg</i> would not be granted; see Section 2.7 (on page<br>541).                                                                                                                |
| 39395<br>39396                   | [EEXIST] | A semaphore identifier exists for the argument <i>key</i> but (( <i>semflg</i> & <code>IPC_CREAT</code> )<br>&& ( <i>semflg</i> & <code>IPC_EXCL</code> )) is non-zero.                                                                                                                                 |
| 39397<br>39398<br>39399<br>39400 | [EINVAL] | The value of <i>nsems</i> is either less than or equal to 0 or greater than the system-<br>imposed limit, or a semaphore identifier exists for the argument <i>key</i> , but the<br>number of semaphores in the set associated with it is less than <i>nsems</i> and<br><i>nsems</i> is not equal to 0. |
| 39401<br>39402                   | [ENOENT] | A semaphore identifier does not exist for the argument <i>key</i> and ( <i>semflg</i><br>& <code>IPC_CREAT</code> ) is equal to 0.                                                                                                                                                                      |

39403 [ENOSPC] A semaphore identifier is to be created but the system-imposed limit on the  
 39404 maximum number of allowed semaphores system-wide would be exceeded.

### 39405 EXAMPLES

#### 39406 Creating a Semaphore Identifier

39407 The following example gets a unique semaphore key using the *ftok()* function, then gets a  
 39408 semaphore ID associated with that key using the *semget()* function (the first call also tests to  
 39409 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as  
 39410 shown by the second call to *semget()*. In creating the semaphore for the queueing process, the  
 39411 program attempts to create one semaphore with read/write permission for all. It also uses the  
 39412 *IPC\_EXCL* flag, which forces *semget()* to fail if the semaphore already exists.

39413 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in  
 39414 the *sbuf* array. The number of processes that can execute concurrently without queuing is  
 39415 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in  
 39416 the program.

```

39417 #include <sys/types.h>
39418 #include <stdio.h>
39419 #include <sys/ipc.h>
39420 #include <sys/sem.h>
39421 #include <sys/stat.h>
39422 #include <errno.h>
39423 #include <unistd.h>
39424 #include <stdlib.h>
39425 #include <pwd.h>
39426 #include <fcntl.h>
39427 #include <limits.h>
39428 ...
39429 key_t semkey;
39430 int semid, pfd, fv;
39431 struct sembuf sbuf;
39432 char *lgn;
39433 char filename[PATH_MAX+1];
39434 struct stat outstat;
39435 struct passwd *pw;
39436 ...
39437 /* Get unique key for semaphore. */
39438 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
39439 perror("IPC error: ftok"); exit(1);
39440 }
39441 /* Get semaphore ID associated with this key. */
39442 if ((semid = semget(semkey, 0, 0)) == -1) {
39443 /* Semaphore does not exist - Create. */
39444 if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
39445 S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
39446 {
39447 /* Initialize the semaphore. */
39448 sbuf.sem_num = 0;
39449 sbuf.sem_op = 2; /* This is the number of runs without queuing. */
39450 sbuf.sem_flg = 0;

```

```

39451 if (semop(semid, &sbuf, 1) == -1) {
39452 perror("IPC error: semop"); exit(1);
39453 }
39454 }
39455 else if (errno == EEXIST) {
39456 if ((semid = semget(semkey, 0, 0)) == -1) {
39457 perror("IPC error 1: semget"); exit(1);
39458 }
39459 }
39460 else {
39461 perror("IPC error 2: semget"); exit(1);
39462 }
39463 }
39464 ...

```

#### 39465 APPLICATION USAGE

39466 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
 39467 Application developers who need to use IPC should design their applications so that modules  
 39468 using the IPC routines described in Section 2.7 (on page 541) can be easily modified to use the  
 39469 alternative interfaces.

#### 39470 RATIONALE

39471 None.

#### 39472 FUTURE DIRECTIONS

39473 None.

#### 39474 SEE ALSO

39475 *semctl()*, *semop()*, *sem\_close()*, *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*,  
 39476 *sem\_unlink()*, *sem\_wait()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<sys/sem.h>`,  
 39477 Section 2.7 (on page 541).

#### 39478 CHANGE HISTORY

39479 First released in Issue 2. Derived from Issue 2 of the SVID.

#### 39480 Issue 4

39481 The function is no longer marked as OPTIONAL FUNCTIONALITY.

39482 The `<sys/types.h>` and `<sys/ipc.h>` headers are removed from the SYNOPSIS section.

39483 The [ENOSYS] error is removed from the ERRORS section.

39484 A FUTURE DIRECTIONS section is added warning application developers about migration to  
 39485 IEEE 1003.4 interfaces for interprocess communication.

#### 39486 Issue 5

39487 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
 39488 DIRECTIONS to a new APPLICATION USAGE section.

39489 **NAME**

39490 semop — XSI semaphore operations

39491 **SYNOPSIS**39492 XSI 

```
#include <sys/sem.h>
```

39493 

```
int semop(int semid, struct sembuf *sops, size_t nsops);
```

39494

39495 **DESCRIPTION**

39496 The *semop()* function operates on XSI semaphores (see the Base Definitions volume of  
 39497 IEEE Std. 1003.1-200x, Section 4.13, Semaphore). It is unspecified whether this function  
 39498 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 39499 page 543).

39500 The *semop()* function is used to perform atomically a user-defined array of semaphore  
 39501 operations on the set of semaphores associated with the semaphore identifier specified by the  
 39502 argument *semid*.

39503 The argument *sops* is a pointer to a user-defined array of semaphore operation structures. The  
 39504 implementation shall not modify elements of this array unless the application uses  
 39505 implementation-defined extensions.

39506 The argument *nsops* is the number of such structures in the array.

39507 Each structure, **sembuf**, includes the following members:

39508

39509

| Member Type | Member Name    | Description          |
|-------------|----------------|----------------------|
| short       | <i>sem_num</i> | Semaphore number.    |
| short       | <i>sem_op</i>  | Semaphore operation. |
| short       | <i>sem_flg</i> | Operation flags.     |

39510

39511

39512

39513 Each semaphore operation specified by *sem\_op* is performed on the corresponding semaphore  
 39514 specified by *semid* and *sem\_num*.

39515 The variable *sem\_op* specifies one of three semaphore operations:

39516 1. If *sem\_op* is a negative integer and the calling process has alter permission, one of the  
 39517 following shall occur:

39518 • If *semval* (see **<sys/sem.h>**) is greater than or equal to the absolute value of *sem\_op*, the  
 39519 absolute value of *sem\_op* is subtracted from *semval*. Also, if (*sem\_flg* & SEM\_UNDO) is  
 39520 non-zero, the absolute value of *sem\_op* is added to the calling process' *semadj* value for  
 39521 the specified semaphore.

39522 • If *semval* is less than the absolute value of *sem\_op* and (*sem\_flg* & IPC\_NOWAIT) is non-  
 39523 zero, *semop()* shall return immediately.

39524 • If *semval* is less than the absolute value of *sem\_op* and (*sem\_flg* & IPC\_NOWAIT) is 0,  
 39525 *semop()* shall increment the *semncnt* associated with the specified semaphore and  
 39526 suspend execution of the calling thread until one of the following conditions occurs:

39527 — The value of *semval* becomes greater than or equal to the absolute value of *sem\_op*.  
 39528 When this occurs, the value of *semncnt* associated with the specified semaphore is  
 39529 decremented, the absolute value of *sem\_op* is subtracted from *semval* and, if (*sem\_flg*  
 39530 & SEM\_UNDO) is non-zero, the absolute value of *sem\_op* is added to the calling  
 39531 process' *semadj* value for the specified semaphore.

- 39532 — The *semid* for which the calling thread is awaiting action is removed from the  
 39533 system. When this occurs, *errno* shall be set equal to [EIDRM] and  $-1$  shall be  
 39534 returned.
- 39535 — The calling thread receives a signal that is to be caught. When this occurs, the value  
 39536 of *semncnt* associated with the specified semaphore is decremented, and the calling  
 39537 thread resumes execution in the manner prescribed in *sigaction()*.
- 39538 2. If *sem\_op* is a positive integer and the calling process has alter permission, the value of  
 39539 *sem\_op* is added to *semval* and, if (*sem\_flg* & SEM\_UNDO) is non-zero, the value of *sem\_op* is  
 39540 subtracted from the calling process' *semadj* value for the specified semaphore.
- 39541 3. If *sem\_op* is 0 and the calling process has read permission, one of the following shall occur:
- 39542 • If *semval* is 0, *semop()* shall return immediately.
- 39543 • If *semval* is non-zero and (*sem\_flg* & IPC\_NOWAIT) is non-zero, *semop()* shall return  
 39544 immediately.
- 39545 • If *semval* is non-zero and (*sem\_flg* & IPC\_NOWAIT) is 0, *semop()* will increment the  
 39546 *semzcnt* associated with the specified semaphore and suspends execution of the calling  
 39547 thread until one of the following occurs:
- 39548 — The value of *semval* becomes 0, at which time the value of *semzcnt* associated with  
 39549 the specified semaphore is decremented.
- 39550 — The *semid* for which the calling thread is awaiting action is removed from the  
 39551 system. When this occurs, *errno* shall be set equal to [EIDRM] and  $-1$  shall be  
 39552 returned.
- 39553 — The calling thread receives a signal that is to be caught. When this occurs, the value  
 39554 of *semzcnt* associated with the specified semaphore is decremented, and the calling  
 39555 thread resumes execution in the manner prescribed in *sigaction()*.
- 39556 Upon successful completion, the value of *sempid* for each semaphore specified in the array  
 39557 pointed to by *sops* shall be set equal to the process ID of the calling process.

**39558 RETURN VALUE**

39559 Upon successful completion, *semop()* shall return 0; otherwise, it shall return  $-1$  and set *errno* to  
 39560 indicate the error.

**39561 ERRORS**

39562 The *semop()* function shall fail if:

|       |          |                                                                                      |  |
|-------|----------|--------------------------------------------------------------------------------------|--|
| 39563 | [E2BIG]  | The value of <i>nsops</i> is greater than the system-imposed maximum.                |  |
| 39564 | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page      |  |
| 39565 |          | 541).                                                                                |  |
| 39566 | [EAGAIN] | The operation would result in suspension of the calling process but ( <i>sem_flg</i> |  |
| 39567 |          | & IPC_NOWAIT) is non-zero.                                                           |  |
| 39568 | [EFBIG]  | The value of <i>sem_num</i> is less than 0 or greater than or equal to the number of |  |
| 39569 |          | semaphores in the set associated with <i>semid</i> .                                 |  |
| 39570 | [EIDRM]  | The semaphore identifier <i>semid</i> is removed from the system.                    |  |
| 39571 | [EINTR]  | The <i>semop()</i> function was interrupted by a signal.                             |  |
| 39572 | [EINVAL] | The value of <i>semid</i> is not a valid semaphore identifier, or the number of      |  |
| 39573 |          | individual semaphores for which the calling process requests a SEM_UNDO              |  |
| 39574 |          | would exceed the system-imposed limit.                                               |  |

39575 [ENOSPC] The limit on the number of individual processes requesting a SEM\_UNDO |  
 39576 would be exceeded.

39577 [ERANGE] An operation would cause a *semval* to overflow the system-imposed limit, or |  
 39578 an operation would cause a *semadj* value to overflow the system-imposed |  
 39579 limit.

### 39580 EXAMPLES

#### 39581 Setting Values in Semaphores

39582 The following example sets the values of the two semaphores associated with the *semid*  
 39583 identifier to the values contained in the *sb* array.

```
39584 #include <sys/sem.h>
39585 ...
39586 int semid;
39587 struct sembuf sb[2];
39588 int nsops = 2;
39589 int result;

39590 /* Adjust value of semaphore in the semaphore array semid. */
39591 sb[0].sem_num = 0;
39592 sb[0].sem_op = -1;
39593 sb[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
39594 sb[1].sem_num = 1;
39595 sb[1].sem_op = 1;
39596 sb[1].sem_flg = 0;

39597 result = semop(semid, sb, nsops);
```

#### 39598 Creating a Semaphore Identifier

39599 The following example gets a unique semaphore key using the *ftok()* function, then gets a  
 39600 semaphore ID associated with that key using the *semget()* function (the first call also tests to  
 39601 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as  
 39602 shown by the second call to *semget()*. In creating the semaphore for the queueing process, the  
 39603 program attempts to create one semaphore with read/write permission for all. It also uses the  
 39604 IPC\_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

39605 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in  
 39606 the *sbuf* array. The number of processes that can execute concurrently without queueing is  
 39607 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in  
 39608 the program.

39609 The final call to *semop()* acquires the semaphore and waits until it is free; the SEM\_UNDO  
 39610 option releases the semaphore when the process exits, waiting until there are less than two  
 39611 processes running concurrently.

```
39612 #include <sys/types.h>
39613 #include <stdio.h>
39614 #include <sys/ipc.h>
39615 #include <sys/sem.h>
39616 #include <sys/stat.h>
39617 #include <errno.h>
39618 #include <unistd.h>
39619 #include <stdlib.h>
```

```

39620 #include <pwd.h>
39621 #include <fcntl.h>
39622 #include <limits.h>
39623 ...
39624 key_t semkey;
39625 int semid, pfd, fv;
39626 struct sembuf sbuf;
39627 char *lgn;
39628 char filename[PATH_MAX+1];
39629 struct stat outstat;
39630 struct passwd *pw;
39631 ...
39632 /* Get unique key for semaphore. */
39633 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
39634 perror("IPC error: ftok"); exit(1);
39635 }
39636 /* Get semaphore ID associated with this key. */
39637 if ((semid = semget(semkey, 0, 0)) == -1) {
39638 /* Semaphore does not exist - Create. */
39639 if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
39640 S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
39641 {
39642 /* Initialize the semaphore. */
39643 sbuf.sem_num = 0;
39644 sbuf.sem_op = 2; /* This is the number of runs without queuing. */
39645 sbuf.sem_flg = 0;
39646 if (semop(semid, &sbuf, 1) == -1) {
39647 perror("IPC error: semop"); exit(1);
39648 }
39649 }
39650 else if (errno == EEXIST) {
39651 if ((semid = semget(semkey, 0, 0)) == -1) {
39652 perror("IPC error 1: semget"); exit(1);
39653 }
39654 }
39655 else {
39656 perror("IPC error 2: semget"); exit(1);
39657 }
39658 }
39659 ...
39660 sbuf.sem_num = 0;
39661 sbuf.sem_op = -1;
39662 sbuf.sem_flg = SEM_UNDO;
39663 if (semop(semid, &sbuf, 1) == -1) {
39664 perror("IPC Error: semop"); exit(1);
39665 }

```

#### 39666 APPLICATION USAGE

39667 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
39668 Application developers who need to use IPC should design their applications so that modules  
39669 using the IPC routines described in Section 2.7 (on page 541) can be easily modified to use the  
39670 alternative interfaces.



39671 **RATIONALE**

39672 None.

39673 **FUTURE DIRECTIONS**

39674 None.

39675 **SEE ALSO**

39676 *exec*, *exit()*, *fork()*, *semctl()*, *semget()*, *sem\_close()*, *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*,  
39677 *sem\_open()*, *sem\_post()*, *sem\_unlink()*, *sem\_wait()*, the Base Definitions volume of  
39678 IEEE Std. 1003.1-200x, <sys/ipc.h>, <sys/sem.h>, <sys/types.h>, Section 2.7 (on page 541)

39679 **CHANGE HISTORY**

39680 First released in Issue 2. Derived from Issue 2 of the SVID.

39681 **Issue 4**

39682 The function is no longer marked as OPTIONAL FUNCTIONALITY.

39683 The &lt;sys/types.h&gt; and &lt;sys/ipc.h&gt; headers are removed from the SYNOPSIS section.

39684 The type of *nsops* is changed to **size\_t**.

39685 The DESCRIPTION is updated to indicate that an implementation does not modify the elements  
39686 of *sops* unless the application uses implementation-defined extensions.

39687 The [ENOSYS] error is removed from the ERRORS section.

39688 A FUTURE DIRECTIONS section is added warning application developers about migration to  
39689 IEEE 1003.4 interfaces for interprocess communication.

39690 **Issue 5**

39691 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
39692 DIRECTIONS to a new APPLICATION USAGE section.

39693 **NAME**

39694 send — send a message on a socket

39695 **SYNOPSIS**

39696 #include &lt;sys/socket.h&gt;

39697 ssize\_t send(int *socket*, const void \**buffer*, size\_t *length*, int *flags*);39698 **DESCRIPTION**39699 The *send()* functions takes the following arguments:39700 *socket* Specifies the socket file descriptor.39701 *buffer* Points to the buffer containing the message to send.39702 *length* Specifies the length of the message in bytes.39703 *flags* Specifies the type of message transmission. Values of this argument are  
39704 formed by logically OR'ing zero or more of the following flags:

39705 MSG\_EOR Terminates a record (if supported by the protocol).

39706 MSG\_OOB Sends out-of-band data on sockets that support out-of-band  
39707 communications. The significance and semantics of out-of-  
39708 band data are protocol-specific.39709 The *send()* function initiates transmission of a message from the specified socket to its peer. The  
39710 *send()* function sends a message only when the socket is connected (including when the peer of a  
39711 connectionless socket has been set via *connect()*).39712 The length of the message to be sent is specified by the *length* argument. If the message is too  
39713 long to pass through the underlying protocol, *send()* shall fail and no data shall be transmitted.39714 Successful completion of a call to *send()* does not guarantee delivery of the message. A return  
39715 value of  $-1$  indicates only locally-detected errors.39716 If space is not available at the sending socket to hold the message to be transmitted, and the  
39717 socket file descriptor does not have O\_NONBLOCK set, *send()* shall block until space is  
39718 available. If space is not available at the sending socket to hold the message to be transmitted,  
39719 and the socket file descriptor does have O\_NONBLOCK set, *send()* shall fail. The *select()* and  
39720 *poll()* functions can be used to determine when it is possible to send more data.39721 The socket in use may require the process to have appropriate privileges to use the *send()*  
39722 function.39723 **RETURN VALUE**39724 Upon successful completion, *send()* shall return the number of bytes sent. Otherwise,  $-1$  shall be  
39725 returned and *errno* set to indicate the error.39726 **ERRORS**39727 The *send()* function shall fail if:

39728 [EAGAIN] or [EWOULDBLOCK]

39729 The socket's file descriptor is marked O\_NONBLOCK and the requested  
39730 operation would block.39731 [EBADF] The *socket* argument is not a valid file descriptor.

39732 [ECONNRESET] A connection was forcibly closed by a peer.

39733 [EDESTADDRREQ]

39734 The socket is not connection-mode and no peer address is set.

- 39735 [EINTR] A signal interrupted *send()* before any data was transmitted.
- 39736 [EMSGSIZE] The message is too large be sent all at once, as the socket requires.
- 39737 [ENOTCONN] The socket is not connected or otherwise has not had the peer pre-specified.
- 39738 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 39739 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or  
39740 more of the values set in *flags*.
- 39741 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is  
39742 no longer connected. In the latter case, and if the socket is of type  
39743 SOCK\_STREAM, the SIGPIPE signal is generated to the calling thread.
- 39744 The *send()* function may fail if:
- 39745 [EACCES] The calling process does not have the appropriate privileges.
- 39746 [EIO] An I/O error occurred while reading from or writing to the file system.
- 39747 [ENETDOWN] The local network interface used to reach the destination is down.
- 39748 [ENETUNREACH]  
39749 No route to the network is present.
- 39750 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 39751 **EXAMPLES**
- 39752 None.
- 39753 **APPLICATION USAGE**
- 39754 The *send()* function is identical to *sendto()* with a null pointer *dest\_len* argument, and to *write()* if  
39755 no flags are used.
- 39756 **RATIONALE**
- 39757 None.
- 39758 **FUTURE DIRECTIONS**
- 39759 None.
- 39760 **SEE ALSO**
- 39761 *connect()*, *getsockopt()*, *poll()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *sendmsg()*, *sendto()*,  
39762 *setsockopt()*, *shutdown()*, *socket()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
39763 <sys/socket.h>
- 39764 **CHANGE HISTORY**
- 39765 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

## 39766 NAME

39767 sendmsg — send a message on a socket using a message structure

## 39768 SYNOPSIS

39769 #include <sys/socket.h>

39770 ssize\_t sendmsg(int *socket*, const struct msghdr \**message*, int *flags*);

## 39771 DESCRIPTION

39772 The *sendmsg()* function sends a message through a connection-mode or connectionless-mode  
 39773 socket. If the socket is connectionless-mode, the message shall be sent to the address specified by  
 39774 **msghdr**. If the socket is connection-mode, the destination address in **msghdr** is ignored.

39775 The *sendmsg()* function takes the following arguments:

|       |                |                                                                                                                                                                                                                                                 |
|-------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 39776 | <i>socket</i>  | Specifies the socket file descriptor.                                                                                                                                                                                                           |
| 39777 | <i>message</i> | Points to a <b>msghdr</b> structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored. |
| 39780 | <i>flags</i>   | Specifies the type of message transmission. The application may specify 0 or the following flag:                                                                                                                                                |
| 39782 | MSG_EOR        | Terminates a record (if supported by the protocol).                                                                                                                                                                                             |
| 39783 | MSG_OOB        | Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol-specific.                                                                                                      |

39786 The *msg\_iov* and *msg\_iovlen* fields of *message* specify zero or more buffers containing the data to  
 39787 be sent. *msg\_iov* points to an array of **iovec** structures; *msg\_iovlen* shall be set to the dimension of  
 39788 this array. In each **iovec** structure, the *iov\_base* field specifies a storage area and the *iov\_len* field  
 39789 gives its size in bytes. Some of these sizes can be zero. The data from each storage area indicated  
 39790 by *msg\_iov* is sent in turn.

39791 Successful completion of a call to *sendmsg()* does not guarantee delivery of the message. A  
 39792 return value of  $-1$  indicates only locally-detected errors.

39793 If space is not available at the sending socket to hold the message to be transmitted and the  
 39794 socket file descriptor does not have O\_NONBLOCK set, *sendmsg()* function blocks until space is  
 39795 available. If space is not available at the sending socket to hold the message to be transmitted  
 39796 and the socket file descriptor does have O\_NONBLOCK set, *sendmsg()* function shall fail.

39797 If the socket protocol supports broadcast and the specified address is a broadcast address for the  
 39798 socket protocol, *sendmsg()* shall fail if the SO\_BROADCAST option is not set for the socket.

39799 The socket in use may require the process to have appropriate privileges to use the *sendmsg()*  
 39800 function.

## 39801 RETURN VALUE

39802 Upon successful completion, *sendmsg()* shall return the number of bytes sent. Otherwise,  $-1$   
 39803 shall be returned and *errno* set to indicate the error.

## 39804 ERRORS

39805 The *sendmsg()* function shall fail if:

39806 [EAGAIN] or [EWOULDBLOCK]

39807 The socket's file descriptor is marked O\_NONBLOCK and the requested  
 39808 operation would block.

|       |                |                                                                                                                                                                                     |
|-------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 39809 | [EAFNOSUPPORT] |                                                                                                                                                                                     |
| 39810 |                | Addresses in the specified address family cannot be used with this socket.                                                                                                          |
| 39811 | [EBADF]        | The <i>socket</i> argument is not a valid file descriptor.                                                                                                                          |
| 39812 | [ECONNRESET]   | A connection was forcibly closed by a peer.                                                                                                                                         |
| 39813 | [EINTR]        | A signal interrupted <i>sendmsg()</i> before any data was transmitted.                                                                                                              |
| 39814 | [EINVAL]       | The sum of the <i>iov_len</i> values overflows an <b>ssize_t</b> .                                                                                                                  |
| 39815 | [EMSGSIZE]     | The message is too large to be sent all at once (as the socket requires), or the <i>msg_iovlen</i> member of the <b>msg_hdr</b> structure pointed to by <i>message</i> is less than |
| 39816 |                | or equal to 0 or is greater than {IOV_MAX}.                                                                                                                                         |
| 39817 |                |                                                                                                                                                                                     |
| 39818 | [ENOTCONN]     | The socket is connection-mode but is not connected.                                                                                                                                 |
| 39819 | [ENOTSOCK]     | The <i>socket</i> argument does not refer a socket.                                                                                                                                 |
| 39820 | [EOPNOTSUPP]   | The <i>socket</i> argument is associated with a socket that does not support one or                                                                                                 |
| 39821 |                | more of the values set in <i>flags</i> .                                                                                                                                            |
| 39822 | [EPIPE]        | The socket is shut down for writing, or the socket is connection-mode and is                                                                                                        |
| 39823 |                | no longer connected. In the latter case, and if the socket is of type                                                                                                               |
| 39824 |                | SOCK_STREAM, the SIGPIPE signal is generated to the calling thread.                                                                                                                 |
| 39825 |                | If the address family of the socket is AF_UNIX, then <i>sendmsg()</i> shall fail if:                                                                                                |
| 39826 | [EIO]          | An I/O error occurred while reading from or writing to the file system.                                                                                                             |
| 39827 | [ELOOP]        | A loop exists in symbolic links encountered during resolution of the path                                                                                                           |
| 39828 |                | name in the socket address.                                                                                                                                                         |
| 39829 | [ENAMETOOLONG] |                                                                                                                                                                                     |
| 39830 |                | A component of a path name exceeded {NAME_MAX} characters, or an entire                                                                                                             |
| 39831 |                | path name exceeded {PATH_MAX} characters.                                                                                                                                           |
| 39832 | [ENOENT]       | A component of the path name does not name an existing file or the path                                                                                                             |
| 39833 |                | name is an empty string.                                                                                                                                                            |
| 39834 | [ENOTDIR]      | A component of the path prefix of the path name in the socket address is not a                                                                                                      |
| 39835 |                | directory.                                                                                                                                                                          |
| 39836 |                | The <i>sendmsg()</i> function may fail if:                                                                                                                                          |
| 39837 | [EACCES]       | Search permission is denied for a component of the path prefix; or write                                                                                                            |
| 39838 |                | access to the named socket is denied.                                                                                                                                               |
| 39839 | [EDESTADDRREQ] |                                                                                                                                                                                     |
| 39840 |                | The socket is not connection-mode and does not have its peer address set, and                                                                                                       |
| 39841 |                | no destination address was specified.                                                                                                                                               |
| 39842 | [EHOSTUNREACH] |                                                                                                                                                                                     |
| 39843 |                | The destination host cannot be reached (probably because the host is down or                                                                                                        |
| 39844 |                | a remote router cannot reach it).                                                                                                                                                   |
| 39845 | [EIO]          | An I/O error occurred while reading from or writing to the file system.                                                                                                             |
| 39846 | [EISCONN]      | A destination address was specified and the socket is already connected.                                                                                                            |
| 39847 | [ENETDOWN]     | The local network interface used to reach the destination is down.                                                                                                                  |
| 39848 | [ENETUNREACH]  |                                                                                                                                                                                     |
| 39849 |                | No route to the network is present.                                                                                                                                                 |

- 39850 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 39851 [ENOMEM] Insufficient memory was available to fulfill the request.
- 39852 If the address family of the socket is AF\_UNIX, then *sendmsg()* may fail if:
- 39853 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
39854 resolution of the path name in the socket address.
- 39855 [ENAMETOOLONG]  
39856 Path name resolution of a symbolic link produced an intermediate result  
39857 whose length exceeds {PATH\_MAX}.
- 39858 **EXAMPLES**  
39859 Done.
- 39860 **APPLICATION USAGE**  
39861 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.
- 39862 **RATIONALE**  
39863 None.
- 39864 **FUTURE DIRECTIONS**  
39865 None.
- 39866 **SEE ALSO**  
39867 *getsockopt()*, *poll()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *send()*, *sendto()*, *setsockopt()*,  
39868 *shutdown()*, *socket()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/socket.h>
- 39869 **CHANGE HISTORY**  
39870 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
- 39871 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
39872 [ELOOP] error condition is added.

39873 **NAME**39874 `sendto` — send a message on a socket39875 **SYNOPSIS**39876 `#include <sys/socket.h>`39877 `ssize_t sendto(int socket, const void *message, size_t length,`39878 `int flags, const struct sockaddr *dest_addr,`39879 `socklen_t dest_len);`39880 **DESCRIPTION**

39881 The `sendto()` function sends a message through a connection-mode or connectionless-mode  
 39882 socket. If the socket is connectionless-mode, the message shall be sent to the address specified by  
 39883 `dest_addr`. If the socket is connection-mode, `dest_addr` is ignored.

39884 The `sendto()` function takes the following arguments:

|       |                        |                                                                                                                                                                     |
|-------|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 39885 | <code>socket</code>    | Specifies the socket file descriptor.                                                                                                                               |
| 39886 | <code>message</code>   | Points to a buffer containing the message to be sent.                                                                                                               |
| 39887 | <code>length</code>    | Specifies the size of the message in bytes.                                                                                                                         |
| 39888 | <code>flags</code>     | Specifies the type of message transmission. Values of this argument are<br>39889 formed by logically OR'ing zero or more of the following flags:                    |
| 39890 | <code>MSG_EOR</code>   | Terminates a record (if supported by the protocol).                                                                                                                 |
| 39891 | <code>MSG_OOB</code>   | Sends out-of-band data on sockets that support out-of-band<br>39892 data. The significance and semantics of out-of-band data are<br>39893 protocol-specific.        |
| 39894 | <code>dest_addr</code> | Points to a <b>sockaddr</b> structure containing the destination address. The length<br>39895 and format of the address depend on the address family of the socket. |
| 39896 | <code>dest_len</code>  | Specifies the length of the <b>sockaddr</b> structure pointed to by the <code>dest_addr</code><br>39897 argument.                                                   |

39898 If the socket protocol supports broadcast and the specified address is a broadcast address for the  
 39899 socket protocol, `sendto()` shall fail if the `SO_BROADCAST` option is not set for the socket.

39900 The `dest_addr` argument specifies the address of the target. The `length` argument specifies the  
 39901 length of the message.

39902 Successful completion of a call to `sendto()` does not guarantee delivery of the message. A return  
 39903 value of `-1` indicates only locally-detected errors.

39904 If space is not available at the sending socket to hold the message to be transmitted and the  
 39905 socket file descriptor does not have `O_NONBLOCK` set, `sendto()` blocks until space is available.  
 39906 If space is not available at the sending socket to hold the message to be transmitted and the  
 39907 socket file descriptor does have `O_NONBLOCK` set, `sendto()` shall fail.

39908 The socket in use may require the process to have appropriate privileges to use the `sendto()`  
 39909 function.

39910 **RETURN VALUE**

39911 Upon successful completion, `sendto()` shall return the number of bytes sent. Otherwise, `-1` shall  
 39912 be returned and `errno` set to indicate the error.

39913 **ERRORS**

- 39914 The *sendto()* function shall fail if:
- 39915 [EAFNOSUPPORT]  
39916 Addresses in the specified address family cannot be used with this socket.
- 39917 [EAGAIN] or [EWOULDBLOCK]  
39918 The socket's file descriptor is marked O\_NONBLOCK and the requested  
39919 operation would block.
- 39920 [EBADF] The *socket* argument is not a valid file descriptor.
- 39921 [ECONNRESET] A connection was forcibly closed by a peer.
- 39922 [EINTR] A signal interrupted *sendto()* before any data was transmitted.
- 39923 [EMSGSIZE] The message is too large to be sent all at once, as the socket requires.
- 39924 [ENOTCONN] The socket is connection-mode but is not connected.
- 39925 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 39926 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or  
39927 more of the values set in *flags*.
- 39928 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is  
39929 no longer connected. In the latter case, and if the socket is of type  
39930 SOCK\_STREAM, the SIGPIPE signal is generated to the calling thread.
- 39931 If the address family of the socket is AF\_UNIX, then *sendto()* shall fail if:
- 39932 [EIO] An I/O error occurred while reading from or writing to the file system.
- 39933 [ELOOP] A loop exists in symbolic links encountered during resolution of the path  
39934 name in the socket address.
- 39935 [ENAMETOOLONG]  
39936 A component of a path name exceeded {NAME\_MAX} characters, or an entire  
39937 path name exceeded {PATH\_MAX} characters.
- 39938 [ENOENT] A component of the path name does not name an existing file or the path  
39939 name is an empty string.
- 39940 [ENOTDIR] A component of the path prefix of the path name in the socket address is not a  
39941 directory.
- 39942 The *sendto()* function may fail if:
- 39943 [EACCES] Search permission is denied for a component of the path prefix; or write  
39944 access to the named socket is denied.
- 39945 [EDESTADDRREQ]  
39946 The socket is not connection-mode and does not have its peer address set, and  
39947 no destination address was specified.
- 39948 [EHOSTUNREACH]  
39949 The destination host cannot be reached (probably because the host is down or  
39950 a remote router cannot reach it).
- 39951 [EINVAL] The *dest\_len* argument is not a valid length for the address family.
- 39952 [EIO] An I/O error occurred while reading from or writing to the file system.



- 39953 [EISCONN] A destination address was specified and the socket is already connected. This  
39954 error may or may not be returned for connection mode sockets.
- 39955 [ENETDOWN] The local network interface used to reach the destination is down. |
- 39956 [ENETUNREACH]  
39957 No route to the network is present.
- 39958 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 39959 [ENOMEM] Insufficient memory was available to fulfill the request. |
- 39960 If the address family of the socket is AF\_UNIX, then *sendto()* may fail if:
- 39961 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during |  
39962 resolution of the path name in the socket address. |
- 39963 [ENAMETOOLONG]  
39964 Path name resolution of a symbolic link produced an intermediate result  
39965 whose length exceeds {PATH\_MAX}.
- 39966 **EXAMPLES**  
39967 None.
- 39968 **APPLICATION USAGE**  
39969 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.
- 39970 **RATIONALE**  
39971 None.
- 39972 **FUTURE DIRECTIONS**  
39973 None.
- 39974 **SEE ALSO**  
39975 *getsockopt()*, *poll()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *send()*, *sendmsg()*, *setsockopt()*,  
39976 *shutdown()*, *socket()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/socket.h> |
- 39977 **CHANGE HISTORY**  
39978 First released in Issue 6. Derived from the XNS, Issue 5.2 specification. |
- 39979 The wording of the mandatory [ELOOP] error condition is updated, and a second optional |  
39980 [ELOOP] error condition is added. |

39981 **NAME**

39982 setbuf — assign buffering to a stream

39983 **SYNOPSIS**

39984 #include &lt;stdio.h&gt;

39985 void setbuf(FILE \*restrict stream, char \*restrict buf);

39986 **DESCRIPTION**

39987 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
39988 conflict between the requirements described here and the ISO C standard is unintentional. This  
39989 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

39990 Except that it returns no value, the function call:

39991 setbuf(stream, buf)

39992 shall be equivalent to:

39993 setvbuf(stream, buf, \_IOFBF, BUFSIZ)

39994 if *buf* is not a null pointer, or to:

39995 setvbuf(stream, buf, \_IONBF, BUFSIZ)

39996 if *buf* is a null pointer.39997 **RETURN VALUE**39998 The *setbuf()* function shall return no value.39999 **ERRORS**

40000 No errors are defined.

40001 **EXAMPLES**

40002 None.

40003 **APPLICATION USAGE**

40004 A common source of error is allocating buffer space as an “automatic” variable in a code block,  
40005 and then failing to close the stream in the same block.

40006 With *setbuf()*, allocating a buffer of {BUFSIZ} bytes does not necessarily imply that all of  
40007 {BUFSIZ} bytes are used for the buffer area.

40008 **RATIONALE**

40009 None.

40010 **FUTURE DIRECTIONS**

40011 None.

40012 **SEE ALSO**40013 *fopen()*, *setvbuf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdio.h>40014 **CHANGE HISTORY**

40015 First released in Issue 1. Derived from Issue 1 of the SVID.

40016 **Issue 6**40017 The prototype for *setbuf()* is updated for alignment with the ISO/IEC 9899:1999 standard.

40018 **NAME**

40019 setcontext — set current user context

40020 **SYNOPSIS**

40021 xSI #include &lt;ucontext.h&gt;

40022 int setcontext(const ucontext\_t \*ucp);

40023

40024 **DESCRIPTION**40025 Refer to *getcontext()*.

40026 **NAME**

40027           setegid — set effective group ID

40028 **SYNOPSIS**

40029           #include &lt;unistd.h&gt;

40030           int setegid(gid\_t *gid*);40031 **DESCRIPTION**40032           If *gid* is equal to the real group ID or the saved set-group-ID, or if the process has appropriate  
40033           privileges, *setegid()* shall set the effective group ID of the calling process to *gid*; the real group  
40034           ID, saved set-group-ID, and any supplementary group IDs shall remain unchanged.40035           The *setegid()* function shall not affect the supplementary group list in any way.40036 **RETURN VALUE**40037           Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
40038           indicate the error.40039 **ERRORS**40040           The *setegid()* function shall fail if:40041           [EINVAL]           The value of the *gid* argument is invalid and is not supported by the  
40042           implementation.40043           [EPERM]           The process does not have appropriate privileges and *gid* does not match the  
40044           real group ID or the saved set-group-ID.40045 **EXAMPLES**

40046           None.

40047 **APPLICATION USAGE**

40048           None.

40049 **RATIONALE**40050           Refer to the RATIONALE section in *setuid()*.40051 **FUTURE DIRECTIONS**

40052           None.

40053 **SEE ALSO**40054           *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the  
40055           Base Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>, <unistd.h>40056 **CHANGE HISTORY**

40057           First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

40058 **NAME**

40059            setenv — add or change environment variable

40060 **SYNOPSIS**

40061            #include &lt;stdlib.h&gt;

40062            int setenv(const char \*envname, const char \*envval, int overwrite);

40063 **DESCRIPTION**

40064            The *setenv()* function updates or adds a variable in the environment of the calling process. The *envname* argument points to a string containing the name of an environment variable to be added or altered. The environment variable shall be set to the value to which *envval* points. The function shall fail if *envname* points to a string which contains an '=' character. If the environment variable named by *envname* already exists and the value of *overwrite* is non-zero, the function shall return success and the environment shall be updated. If the environment variable named by *envname* already exists and the value of *overwrite* is zero, the function shall return success and the environment shall remain unchanged.

40072            If the application modifies *environ* or the pointers to which it points, the behavior of *setenv()* is undefined. The *setenv()* function shall update the list of pointers to which *environ* points.

40074            The strings described by *envname* and *envval* are copied by this function.

40075            The *setenv()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

40077 **RETURN VALUE**

40078            Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to indicate the error, and the environment shall be unchanged.

40080 **ERRORS**

40081            The *setenv()* function shall fail if:

40082            [EINVAL]            The *name* argument is a null pointer, points to an empty string, or points to a string containing an '=' character.

40084            [ENOMEM]           Insufficient memory was available to add a variable or its value to the environment.

40086 **EXAMPLES**

40087            None.

40088 **APPLICATION USAGE**

40089            None.

40090 **RATIONALE**

40091            Unanticipated results may occur if *setenv()* changes the external variable *environ*. In particular, if the optional *envp* argument to *main()* is present, it is not changed, and thus may point to an obsolete copy of the environment (as may any other copy of *environ*). However, other than the aforementioned restriction, the developers of IEEE Std. 1003.1-200x intended that the traditional method of walking through the environment by way of the *environ* pointer must be supported.

40096            It was decided that *setenv()* should be required by this revision because it addresses a piece of missing functionality, and does not impose a significant burden on the implementor.

40098            There was considerable debate as to whether the System V *putenv()* function or the BSD *setenv()* function should be required as a mandatory function. The *setenv()* function was chosen because it permitted the implementation of *unsetenv()* function to delete environmental variables, without specifying an additional interface. The *putenv()* function is available as an XSI extension.

40103           The standard developers considered requiring that *setenv()* indicate an error when a call to it  
40104           would result in exceeding {ARG\_MAX}. The requirement was rejected since the condition might  
40105           be temporary, with the application eventually reducing the environment size. The ultimate  
40106           success or failure depends on the size at the time of a call to *exec*, which returns an indication of  
40107           this error condition.

40108 **FUTURE DIRECTIONS**

40109           None.

40110 **SEE ALSO**

40111           *getenv()*, *unsetenv()*, the Base Definitions volume of IEEE Std. 1003.1-200x, `<stdlib.h>`, |  
40112           `<sys/types.h>`, `<unistd.h>`

40113 **CHANGE HISTORY**

40114           First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

40115 **NAME**

40116            seteuid — set effective user ID

40117 **SYNOPSIS**

40118            #include &lt;unistd.h&gt;

40119            int seteuid(uid\_t uid);

40120 **DESCRIPTION**

40121            If *uid* is equal to the real user ID or the saved set-user-ID, or if the process has appropriate  
40122            privileges, *seteuid()* shall set the effective user ID of the calling process to *uid*; the real user ID  
40123            and saved set-user-ID shall remain unchanged.

40124            The *seteuid()* function shall not affect the supplementary group list in any way.40125 **RETURN VALUE**

40126            Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
40127            indicate the error.

40128 **ERRORS**40129            The *seteuid()* function shall fail if:

40130            [EINVAL]            The value of the *uid* argument is invalid and is not supported by the  
40131            implementation.

40132            [EPERM]            The process does not have appropriate privileges and *uid* does not match the  
40133            real group ID or the saved set-group-ID.

40134 **EXAMPLES**

40135            None.

40136 **APPLICATION USAGE**

40137            None.

40138 **RATIONALE**40139            Refer to the RATIONALE section in *setuid()*.40140 **FUTURE DIRECTIONS**

40141            None.

40142 **SEE ALSO**

40143            *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the  
40144            Base Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>, <unistd.h>

40145 **CHANGE HISTORY**

40146            First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

40147 **NAME**

40148        setgid — set-group-ID

40149 **SYNOPSIS**

40150        #include &lt;unistd.h&gt;

40151        int setgid(gid\_t *gid*);40152 **DESCRIPTION**40153        If the process has appropriate privileges, *setgid()* shall set the real group ID, effective group ID,  
40154        and the saved set-group-ID of the calling process to *gid*.40155        If the process does not have appropriate privileges, but *gid* is equal to the real group ID or the  
40156        saved set-group-ID, *setgid()* shall set the effective group ID to *gid*; the real group ID and saved  
40157        set-group-ID shall remain unchanged.40158        The *setgid()* function shall not affect the supplementary group list in any way.

40159        Any supplementary group IDs of the calling process shall remain unchanged.

40160 **RETURN VALUE**40161        Upon successful completion, 0 is returned. Otherwise, -1 shall be returned and *errno* set to  
40162        indicate the error.40163 **ERRORS**40164        The *setgid()* function shall fail if:40165        [EINVAL]        The value of the *gid* argument is invalid and is not supported by the  
40166        implementation.40167        [EPERM]        The process does not have appropriate privileges and *gid* does not match the  
40168        real group ID or the saved set-group-ID.40169 **EXAMPLES**

40170        None.

40171 **APPLICATION USAGE**

40172        None.

40173 **RATIONALE**40174        Refer to the RATIONALE section in *setuid()*.40175 **FUTURE DIRECTIONS**

40176        None.

40177 **SEE ALSO**40178        *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setregid()*, *setreuid()*, *setuid()*, the  
40179        Base Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>, <unistd.h>40180 **CHANGE HISTORY**

40181        First released in Issue 1. Derived from Issue 1 of the SVID.

40182 **Issue 4**40183        The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
40184        XSI-conformant systems.

40185        The following change is incorporated for alignment with the FIPS requirements:

- 40186
- All references to the saved set-user-ID are marked as extensions. This is because Issue 4  
40187        defines this mechanism as mandatory, whereas the ISO POSIX-1 standard defines that it is  
40188        only supported if `_POSIX_SAVED_IDS` is set.



40189 **Issue 6**

40190 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

40191 The following new requirements on POSIX implementations derive from alignment with the  
40192 Single UNIX Specification:

40193 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
40194 required for conforming implementations of previous POSIX specifications, it was not  
40195 required for UNIX applications.

40196 • Functionality associated with `_POSIX_SAVED_IDS` is now mandated. This is a FIPS  
40197 requirement.

40198 The following changes were made to align with the IEEE P1003.1a draft standard:

40199 • The effects of `setgid()` in processes without appropriate privileges are changed

40200 • A requirement that the supplementary group list is not affected is added.

40201 **NAME**

40202           setgrent — reset group database to first entry

40203 **SYNOPSIS**

40204 xSI       #include <grp.h>

40205           void setgrent(void);

40206

40207 **DESCRIPTION**

40208           Refer to *endgrent()*.

40209 **NAME**

40210           sethostent — network host database functions

40211 **SYNOPSIS**

40212           #include <netdb.h>

40213           void sethostent(int *stayopen*);

40214 **DESCRIPTION**

40215           Refer to *endhostent()*.

40216 **NAME**

40217       setitimer — set value of interval timer

40218 **SYNOPSIS**

40219 XSI       #include <sys/time.h>

40220       int setitimer(int *which*, const struct itimerval \*restrict *value*,  
40221                    struct itimerval \*restrict *ovalue*);

40222

40223 **DESCRIPTION**

40224       Refer to *getitimer()*.

40225 **NAME**

40226 setjmp — set jump point for a non-local goto

40227 **SYNOPSIS**

40228 #include &lt;setjmp.h&gt;

40229 int setjmp(jmp\_buf env);

40230 **DESCRIPTION**

40231 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
40232 conflict between the requirements described here and the ISO C standard is unintentional. This  
40233 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

40234 A call to *setjmp()*, shall save the calling environment in its *env* argument for later use by  
40235 *longjmp()*.

40236 It is unspecified whether *setjmp()* is a macro or a function. If a macro definition is suppressed in  
40237 order to access an actual function, or a program defines an external identifier with the name  
40238 *setjmp*, the behavior is undefined.

40239 All accessible objects have values as of the time *longjmp()* was called, except that the values of  
40240 objects of automatic storage duration which are local to the function containing the invocation of  
40241 the corresponding *setjmp()* which do not have volatile-qualified type and which are changed  
40242 between the *setjmp()* invocation and *longjmp()* call are indeterminate.

40243 An application shall ensure that an invocation of *setjmp()* appears in one of the following  
40244 contexts only:

- 40245 • The entire controlling expression of a selection or iteration statement
- 40246 • One operand of a relational or equality operator with the other operand an integral constant  
40247 expression, with the resulting expression being the entire controlling expression of a  
40248 selection or iteration statement
- 40249 • The operand of a unary '!' operator with the resulting expression being the entire  
40250 controlling expression of a selection or iteration
- 40251 • The entire expression of an expression statement (possibly cast to **void**)

40252 If the invocation appears in any other context, the behavior is undefined.

40253 **RETURN VALUE**

40254 If the return is from a direct invocation, *setjmp()* shall return 0. If the return is from a call to  
40255 *longjmp()*, *setjmp()* shall return a non-zero value.

40256 **ERRORS**

40257 No errors are defined.

40258 **EXAMPLES**

40259 None.

40260 **APPLICATION USAGE**

40261 In general, *sigsetjmp()* is more useful in dealing with errors and interrupts encountered in a low-  
40262 level subroutine of a program.

40263 **RATIONALE**

40264 None.

40265 **FUTURE DIRECTIONS**

40266 None.

40267 **SEE ALSO**40268 *longjmp()*, *sigsetjmp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <setjmp.h>40269 **CHANGE HISTORY**

40270 First released in Issue 1. Derived from Issue 1 of the SVID.

40271 **Issue 4**40272 This issue states that *setjmp()* is a macro or a function; previous issues stated that it was a macro.40273 Warnings have also been added about the suppression of a *setjmp()* macro definition.40274 Text describing the accessibility of objects after a *longjmp()* call is added to the DESCRIPTION.40275 This text is imported from the entry for *longjmp()*.40276 Text describing the contexts in which calls to *setjmp()* are valid is moved to the DESCRIPTION

40277 from the APPLICATION USAGE section.

40278 The APPLICATION USAGE section is changed to refer to *sigsetjmp()*.40279 **Issue 6**

40280 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

40281 **NAME**40282 setkey — set encoding key (**CRYPT**)40283 **SYNOPSIS**

40284 XSI #include &lt;stdlib.h&gt;

40285 void setkey(const char \*key);

40286

40287 **DESCRIPTION**

40288 The *setkey()* function provides (rather primitive) access to an implementation-defined encoding  
40289 algorithm. The argument of *setkey()* is an array of length 64 bytes containing only the bytes with  
40290 numerical value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each  
40291 group is ignored; this gives a 56-bit key which is used by the algorithm. This is the key that shall  
40292 be used with the algorithm to encode a string *block* passed to *encrypt()*.

40293 The *setkey()* function shall not change the setting of *errno* if successful. An application wishing to  
40294 check for error situations should set *errno* to 0 before calling *setkey()*. If *errno* is non-zero on  
40295 return, an error has occurred.

40296 The *setkey()* function need not be reentrant. A function that is not required to be reentrant is not  
40297 required to be thread-safe.

40298 **RETURN VALUE**

40299 No values are returned.

40300 **ERRORS**40301 The *setkey()* function shall fail if:

40302 [ENOSYS] The functionality is not supported on this implementation.

40303 **EXAMPLES**

40304 None.

40305 **APPLICATION USAGE**

40306 Decoding need not be implemented in all environments. This is related to U.S. Government  
40307 restrictions on encryption and decryption routines: the DES decryption algorithm cannot be  
40308 exported outside the U.S. Historical practice has been to ship a different version of the  
40309 encryption library without the decryption feature in the routines supplied. Thus the exported  
40310 version of *encrypt()* does encoding but not decoding.

40311 **RATIONALE**

40312 None.

40313 **FUTURE DIRECTIONS**

40314 None.

40315 **SEE ALSO**40316 *crypt()*, *encrypt()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>40317 **CHANGE HISTORY**

40318 First released in Issue 1. Derived from Issue 1 of the SVID.

40319 **Issue 4**40320 The type of argument *key* is changed from **char\*** to **const char\***.

40321 The description of the array is put in terms of bytes instead of characters.

40322 The APPLICATION USAGE section is added.

40323 **Issue 5**

40324

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.



40325 **NAME**

40326 setlocale — set program locale

40327 **SYNOPSIS**

40328 #include &lt;locale.h&gt;

40329 char \*setlocale(int *category*, const char \**locale*);40330 **DESCRIPTION**

40331 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 40332 conflict between the requirements described here and the ISO C standard is unintentional. This  
 40333 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

40334 The *setlocale()* function selects the appropriate piece of the program's locale, as specified by the  
 40335 *category* and *locale* arguments, and may be used to change or query the program's entire locale or  
 40336 portions thereof. The value *LC\_ALL* for *category* names the program's entire locale; other values  
 40337 for *category* name only a part of the program's locale:

40338 *LC\_COLLATE* Affects the behavior of regular expressions and the collation functions.

40339 *LC\_CTYPE* Affects the behavior of regular expressions, character classification, character  
 40340 conversion functions, and wide-character functions.

40341 CX *LC\_MESSAGES* Affects what strings are expected by commands and utilities as affirmative or  
 40342 negative responses.

40343 XSI It also affects what strings are given by commands and utilities as affirmative  
 40344 or negative responses, and the content of messages.

40345 *LC\_MONETARY* Affects the behavior of functions that handle monetary values.

40346 *LC\_NUMERIC* Affects the behavior of functions that handle numeric values.

40347 *LC\_TIME* Affects the behavior of the time conversion functions.

40348 The *locale* argument is a pointer to a character string containing the required setting of *category*.  
 40349 The contents of this string are implementation-defined. In addition, the following preset values  
 40350 of *locale* are defined for all settings of *category*:

40351 CX "POSIX" Specifies the minimal environment for C-language translation called POSIX  
 40352 locale. If *setlocale()* is not invoked, the POSIX locale is the default at entry to  
 40353 *main()*.

40354 "C" Same as "POSIX".

40355 "" Specifies an implementation-defined native environment. For XSI-conformant  
 40356 systems, this corresponds to the value of the associated environment  
 40357 variables, *LC\_\** and *LANG*; see the Base Definitions volume of  
 40358 IEEE Std. 1003.1-200x, Chapter 7, Locale and the Base Definitions volume of  
 40359 IEEE Std. 1003.1-200x, Chapter 8, Environment Variables.

40360 A null pointer Used to direct *setlocale()* to query the current internationalized environment  
 40361 and return the name of the *locale*).

40362 THR The locale state is common to all threads within a process.

40363 **RETURN VALUE**

40364 Upon successful completion, *setlocale()* shall return the string associated with the specified  
 40365 category for the new locale. Otherwise, *setlocale()* shall return a null pointer and the program's  
 40366 locale is not changed.

40367 A null pointer for *locale* causes *setlocale()* to return a pointer to the string associated with the  
40368 *category* for the program's current locale. The program's locale shall not be changed.

40369 The string returned by *setlocale()* is such that a subsequent call with that string and its associated  
40370 *category* shall restore that part of the program's locale. The application shall not modify the string  
40371 returned which may be overwritten by a subsequent call to *setlocale()*.

#### 40372 ERRORS

40373 No errors are defined.

#### 40374 EXAMPLES

40375 None.

#### 40376 APPLICATION USAGE

40377 The following code illustrates how a program can initialize the international environment for  
40378 one language, while selectively modifying the program's locale such that regular expressions  
40379 and string operations can be applied to text recorded in a different language:

```
40380 setlocale(LC_ALL, "De");
40381 setlocale(LC_COLLATE, "Fr@dict");
```

40382 Internationalized programs must call *setlocale()* to initiate a specific language operation. This can  
40383 be done by calling *setlocale()* as follows:

```
40384 setlocale(LC_ALL, "");
```

40385 Changing the setting of *LC\_MESSAGES* has no effect on catalogs that have already been opened  
40386 by calls to *catopen()*.

#### 40387 RATIONALE

40388 The ISO C standard defines a collection of functions to support internationalization. One of the  
40389 most significant aspects of these functions is a facility to set and query the *international*  
40390 *environment*. The international environment is a repository of information that affects the  
40391 behavior of certain functionality, namely:

- 40392 1. Character handling
- 40393 2. String handling (that is, collating)
- 40394 3. Date/time formatting
- 40395 4. Numeric editing

40396 The *setlocale()* function provides the application developer with the ability to set all or portions,  
40397 called *categories*, of the international environment. These categories correspond to the areas of  
40398 functionality, mentioned above. The syntax for *setlocale()* is as follows:

```
40399 char *setlocale(int category, const char *locale);
```

40400 where *category* is the name of one of five categories, namely:

```
40401 LC_COLLATE
40402 LC_CTYPE
40403 LC_MESSAGES
40404 LC_MONETARY
40405 LC_NUMERIC
40406 LC_TIME
```

40407 In addition, a special value called *LC\_ALL* directs *setlocale()* to set all categories.

40408 There are two primary uses of *setlocale()*:

- 40409           1. Querying the international environment to find out what it is set to  
 40410           2. Setting the international environment, or *locale*, to a specific value

40411           The behavior of *setlocale()* in these two areas is described below. Since it is difficult to describe  
 40412           the behavior in words, examples are used to illustrate the behavior of specific uses.

40413           To query the international environment, *setlocale()* is invoked with a specific category and the  
 40414           NULL pointer as the locale. The NULL pointer is a special directive to *setlocale()* that tells it to  
 40415           query rather than set the international environment. The following syntax is used to query the  
 40416           name of the international environment:

```
40417 setlocale({LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, \
40418 LC_NUMERIC, LC_TIME}, (char *) NULL);
```

40419           The *setlocale()* function shall return the string corresponding to the current international  
 40420           environment. This value may be used by a subsequent call to *setlocale()* to reset the international  
 40421           environment to this value. However, it should be noted that the return value from *setlocale()* is a  
 40422           pointer to a static area within the function and is not guaranteed to remain unchanged (that is, it  
 40423           may be modified by a subsequent call to *setlocale()*). Therefore, if the purpose of calling  
 40424           *setlocale()* is to save the value of the current international environment so it can be changed and  
 40425           reset later, the return value should be copied to an array of **char** in the calling program.

40426           There are three ways to set the international environment with *setlocale()*:

40427           *setlocale(category, string)*

40428           This usage sets a specific *category* in the international environment to a specific value  
 40429           corresponding to the value of the *string*. A specific example is provided below:

```
40430 setlocale(LC_ALL, "Fr_FR.8859");
```

40431           In this example, all categories of the international environment are set to the locale  
 40432           corresponding to the string "Fr\_FR.8859", or to the French language as spoken in France  
 40433           using the ISO/IEC 8859-1:1998 standard codeset.

40434           If the string does not correspond to a valid locale, *setlocale()* shall return a NULL pointer  
 40435           and the international environment is not changed. Otherwise, *setlocale()* shall return the  
 40436           name of the locale just set.

40437           *setlocale(category, "C")*

40438           The ISO C standard states that one locale must exist on all conforming implementations.  
 40439           The name of the locale is C and corresponds to a minimal international environment needed  
 40440           to support the C programming language.

40441           *setlocale(category, "")*

40442           This sets a specific category to an implementation-defined default. For POSIX-conforming  
 40443           systems, this corresponds to the value of the environment variables.

#### 40444 FUTURE DIRECTIONS

40445           None.

#### 40446 SEE ALSO

40447           *exec*, *isalnum()*, *isalpha()*, *iscntrl()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
 40448           *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
 40449           *iswspace()*, *iswupper()*, *localeconv()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *nl\_langinfo()*, *printf()*, *scanf()*,  
 40450           *setlocale()*, *strcoll()*, *strerror()*, *strfmon()*, *strtod()*, *strxfrm()*, *tolower()*, *toupper()*, *towlower()*,  
 40451           *towupper()*, *wscoll()*, *wctod()*, *wcstombs()*, *wcsxfrm()*, *wctomb()*, the Base Definitions volume of  
 40452           IEEE Std. 1003.1-200x, <langinfo.h>, <locale.h>

40453 **CHANGE HISTORY**

40454 First released in Issue 3.

40455 **Issue 4**40456 The description of *LC\_MESSAGES* is extended to indicate that this category also determines  
40457 what strings are produced by commands and utilities for affirmative and negative responses,  
40458 and that it affects the content of other program messages. This is marked as an extension.40459 References to *nl\_langinfo()* are removed.40460 The description of the implementation-defined native locale (" ") is clarified by stating the related  
40461 environment variables explicitly.

40462 The APPLICATION USAGE section is expanded.

40463 The following changes are incorporated for alignment with the ISO C standard and the  
40464 ISO POSIX-1 standard:40465 • The type of the argument *locale* is changed from **char\*** to **const char\***.

40466 • The name "POSIX" is added to the list of standard locale names.

40467 The following change is incorporated for alignment with the ISO POSIX-2 standard:

40468 • The *LC\_MESSAGES* value for *category* is added to the DESCRIPTION.40469 **Issue 5**

40470 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

40471 **Issue 6**

40472 Extensions beyond the ISO C standard are now marked.

40473 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

40474 **NAME**

40475 setlogmask — set log priority mask

40476 **SYNOPSIS**

40477 xSI #include &lt;syslog.h&gt;

40478 int setlogmask(int *maskpri*);

40479

40480 **DESCRIPTION**40481 Refer to *closelog()*.

## setnetent()

40482 **NAME**

40483           setnetent — network database function

40484 **SYNOPSIS**

40485           #include <netdb.h>

40486           void setnetent(int stayopen);

40487 **DESCRIPTION**

40488           Refer to *endnetent()*.

40489 **NAME**

40490 setpgid — set process group ID for job control

40491 **SYNOPSIS**

40492 #include &lt;unistd.h&gt;

40493 int setpgid(pid\_t pid, pid\_t pgid);

40494 **DESCRIPTION**

40495 The *setpgid()* function is used either to join an existing process group or create a new process  
 40496 group within the session of the calling process. The process group ID of a session leader shall  
 40497 not change. Upon successful completion, the process group ID of the process with a process ID  
 40498 that matches *pid* shall be set to *pgid*. As a special case, if *pid* is 0, the process ID of the calling  
 40499 process shall be used. Also, if *pgid* is 0, the process group ID of the indicated process shall be  
 40500 used.

40501 **RETURN VALUE**

40502 Upon successful completion, *setpgid()* shall return 0; otherwise, -1 shall be returned and *errno*  
 40503 shall be set to indicate the error.

40504 **ERRORS**40505 The *setpgid()* function shall fail if:

40506 [EACCES] The value of the *pid* argument matches the process ID of a child process of the  
 40507 calling process and the child process has successfully executed one of the *exec*  
 40508 functions.

40509 [EINVAL] The value of the *pgid* argument is less than 0, or is not a value supported by  
 40510 the implementation.

40511 [EPERM] The process indicated by the *pid* argument is a session leader.

40512 [EPERM] The value of the *pid* argument matches the process ID of a child process of the  
 40513 calling process and the child process is not in the same session as the calling  
 40514 process.

40515 [EPERM] The value of the *pgid* argument is valid but does not match the process ID of  
 40516 the process indicated by the *pid* argument and there is no process with a  
 40517 process group ID that matches the value of the *pgid* argument in the same  
 40518 session as the calling process.

40519 [ESRCH] The value of the *pid* argument does not match the process ID of the calling  
 40520 process or of a child process of the calling process.

40521 **EXAMPLES**

40522 None.

40523 **APPLICATION USAGE**

40524 None.

40525 **RATIONALE**

40526 The *setpgid()* function is used to group processes together for the purpose of signaling,  
 40527 placement in foreground or background, and other job control actions.

40528 The *setpgid()* function is similar to the *setpgrp()* function of 4.2 BSD, except that 4.2 BSD allowed  
 40529 the specified new process group to assume any value. This presents certain security problems  
 40530 and is more flexible than necessary to support job control.

40531 To provide tighter security, *setpgid()* only allows the calling process to join a process group  
 40532 already in use inside its session or create a new process group whose process group ID was

40533 equal to its process ID.

40534 When a job control shell spawns a new job, the processes in the job must be placed into a new  
40535 process group via *setpgid()*. There are two timing constraints involved in this action:

40536 1. The new process must be placed in the new process group before the appropriate program  
40537 is launched via one of the *exec* functions.

40538 2. The new process must be placed in the new process group before the shell can correctly  
40539 send signals to the new process group.

40540 To address these constraints, the following actions are performed. The new processes call  
40541 *setpgid()* to alter their own process groups after *fork()* but before *exec*. This satisfies the first  
40542 constraint. Under 4.3 BSD, the second constraint is satisfied by the synchronization property of  
40543 *vfork()*; that is, the shell is suspended until the child has completed the *exec*, thus ensuring that  
40544 the child has completed the *setpgid()*. A new version of *fork()* with this same synchronization  
40545 property was considered, but it was decided instead to merely allow the parent shell process to  
40546 adjust the process group of its child processes via *setpgid()*. Both timing constraints are now  
40547 satisfied by having both the parent shell and the child attempt to adjust the process group of the  
40548 child process; it does not matter which succeeds first.

40549 Because it would be confusing to an application to have its process group change after it began  
40550 executing (that is, after *exec*), and because the child process would already have adjusted its  
40551 process group before this, the [EACCES] error was added to disallow this.

40552 One non-obvious use of *setpgid()* is to allow a job control shell to return itself to its original  
40553 process group (the one in effect when the job control shell was executed). A job control shell  
40554 does this before returning control back to its parent when it is terminating or suspending itself as  
40555 a way of restoring its job control “state” back to what its parent would expect. (Note that the  
40556 original process group of the job control shell typically matches the process group of its parent,  
40557 but this is not necessarily always the case.)

#### 40558 FUTURE DIRECTIONS

40559 None.

#### 40560 SEE ALSO

40561 *exec*, *getpgrp()*, *setsid()*, *tcsetpgrp()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
40562 <sys/types.h>, <unistd.h>

#### 40563 CHANGE HISTORY

40564 First released in Issue 3.

40565 Entry included for alignment with the POSIX.1-1988 standard.

#### 40566 Issue 4

40567 The function is no longer marked as OPTIONAL FUNCTIONALITY.

40568 The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
40569 XSI-conformant systems.

40570 The <unistd.h> header is added to the SYNOPSIS section.

40571 The DESCRIPTION in Issue 3 defined the behavior of this function for implementations that  
40572 either supported or did not support job control. As job control is defined as mandatory in Issue  
40573 4, only the former of these is now described.

40574 The [ENOSYS] error is removed from the ERRORS section.



40575 **Issue 6**

40576 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

40577 The following new requirements on POSIX implementations derive from alignment with the  
40578 Single UNIX Specification:

- 40579 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
40580 required for conforming implementations of previous POSIX specifications, it was not  
40581 required for UNIX applications.
- 40582 • The `setpgid()` function is mandatory since `_POSIX_JOB_CONTROL` is required to be defined  
40583 in this issue. This is a FIPS requirement.

40584 **NAME**

40585 setpgrp — set process group ID

40586 **SYNOPSIS**

40587 XSI #include &lt;unistd.h&gt;

40588 pid\_t setpgrp(void);

40589

40590 **DESCRIPTION**

40591 If the calling process is not already a session leader, *setpgrp()* sets the process group ID of the  
40592 calling process to the process ID of the calling process. If *setpgrp()* creates a new session, then  
40593 the new session has no controlling terminal.

40594 The *setpgrp()* function has no effect when the calling process is a session leader.

40595 **RETURN VALUE**40596 Upon completion, *setpgrp()* shall return the process group ID.40597 **ERRORS**

40598 No errors are defined.

40599 **EXAMPLES**

40600 None.

40601 **APPLICATION USAGE**

40602 None.

40603 **RATIONALE**

40604 None.

40605 **FUTURE DIRECTIONS**

40606 None.

40607 **SEE ALSO**

40608 *exec*, *fork()*, *getpid()*, *getsid()*, *kill()*, *setpgid()*, *setsid()*, the Base Definitions volume of  
40609 IEEE Std. 1003.1-200x, <unistd.h>

40610 **CHANGE HISTORY**

40611 First released in Issue 4, Version 2.

40612 **Issue 5**

40613 Moved from X/OPEN UNIX extension to BASE.

40614 **NAME**

40615           setpriority — set the nice value

40616 **SYNOPSIS**

40617 xSI       #include &lt;sys/resource.h&gt;

40618           int setpriority(int *which*, id\_t *who*, int *nice*);

40619

40620 **DESCRIPTION**40621           Refer to *getpriority()*.

# setprotoent()

40622 **NAME**

40623           setprotoent — network protocol database functions

40624 **SYNOPSIS**

40625           #include <netdb.h>

40626           void setprotoent(int *stayopen*);

40627 **DESCRIPTION**

40628           Refer to *endprotoent()*.

40629 **NAME**

40630           setpwent — user database function

40631 **SYNOPSIS**

40632 xSI       #include &lt;pwd.h&gt;

40633           void setpwent(void);

40634

40635 **DESCRIPTION**40636           Refer to *endpwent()*.

40637 **NAME**

40638 setregid — set real and effective group IDs

40639 **SYNOPSIS**

40640 XSI #include &lt;unistd.h&gt;

40641 int setregid(gid\_t rgid, gid\_t egid);

40642

40643 **DESCRIPTION**40644 The *setregid()* function is used to set the real and effective group IDs of the calling process.40645 If *rgid* is  $-1$ , the real group ID shall not be changed; if *egid* is  $-1$ , the effective group ID shall not  
40646 be changed.

40647 The real and effective group IDs may be set to different values in the same call.

40648 Only a process with appropriate privileges can set the real group ID and the effective group ID  
40649 to any valid value.40650 A non-privileged process can set either the real group ID to the saved set-group-ID from one of  
40651 the *exec* family of functions, or the effective group ID to the saved set-group-ID or the real group  
40652 ID.

40653 Any supplementary group IDs of the calling process remain unchanged.

40654 **RETURN VALUE**40655 Upon successful completion, 0 shall be returned. Otherwise,  $-1$  shall be returned and *errno* set to  
40656 indicate the error, and neither of the group IDs are changed.40657 **ERRORS**40658 The *setregid()* function shall fail if:40659 [EINVAL] The value of the *rgid* or *egid* argument is invalid or out-of-range.40660 [EPERM] The process does not have appropriate privileges and a change other than  
40661 changing the real group ID to the saved set-group-ID, or changing the  
40662 effective group ID to the real group ID or the saved set-group-ID, was  
40663 requested.40664 **EXAMPLES**

40665 None.

40666 **APPLICATION USAGE**40667 If a set-group-ID process sets its effective group ID to its real group ID, it can still set its effective  
40668 group ID back to the saved set-group-ID.40669 **RATIONALE**

40670 None.

40671 **FUTURE DIRECTIONS**

40672 None.

40673 **SEE ALSO**40674 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setreuid()*, *setuid()*, the  
40675 Base Definitions volume of IEEE Std. 1003.1-200x, <unistd.h>40676 **CHANGE HISTORY**

40677 First released in Issue 4, Version 2.

40678 **Issue 5**

40679 Moved from X/OPEN UNIX extension to BASE.

40680 The DESCRIPTION is updated to indicate that the saved set-group-ID can be set by any of the  
40681 *exec* family of functions, not just *execev()*.

40682 **NAME**

40683 setreuid — set real and effective user IDs

40684 **SYNOPSIS**40685 XSI `#include <unistd.h>`40686 `int setreuid(uid_t ruid, uid_t euid);`

40687

40688 **DESCRIPTION**

40689 The *setreuid()* function sets the real and effective user IDs of the current process to the values  
 40690 specified by the *ruid* and *euid* arguments. If *ruid* or *euid* is  $-1$ , the corresponding effective or real  
 40691 user ID of the current process is left unchanged.

40692 A process with appropriate privileges can set either ID to any value. An unprivileged process  
 40693 can only set the effective user ID if the *euid* argument is equal to either the real, effective, or  
 40694 saved user ID of the process.

40695 It is unspecified whether a process without appropriate privileges is permitted to change the real  
 40696 user ID to match the current real, effective, or saved set-user-ID of the process.

40697 **RETURN VALUE**

40698 Upon successful completion, 0 shall be returned. Otherwise,  $-1$  shall be returned and *errno* set to  
 40699 indicate the error.

40700 **ERRORS**40701 The *setreuid()* function shall fail if:40702 [EINVAL] The value of the *ruid* or *euid* argument is invalid or out-of-range. |

40703 [EPERM] The current process does not have appropriate privileges, and either an |  
 40704 attempt was made to change the effective user ID to a value other than the  
 40705 real user ID or the saved set-user-ID or an attempt was made to change the  
 40706 real user ID to a value not permitted by the implementation.

40707 **EXAMPLES**40708 **Setting the Effective User ID to the Real User ID**

40709 The following example sets the effective user ID of the calling process to the real user ID, so that  
 40710 files created later will be owned by the current user.

```
40711 #include <unistd.h>
40712 #include <sys/types.h>
40713 ...
40714 setreuid(getuid(), getuid());
40715 ...
```

40716 **APPLICATION USAGE**

40717 None.

40718 **RATIONALE**

40719 None.

40720 **FUTURE DIRECTIONS**

40721 None.



40722 **SEE ALSO**

40723 *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setuid()*, the Base  
40724 Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>

40725 **CHANGE HISTORY**

40726 First released in Issue 4, Version 2.

40727 **Issue 5**

40728 Moved from X/OPEN UNIX extension to BASE.

40729 **NAME**

40730 setrlimit — control maximum resource consumption

40731 **SYNOPSIS**

40732 XSI #include <sys/resource.h>

40733 int setrlimit(int resource, const struct rlimit \*rlp);

40734

40735 **DESCRIPTION**

40736 Refer to *getrlimit()*.

40737 **NAME**

40738           setservent — network services database functions

40739 **SYNOPSIS**

40740           #include <netdb.h>

40741           void setservent(int *stayopen*);

40742 **DESCRIPTION**

40743           Refer to *endservent()*.

40744 **NAME**

40745 setsid — create session and set process group ID

40746 **SYNOPSIS**

40747 #include <unistd.h>

40748 pid\_t setsid(void);

40749 **DESCRIPTION**

40750 The *setsid()* function creates a new session, if the calling process is not a process group leader.  
40751 Upon return the calling process shall be the session leader of this new session, shall be the  
40752 process group leader of a new process group, and shall have no controlling terminal. The  
40753 process group ID of the calling process shall be set equal to the process ID of the calling process.  
40754 The calling process shall be the only process in the new process group and the only process in  
40755 the new session.

40756 **RETURN VALUE**

40757 Upon successful completion, *setsid()* shall return the value of the new process group ID of the  
40758 calling process. Otherwise, it shall return (**pid\_t**)-1 and set *errno* to indicate the error.

40759 **ERRORS**

40760 The *setsid()* function shall fail if:

40761 [EPERM] The calling process is already a process group leader, or the process group ID  
40762 of a process other than the calling process matches the process ID of the  
40763 calling process.

40764 **EXAMPLES**

40765 None.

40766 **APPLICATION USAGE**

40767 None.

40768 **RATIONALE**

40769 The *setsid()* function is similar to the *setpgrp()* function of System V. System V, without job  
40770 control, groups processes into process groups and creates new process groups via *setpgrp()*; only  
40771 one process group may be part of a login session.

40772 Job control allows multiple process groups within a login session. In order to limit job control  
40773 actions so that they can only affect processes in the same login session, this volume of  
40774 IEEE Std. 1003.1-200x adds the concept of a session that is created via *setsid()*. The *setsid()*  
40775 function also creates the initial process group contained in the session. Additional process  
40776 groups can be created via the *setpgid()* function. A System V process group would correspond to  
40777 a POSIX System Interfaces session containing a single POSIX process group. Note that this  
40778 function requires that the calling process not be a process group leader. The usual way to ensure  
40779 this is true is to create a new process with *fork()* and have it call *setsid()*. The *fork()* function  
40780 guarantees that the process ID of the new process does not match any existing process group ID.

40781 **FUTURE DIRECTIONS**

40782 None.

40783 **SEE ALSO**

40784 *getsid()*, *setpgid()*, *setpgrp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>,  
40785 <unistd.h>

40786 **CHANGE HISTORY**

40787 First released in Issue 3.

40788 Entry included for alignment with the POSIX.1-1988 standard.

40789 **Issue 4**40790 The `<sys/types.h>` header is now marked as optional (OH); this header need not be included on XSI-conformant systems.40792 The `<unistd.h>` header is added to the SYNOPSIS section.40793 The argument list is explicitly defined as **void**.40794 **Issue 6**40795 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

40796 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 40797
- 40798 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
40799 required for conforming implementations of previous POSIX specifications, it was not  
40800 required for UNIX applications.

## 40801 NAME

40802 setsockopt — set the socket options

## 40803 SYNOPSIS

40804 #include &lt;sys/socket.h&gt;

40805 int setsockopt(int *socket*, int *level*, int *option\_name*,  
40806 const void \**option\_value*, socklen\_t *option\_len*);

## 40807 DESCRIPTION

40808 The *setsockopt()* function sets the option specified by the *option\_name* argument, at the protocol  
40809 level specified by the *level* argument, to the value pointed to by the *option\_value* argument for the  
40810 socket associated with the file descriptor specified by the *socket* argument.40811 The *level* argument specifies the protocol level at which the option resides. To set options at the  
40812 socket level, specify the *level* argument as SOL\_SOCKET. To set options at other levels, supply  
40813 the appropriate *level* identifier for the protocol controlling the option. For example, to indicate  
40814 that an option is interpreted by the TCP (Transport Control Protocol), set *level* to IPPROTO\_TCP  
40815 as defined in the <netinet/in.h> header.40816 The *option\_name* argument specifies a single option to set. The *option\_name* argument and any  
40817 specified options are passed uninterpreted to the appropriate protocol module for  
40818 interpretations. The <sys/socket.h> header defines the socket-level options. The options are as  
40819 follows:40820 SO\_DEBUG Turns on recording of debugging information. This option enables or  
40821 disables debugging in the underlying protocol modules. This option takes  
40822 an **int** value. This is a Boolean option.40823 SO\_BROADCAST Permits sending of broadcast messages, if this is supported by the  
40824 protocol. This option takes an **int** value. This is a Boolean option.40825 SO\_REUSEADDR Specifies that the rules used in validating addresses supplied to *bind()*  
40826 should allow reuse of local addresses, if this is supported by the protocol.  
40827 This option takes an **int** value. This is a Boolean option.40828 SO\_KEEPALIVE Keeps connections active by enabling the periodic transmission of  
40829 messages, if this is supported by the protocol. This option takes an **int**  
40830 value.40831 If the connected socket fails to respond to these messages, the connection  
40832 is broken and threads writing to that socket are notified with a SIGPIPE  
40833 signal.

40834 This is a Boolean option.

40835 SO\_LINGER Lingers on a *close()* if data is present. This option controls the action  
40836 taken when unsent messages queue on a socket and *close()* is performed.  
40837 If SO\_LINGER is set, the system blocks the process during *close()* until it  
40838 can transmit the data or until the time expires. If SO\_LINGER is not  
40839 specified, and *close()* is issued, the system handles the call in a way that  
40840 allows the process to continue as quickly as possible. This option takes a  
40841 **linger** structure, as defined in the <sys/socket.h> header, to specify the  
40842 state of the option and linger interval.40843 SO\_OOBINLINE Leaves received out-of-band data (data marked urgent) inline. This  
40844 option takes an **int** value. This is a Boolean option.

|       |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 40845 | SO_SNDBUF           | Sets send buffer size. This option takes an <b>int</b> value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 40846 | SO_RCVBUF           | Sets receive buffer size. This option takes an <b>int</b> value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 40847 | SO_DONTROUTE        | Requests that outgoing messages bypass the standard routing facilities. The destination shall be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option takes an <b>int</b> value. This is a Boolean option.                                                                                                                                                                                                                                                                                                                    |
| 40848 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40849 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40850 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40851 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40852 | SO_RCVLOWAT         | Sets the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different than that returned; for example, out-of-band data.) This option takes an <b>int</b> value. Note that not all implementations allow this option to be set.                                                                                               |
| 40853 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40854 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40855 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40856 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40857 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40858 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40859 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40860 | SO_RCVTIMEO         | Sets the timeout value that specifies the maximum amount of time an input function waits until it completes. It accepts a <b>timeval</b> structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data is received. The default for this option is zero, which indicates that a receive operation shall not time out. This option takes a <b>timeval</b> structure. Note that not all implementations allow this option to be set. |
| 40861 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40862 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40863 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40864 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40865 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40866 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40867 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40868 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40869 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40870 | SO_SNDLOWAT         | Sets the minimum number of bytes to process for socket output operations. Non-blocking output operations shall process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option takes an <b>int</b> value. Note that not all implementations allow this option to be set.                                                                                                                                                                                                                                                                                                                                           |
| 40871 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40872 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40873 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40874 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40875 | SO_SNDTIMEO         | Sets the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent. If a send operation has blocked for this time, it shall return with a partial count or with <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data is sent. The default for this option is zero, which indicates that a send operation shall not time out. This option stores a <b>timeval</b> structure. Note that not all implementations allow this option to be set.                                                                                                                                                                                |
| 40876 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40877 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40878 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40879 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40880 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40881 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40882 |                     | For Boolean options, 0 indicates that the option is disabled and 1 indicates that the option is enabled.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 40883 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40884 |                     | Options at other protocol levels vary in format and name.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 40885 | <b>RETURN VALUE</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40886 |                     | Upon successful completion, <i>setsockopt()</i> shall return 0. Otherwise, -1 shall be returned and <i>errno</i> set to indicate the error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 40887 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40888 | <b>ERRORS</b>       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40889 |                     | The <i>setsockopt()</i> function shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 40890 | [EBADF]             | The <i>socket</i> argument is not a valid file descriptor.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

|       |                          |                                                                                                                                                                                                                                                                                                                                                               |
|-------|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 40891 | [EDOM]                   | The send and receive timeout values are too big to fit into the timeout fields in the socket structure.                                                                                                                                                                                                                                                       |
| 40892 |                          |                                                                                                                                                                                                                                                                                                                                                               |
| 40893 | [EINVAL]                 | The specified option is invalid at the specified socket level or the socket has been shut down.                                                                                                                                                                                                                                                               |
| 40894 |                          |                                                                                                                                                                                                                                                                                                                                                               |
| 40895 | [EISCONN]                | The socket is already connected, and a specified option cannot be set while the socket is connected.                                                                                                                                                                                                                                                          |
| 40896 |                          |                                                                                                                                                                                                                                                                                                                                                               |
| 40897 | [ENOPROTOOPT]            |                                                                                                                                                                                                                                                                                                                                                               |
| 40898 |                          | The option is not supported by the protocol.                                                                                                                                                                                                                                                                                                                  |
| 40899 | [ENOTSOCK]               | The <i>socket</i> argument does not refer to a socket.                                                                                                                                                                                                                                                                                                        |
| 40900 |                          | The <i>setsockopt()</i> function may fail if:                                                                                                                                                                                                                                                                                                                 |
| 40901 | [ENOMEM]                 | There was insufficient memory available for the operation to complete.                                                                                                                                                                                                                                                                                        |
| 40902 | [ENOBUFS]                | Insufficient resources are available in the system to complete the call.                                                                                                                                                                                                                                                                                      |
| 40903 | <b>EXAMPLES</b>          |                                                                                                                                                                                                                                                                                                                                                               |
| 40904 |                          | None.                                                                                                                                                                                                                                                                                                                                                         |
| 40905 | <b>APPLICATION USAGE</b> |                                                                                                                                                                                                                                                                                                                                                               |
| 40906 |                          | The <i>setsockopt()</i> function provides an application program with the means to control socket behavior. An application program can use <i>setsockopt()</i> to allocate buffer space, control timeouts, or permit socket data broadcasts. The <code>&lt;sys/socket.h&gt;</code> header defines the socket-level options available to <i>setsockopt()</i> . |
| 40907 |                          |                                                                                                                                                                                                                                                                                                                                                               |
| 40908 |                          |                                                                                                                                                                                                                                                                                                                                                               |
| 40909 |                          |                                                                                                                                                                                                                                                                                                                                                               |
| 40910 |                          | Options may exist at multiple protocol levels. The SO_ options are always present at the uppermost socket level.                                                                                                                                                                                                                                              |
| 40911 |                          |                                                                                                                                                                                                                                                                                                                                                               |
| 40912 | <b>RATIONALE</b>         |                                                                                                                                                                                                                                                                                                                                                               |
| 40913 |                          | None.                                                                                                                                                                                                                                                                                                                                                         |
| 40914 | <b>FUTURE DIRECTIONS</b> |                                                                                                                                                                                                                                                                                                                                                               |
| 40915 |                          | None.                                                                                                                                                                                                                                                                                                                                                         |
| 40916 | <b>SEE ALSO</b>          |                                                                                                                                                                                                                                                                                                                                                               |
| 40917 |                          | Section 2.10 (on page 562), <i>bind()</i> , <i>endprotoent()</i> , <i>getsockopt()</i> , <i>socket()</i> , the Base Definitions                                                                                                                                                                                                                               |
| 40918 |                          | volume of IEEE Std. 1003.1-200x, <code>&lt;netinet/in.h&gt;</code> , <code>&lt;sys/socket.h&gt;</code>                                                                                                                                                                                                                                                        |
| 40919 | <b>CHANGE HISTORY</b>    |                                                                                                                                                                                                                                                                                                                                                               |
| 40920 |                          | First released in Issue 6. Derived from the XNS, Issue 5.2 specification.                                                                                                                                                                                                                                                                                     |



40921 **NAME**

40922            setstate — switch pseudorandom number generator state arrays

40923 **SYNOPSIS**

40924 xSI        #include &lt;stdlib.h&gt;

40925            char \*setstate(const char \*state);

40926

40927 **DESCRIPTION**40928            Refer to *initstate()*.

40929 **NAME**

40930           setuid — set user ID

40931 **SYNOPSIS**

40932           #include &lt;unistd.h&gt;

40933           int setuid(uid\_t uid);

40934 **DESCRIPTION**40935           If the process has appropriate privileges, *setuid()* shall set the real user ID, effective user ID, and  
40936           the saved set-user-ID of the calling process to *uid*.40937           If the process does not have appropriate privileges, but *uid* is equal to the real user ID or the  
40938           saved set-user-ID, *setuid()* shall set the effective user ID to *uid*; the real user ID and saved set-  
40939           user-ID shall remain unchanged.40940           The *setuid()* function shall not affect the supplementary group list in any way.40941 **RETURN VALUE**40942           Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
40943           indicate the error.40944 **ERRORS**40945           The *setuid()* function shall fail, return -1, and set *errno* to the corresponding value if one or more  
40946           of the following are true:40947           [EINVAL]           The value of the *uid* argument is invalid and not supported by the  
40948           implementation.40949           [EPERM]           The process does not have appropriate privileges and *uid* does not match the  
40950           real user ID or the saved set-user-ID.40951 **EXAMPLES**

40952           None.

40953 **APPLICATION USAGE**

40954           None.

40955 **RATIONALE**40956           The various behaviors of the *setuid()* and *setgid()* functions when called by non-privileged  
40957           processes reflect the behavior of different historical implementations. For portability, it is  
40958           recommended that new non-privileged applications use the *seteuid()* and *setegid()* functions  
40959           instead.40960           The saved set-user-ID capability allows a program to regain the effective user ID established at  
40961           the last *exec* call. Similarly, the saved set-group-ID capability allows a program to regain the  
40962           effective group ID established at the last *exec* call. These capabilities are derived from System V.  
40963           Without them, a program might have to run as superuser in order to perform the same  
40964           functions, because superuser can write on the user's files. This is a problem because such a  
40965           program can write on any user's files, and so must be carefully written to emulate the  
40966           permissions of the calling process properly. In System V, these capabilities have traditionally  
40967           been implemented only via the *setuid()* and *setgid()* functions for non-privileged processes. The  
40968           fact that the behavior of those functions was different for privileged processes made them  
40969           difficult to use. The POSIX.1-1990 standard defined the *setuid()* function to behave differently  
40970           for privileged and unprivileged users. When the caller had the appropriate privilege, the  
40971           function set the calling process's real user ID, effective user ID, and saved set-user ID on  
40972           implementations that supported it. When the caller did not have the appropriate privilege, the  
40973           function set only the effective user ID, subject to permission checks. The former use is generally  
40974           needed for utilities like *login* and *su*, which are not portable applications and thus outside the

40975 scope of IEEE Std. 1003.1-200x. These utilities wish to change the user ID irrevocably to a new  
 40976 value, generally that of an unprivileged user. The latter use is needed for portable applications  
 40977 that are installed with the set-user-ID bit and need to perform operations using the real user ID.

40978 IEEE Std. 1003.1-200x augments the latter functionality with a mandatory feature named  
 40979 `_POSIX_SAVED_IDS`. This feature permits a set-user-ID application to switch its effective user  
 40980 ID back and forth between the values of its *exec*-time real user ID and effective user ID.  
 40981 Unfortunately, the POSIX.1-1990 standard did not permit a portable application using this  
 40982 feature to work properly when it happened to be executed with the (implementation-defined)  
 40983 appropriate privilege. Furthermore, the application did not even have a means to tell whether it  
 40984 had this privilege. Because the saved set-user-ID feature is quite desirable for applications, as  
 40985 evidenced by the fact that NIST required it in FIPS 151-2, it has been mandated by  
 40986 IEEE Std. 1003.1-200x. However, there are implementors who have been reluctant to support it  
 40987 given the limitation described above.

40988 The 4.3BSD system handles the problem by supporting separate functions: *setuid()* (which  
 40989 always sets both the real and effective user IDs, like *setuid()* in IEEE Std. 1003.1-200x for  
 40990 privileged users), and *seteuid()* (which always sets just the effective user ID, like *setuid()* in  
 40991 IEEE Std. 1003.1-200x for non-privileged users). This separation of functionality into distinct  
 40992 functions seems desirable. 4.3BSD does not support the saved set-user-ID feature. It supports  
 40993 similar functionality of switching the effective user ID back and forth via *setreuid()*, which  
 40994 permits reversing the real and effective user IDs. This model seems less desirable than the saved  
 40995 set-user-ID because the real user ID changes as a side effect. The current 4.4BSD includes saved  
 40996 effective IDs and uses them for *seteuid()* and *setegid()* as described above. The *setreuid()* and  
 40997 *setregid()* functions will be deprecated or removed.

40998 The solution here is:

- 40999 • Require that all implementations support the functionality of the saved set-user-ID, which is  
 41000 set by the *exec* functions and by privileged calls to *setuid()*.
- 41001 • Add the *seteuid()* and *setegid()* functions as portable alternatives to *setuid()* and *setgid()* for  
 41002 non-privileged and privileged processes.

41003 Historical systems have provided two mechanisms for a set-user-ID process to change its  
 41004 effective user ID to be the same as its real user ID in such a way that it could return to the  
 41005 original effective user ID: the use of the *setuid()* function in the presence of a saved set-user-ID,  
 41006 or the use of the BSD *setreuid()* function, which was able to swap the real and effective user IDs.  
 41007 The changes included in IEEE Std. 1003.1-200x provide a new mechanism using *seteuid()*  
 41008 in conjunction with a saved set-user-ID. Thus, all implementations with the new *seteuid()*  
 41009 mechanism will have a saved set-user-ID for each process, and most of the behavior controlled  
 41010 by `_POSIX_SAVED_IDS` has been changed to agree with the case where the option was defined.  
 41011 The *kill()* function is an exception. Implementors of the new *seteuid()* mechanism will generally  
 41012 be required to maintain compatibility with the older mechanisms previously supported by their  
 41013 systems. However, compatibility with this use of *setreuid()* and with the `_POSIX_SAVED_IDS`  
 41014 behavior of *kill()* is unfortunately complicated. If an implementation with a saved set-user-ID  
 41015 allows a process to use *setreuid()* to swap its real and effective user IDs, but were to leave the  
 41016 saved set-user-ID unmodified, the process would then have an effective user ID equal to the  
 41017 original real user ID, and both real and saved set-user-ID would be equal to the original effective  
 41018 user ID. In that state, the real user would be unable to kill the process, even though the effective  
 41019 user ID of the process matches that of the real user, if the *kill()* behavior of `_POSIX_SAVED_IDS`  
 41020 was used. This is obviously not acceptable. The alternative choice, which is used in at least one  
 41021 implementation, is to change the saved set-user-ID to the effective user ID during most calls to  
 41022 *setreuid()*. The standard developers considered that alternative to be less correct than the  
 41023 retention of the old behavior of *kill()* in such systems. Current conforming applications shall

41024 accommodate either behavior from *kill()*, and there appears to be no strong reason for *kill()* to  
41025 check the saved set-user-ID rather than the effective user ID.

#### 41026 FUTURE DIRECTIONS

41027 None.

#### 41028 SEE ALSO

41029 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, the  
41030 Base Definitions volume of IEEE Std. 1003.1-200x, <**sys/types.h**>, <**unistd.h**>

#### 41031 CHANGE HISTORY

41032 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 41033 Issue 4

41034 The <**sys/types.h**> header is now marked as optional (OH); this header need not be included on  
41035 XSI-conformant systems.

41036 The <**unistd.h**> header is added to the SYNOPSIS section.

41037 The following change is incorporated for alignment with the FIPS requirements:

- 41038 • All references to the saved set-user-ID are marked as extensions. This is because Issue 4  
41039 defines this mechanism as mandatory, whereas the ISO POSIX-1 standard defines that it is  
41040 only supported if `_POSIX_SAVED_IDS` is set.

#### 41041 Issue 6

41042 In the SYNOPSIS, the inclusion of <**sys/types.h**> is no longer required.

41043 The following new requirements on POSIX implementations derive from alignment with the  
41044 Single UNIX Specification:

- 41045 • The requirement to include <**sys/types.h**> has been removed. Although <**sys/types.h**> was  
41046 required for conforming implementations of previous POSIX specifications, it was not  
41047 required for UNIX applications.
- 41048 • The functionality associated with `_POSIX_SAVED_IDS` is now mandatory. This is a FIPS  
41049 requirement.

41050 The following changes were made to align with the IEEE P1003.1a draft standard:

- 41051 • The effects of *setuid()* in processes without appropriate privileges are changed.
- 41052 • A requirement that the supplementary group list is not affected is added.

41053 **NAME**

41054 setutxent — reset user accounting database to first entry

41055 **SYNOPSIS**

41056 xSI #include &lt;utmpx.h&gt;

41057 void setutxent(void);

41058

41059 **DESCRIPTION**41060 Refer to *endutxent()*.

41061 **NAME**

41062 setvbuf — assign buffering to a stream

41063 **SYNOPSIS**

41064 #include &lt;stdio.h&gt;

41065 int setvbuf(FILE \*restrict *stream*, char \*restrict *buf*, int *type*,  
41066 size\_t *size*);41067 **DESCRIPTION**41068 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
41069 conflict between the requirements described here and the ISO C standard is unintentional. This  
41070 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.41071 The *setvbuf()* function may be used after the stream pointed to by *stream* is associated with an  
41072 open file but before any other operation (other than an unsuccessful call to *setvbuf()*) is  
41073 performed on the stream. The argument *type* determines how *stream* shall be buffered, as  
41074 follows:

- 41075
- {\_IOFBF} shall cause input/output to be fully buffered.
  - {\_IOLBF} shall cause input/output to be line buffered.
  - {\_IONBF} shall cause input/output to be unbuffered.

41078 If *buf* is not a null pointer, the array it points to may be used instead of a buffer allocated by  
41079 *setvbuf()* and the argument *size* specifies the size of the array; otherwise, *size* may determine the  
41080 size of a buffer allocated by the *setvbuf()* function. The contents of the array at any time are  
41081 indeterminate.

41082 For information about streams, see Section 2.5 (on page 535).

41083 **RETURN VALUE**41084 Upon successful completion, *setvbuf()* shall return 0. Otherwise, it shall return a non-zero value  
41085 cx if an invalid value is given for *type* or if the request cannot be honored, and may set *errno* to  
41086 indicate the error.41087 **ERRORS**41088 The *setvbuf()* function may fail if:41089 cx [EBADF] The file descriptor underlying *stream* is not valid.41090 **EXAMPLES**

41091 None.

41092 **APPLICATION USAGE**41093 A common source of error is allocating buffer space as an “automatic” variable in a code block,  
41094 and then failing to close the stream in the same block.41095 With *setvbuf()*, allocating a buffer of *size* bytes does not necessarily imply that all of *size* bytes are  
41096 used for the buffer area.41097 Applications should note that many implementations only provide line buffering on input from  
41098 terminal devices.41099 **RATIONALE**

41100 None.

41101 **FUTURE DIRECTIONS**

41102 None.

41103 **SEE ALSO**41104 *fopen()*, *setbuf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**stdio.h**>41105 **CHANGE HISTORY**

41106 First released in Issue 1. Derived from Issue 1 of the SVID.

41107 **Issue 4**

41108 The second paragraph of the DESCRIPTION is now in Section 2.5 (on page 535).

41109 The [EBADF] error is marked as an extension.

41110 The APPLICATION USAGE section is expanded.

41111 The following change is incorporated for alignment with the ISO C standard:

- 41112
- This function is no longer marked as an extension.

41113 **Issue 6**

41114 Extensions beyond the ISO C standard are now marked.

41115 The *setvbuf()* prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

## 41116 NAME

41117 shm\_open — open a shared memory object (**REALTIME**)

## 41118 SYNOPSIS

41119 SHM #include &lt;sys/mman.h&gt;

41120 int shm\_open(const char \*name, int oflag, mode\_t mode);

41121

## 41122 DESCRIPTION

41123 The *shm\_open()* function establishes a connection between a shared memory object and a file  
 41124 descriptor. It creates an open file description that refers to the shared memory object and a file  
 41125 descriptor that refers to that open file description. The file descriptor is used by other functions  
 41126 to refer to that shared memory object. The *name* argument points to a string naming a shared  
 41127 memory object. It is unspecified whether the name appears in the file system and is visible to  
 41128 other functions that take path names as arguments. The *name* argument conforms to the  
 41129 construction rules for a path name. If *name* begins with the slash character, then processes calling  
 41130 *shm\_open()* with the same value of *name* refer to the same shared memory object, as long as that  
 41131 name has not been removed. If *name* does not begin with the slash character, the effect is  
 41132 implementation-defined. The interpretation of slash characters other than the leading slash  
 41133 character in *name* is implementation-defined.

41134 If successful, *shm\_open()* shall return a file descriptor for the shared memory object that is the  
 41135 lowest numbered file descriptor not currently open for that process. The open file description is  
 41136 new, and therefore the file descriptor does not share it with any other processes. It is unspecified  
 41137 whether the file offset is set. The FD\_CLOEXEC file descriptor flag associated with the new file  
 41138 descriptor is set.

41139 The file status flags and file access modes of the open file description are according to the value  
 41140 of *oflag*. The *oflag* argument is the bitwise-inclusive OR of the following flags defined in the  
 41141 header <fcntl.h>. Applications specify exactly one of the first two values (access modes) below  
 41142 in the value of *oflag*:

41143 O\_RDONLY Open for read access only.

41144 O\_RDWR Open for read or write access.

41145 Any combination of the remaining flags may be specified in the value of *oflag*:

41146 O\_CREAT If the shared memory object exists, this flag has no effect, except as noted  
 41147 under O\_EXCL below. Otherwise, the shared memory object is created; the user ID of the shared memory object shall be set to the effective user ID of the  
 41148 process; the group ID of the shared memory object is set to a system default group ID or to the effective group ID of the process. The permission bits of the  
 41149 shared memory object shall be set to the value of the *mode* argument except  
 41150 those set in the file mode creation mask of the process. When bits in *mode*  
 41151 other than the file permission bits are set, the effect is unspecified. The *mode*  
 41152 argument does not affect whether the shared memory object is opened for  
 41153 reading, for writing, or for both. The shared memory object has a size of zero.  
 41154  
 41155

41156 O\_EXCL If O\_EXCL and O\_CREAT are set, *shm\_open()* fails if the shared memory  
 41157 object exists. The check for the existence of the shared memory object and the  
 41158 creation of the object if it does not exist is atomic with respect to other  
 41159 processes executing *shm\_open()* naming the same shared memory object with  
 41160 O\_EXCL and O\_CREAT set. If O\_EXCL is set and O\_CREAT is not set, the  
 41161 result is undefined.



41162 O\_TRUNC If the shared memory object exists, and it is successfully opened O\_RDWR,  
 41163 the object shall be truncated to zero length and the mode and owner shall be  
 41164 unchanged by this function call. The result of using O\_TRUNC with  
 41165 O\_RDONLY is undefined.

41166 When a shared memory object is created, the state of the shared memory object, including all  
 41167 data associated with the shared memory object, persists until the shared memory object is  
 41168 unlinked and all other references are gone. It is unspecified whether the name and shared  
 41169 memory object state remain valid after a system reboot.

#### 41170 RETURN VALUE

41171 Upon successful completion, the *shm\_open()* function shall return a non-negative integer  
 41172 representing the lowest numbered unused file descriptor. Otherwise, it shall return `-1` and set  
 41173 *errno* to indicate the error.

#### 41174 ERRORS

41175 The *shm\_open()* function shall fail if:

41176 [EACCES] The shared memory object exists and the permissions specified by *oflag* are  
 41177 denied, or the shared memory object does not exist and permission to create  
 41178 the shared memory object is denied, or O\_TRUNC is specified and write  
 41179 permission is denied.

41180 [EEXIST] O\_CREAT and O\_EXCL are set and the named shared memory object already  
 41181 exists.

41182 [EINTR] The *shm\_open()* operation was interrupted by a signal.

41183 [EINVAL] The *shm\_open()* operation is not supported for the given name.

41184 [EMFILE] Too many file descriptors are currently in use by this process.

41185 [ENAMETOOLONG]

41186 The length of the *name* argument exceeds {PATH\_MAX} or a path name  
 41187 component is longer than {NAME\_MAX}.

41188 [ENFILE] Too many shared memory objects are currently open in the system.

41189 [ENOENT] O\_CREAT is not set and the named shared memory object does not exist.

41190 [ENOSPC] There is insufficient space for the creation of the new shared memory object.

#### 41191 EXAMPLES

41192 None.

#### 41193 APPLICATION USAGE

41194 None.

#### 41195 RATIONALE

41196 When the Memory Mapped Files option is supported, the normal *open()* call is used to obtain a  
 41197 descriptor to a file to be mapped according to existing practice with *mmap()*. When the Shared  
 41198 Memory Objects option is supported, the *shm\_open()* function is used to obtain a descriptor to  
 41199 the shared memory object to be mapped.

41200 There is ample precedent for having a file descriptor represent several types of objects. In the  
 41201 POSIX.1-1990 standard, a file descriptor can represent a file, a pipe, a FIFO, a tty, or a directory.  
 41202 Many implementations simply have an operations vector, which is indexed by the file descriptor  
 41203 type and does very different operations. Note that in some cases the file descriptor passed to  
 41204 generic operations on file descriptors are returned by *open()* or *creat()* and in some cases  
 41205 returned by alternate functions, such as *pipe()*. The latter technique is used by *shm\_open()*.

41206 Note that such shared memory objects can actually be implemented as mapped files. In both  
41207 cases, the size can be set after the open using *ftruncate()*. The *shm\_open()* function itself does not  
41208 create a shared object of a specified size because this would duplicate an extant function that set  
41209 the size of an object referenced by a file descriptor.

41210 On implementations where memory objects are implemented using the existing file system, the  
41211 *shm\_open()* function may be implemented using a macro that invokes *open()*, and the  
41212 *shm\_unlink()* function may be implemented using a macro that invokes *unlink()*.

41213 For implementations without a permanent file system, the definition of the name of the memory  
41214 objects is allowed not to survive a system reboot. Note that this allows systems with a  
41215 permanent file system to implement memory objects as data structures internal to the  
41216 implementation as well.

41217 On implementations that choose to implement memory objects using memory directly, a  
41218 *shm\_open()* followed by a *ftruncate()* and *close()* can be used to preallocate a shared memory  
41219 area and to set the size of that preallocation. This may be necessary for systems without virtual  
41220 memory hardware support in order to ensure that the memory is contiguous.

41221 The set of valid open flags to *shm\_open()* was restricted to *O\_RDONLY*, *O\_RDWR*, *O\_CREAT*,  
41222 and *O\_TRUNC* because these could be easily implemented on most memory mapping systems.  
41223 This volume of IEEE Std. 1003.1-200x is silent on the results if the implementation cannot supply  
41224 the requested file access because of implementation-defined reasons, including hardware ones.

41225 The error conditions [EACCES] and [ENOTSUP] are provided to inform the application that the  
41226 implementation cannot complete a request.

41227 [EACCES] indicates for implementation-defined reasons, probably hardware-related, that the  
41228 implementation cannot comply with a requested mode because it conflicts with another  
41229 requested mode. An example might be that an application desires to open a memory object two  
41230 times, mapping different areas with different access modes. If the implementation cannot map a  
41231 single area into a process space in two places, which would be required if different access modes  
41232 were required for the two areas, then the implementation may inform the application at the time  
41233 of the second open.

41234 [ENOTSUP] indicates for implementation-defined reasons, probably hardware-related, that the  
41235 implementation cannot comply with a requested mode at all. An example would be that the  
41236 hardware of the implementation cannot support write-only shared memory areas.

41237 On all implementations, it may be desirable to restrict the location of the memory objects to  
41238 specific file systems for performance (such as a RAM disk) or implementation-defined reasons  
41239 (shared memory supported directly only on certain file systems). The *shm\_open()* function may  
41240 be used to enforce these restrictions. There are a number of methods available to the application  
41241 to determine an appropriate name of the file or the location of an appropriate directory. One  
41242 way is from the environment via *getenv()*. Another would be from a configuration file.

41243 This volume of IEEE Std. 1003.1-200x specifies that memory objects have initial contents of zero  
41244 when created. This is consistent with current behavior for both files and newly allocated  
41245 memory. For those implementations that use physical memory, it would be possible that such  
41246 implementations could simply use available memory and give it to the process uninitialized.  
41247 This, however, is not consistent with standard behavior for the uninitialized data area, the stack,  
41248 and of course, files. Finally, it is highly desirable to set the allocated memory to zero for security  
41249 reasons. Thus, initializing memory objects to zero is required.

41250 **FUTURE DIRECTIONS**

41251 None.

41252 **SEE ALSO**41253 *close()*, *dup()*, *exec*, *fcntl()*, *mmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm\_unlink()*, *umask()*, the Base  
41254 Definitions volume of IEEE Std. 1003.1-200x, <fcntl.h>, <sys/mman.h>41255 **CHANGE HISTORY**

41256 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

41257 **Issue 6**41258 The *shm\_open()* function is marked as part of the Shared Memory Objects option.41259 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
41260 implementation does not support the Shared Memory Objects option.

41261 **NAME**41262 shm\_unlink — remove a shared memory object (**REALTIME**)41263 **SYNOPSIS**

41264 SHM #include &lt;sys/mman.h&gt;

41265 int shm\_unlink(const char \* name);

41266

41267 **DESCRIPTION**41268 The *shm\_unlink()* function removes the name of the shared memory object named by the string  
41269 pointed to by *name*.41270 If one or more references to the shared memory object exist when the object is unlinked, the  
41271 name is removed before *shm\_unlink()* returns, but the removal of the memory object contents is  
41272 postponed until all open and map references to the shared memory object have been removed.41273 Even if the object continues to exist after the last *shm\_unlink()*, reuse of the name shall cause a  
41274 new shared memory object to be created.41275 **RETURN VALUE**41276 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be  
41277 returned and *errno* set to indicate the error. If -1 is returned, the named shared memory object  
41278 shall not be changed by this function call.41279 **ERRORS**41280 The *shm\_unlink()* function shall fail if:

41281 [EACCES] Permission is denied to unlink the named shared memory object.

41282 [ENAMETOOLONG]

41283 The length of the *name* argument exceeds {PATH\_MAX} or a path name  
41284 component is longer than {NAME\_MAX}.

41285 [ENOENT] The named shared memory object does not exist.

41286 **EXAMPLES**

41287 None.

41288 **APPLICATION USAGE**41289 Names of memory objects that were allocated with *open()* are deleted with *unlink()* in the usual  
41290 fashion. Names of memory objects that were allocated with *shm\_open()* are deleted with  
41291 *shm\_unlink()*. Note that the actual memory object is not destroyed until the last close and  
41292 unmap on it have occurred if it was already in use.41293 **RATIONALE**

41294 None.

41295 **FUTURE DIRECTIONS**

41296 None.

41297 **SEE ALSO**41298 *close()*, *mmap()*, *munmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm\_open()*, the Base Definitions volume  
41299 of IEEE Std. 1003.1-200x, <sys/mman.h>41300 **CHANGE HISTORY**

41301 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

41302 **Issue 6**

- 41303 The *shm\_unlink()* function is marked as part of the Shared Memory Objects option.
- 41304 In the DESCRIPTION, text is added to clarify that reusing the same name after a *shm\_unlink()*  
41305 will not attach to the old shared memory object.
- 41306 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
41307 implementation does not support the Shared Memory Objects option.

41308 **NAME**

41309 shmat — XSI shared memory attach operation

41310 **SYNOPSIS**41311 XSI 

```
#include <sys/shm.h>
```

41312 

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

41313

41314 **DESCRIPTION**

41315 The *shmat()* function operates on XSI shared memory (see the Base Definitions volume of  
 41316 IEEE Std. 1003.1-200x, Section 3.342, Shared Memory Object). It is unspecified whether this  
 41317 function interoperates with the realtime interprocess communication facilities defined in Section  
 41318 2.8 (on page 543).

41319 The *shmat()* function attaches the shared memory segment associated with the shared memory  
 41320 identifier specified by *shmid* to the address space of the calling process. The segment is attached  
 41321 at the address specified by one of the following criteria:

- 41322 • If *shmaddr* is a null pointer, the segment is attached at the first available address as selected  
 41323 by the system.
- 41324 • If *shmaddr* is not a null pointer and (*shmflg* &SHM\_RND) is non-zero, the segment is attached  
 41325 at the address given by (*shmaddr* - ((*uintptr\_t*)*shmaddr* %SHMLBA)). The character '%' is the  
 41326 C-language remainder operator.
- 41327 • If *shmaddr* is not a null pointer and (*shmflg* &SHM\_RND) is 0, the segment is attached at the  
 41328 address given by *shmaddr*.
- 41329 • The segment is attached for reading if (*shmflg* &SHM\_RDONLY) is non-zero and the calling  
 41330 process has read permission; otherwise, if it is 0 and the calling process has read and write  
 41331 permission, the segment is attached for reading and writing.

41332 **RETURN VALUE**

41333 Upon successful completion, *shmat()* shall increment the value of *shm\_nattach* in the data  
 41334 structure associated with the shared memory ID of the attached shared memory segment and  
 41335 return the segment's start address.

41336 Otherwise, the shared memory segment shall not be attached, *shmat()* shall return -1, and *errno*  
 41337 shall be set to indicate the error.

41338 **ERRORS**41339 The *shmat()* function shall fail if:

- |                                           |          |                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------------------|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 41340<br>41341                            | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page 541).                                                                                                                                                                                                                                                                                                                              |
| 41342<br>41343<br>41344<br>41345<br>41346 | [EINVAL] | The value of <i>shmid</i> is not a valid shared memory identifier, the <i>shmaddr</i> is not a null pointer, and the value of ( <i>shmaddr</i> - (( <i>uintptr_t</i> ) <i>shmaddr</i> %SHMLBA)) is an illegal address for attaching shared memory; or the <i>shmaddr</i> is not a null pointer, ( <i>shmflg</i> &SHM_RND) is 0, and the value of <i>shmaddr</i> is an illegal address for attaching shared memory. |
| 41347<br>41348                            | [EMFILE] | The number of shared memory segments attached to the calling process would exceed the system-imposed limit.                                                                                                                                                                                                                                                                                                        |
| 41349<br>41350                            | [ENOMEM] | The available data space is not large enough to accommodate the shared memory segment.                                                                                                                                                                                                                                                                                                                             |

41351 **EXAMPLES**

41352 None.

41353 **APPLICATION USAGE**

41354 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
41355 Application developers who need to use IPC should design their applications so that modules  
41356 using the IPC routines described in Section 2.7 (on page 541) can be easily modified to use the  
41357 alternative interfaces.

41358 **RATIONALE**

41359 None.

41360 **FUTURE DIRECTIONS**

41361 None.

41362 **SEE ALSO**

41363 *exec*, *exit()*, *fork()*, *shmctl()*, *shmdt()*, *shmget()*, *shm\_open()*, *shm\_unlink()*, the Base Definitions  
41364 volume of IEEE Std. 1003.1-200x, <sys/shm.h>, Section 2.7 (on page 541)

41365 **CHANGE HISTORY**

41366 First released in Issue 2. Derived from Issue 2 of the SVID.

41367 **Issue 4**

41368 The function is no longer marked as OPTIONAL FUNCTIONALITY.

41369 The &lt;sys/types.h&gt; and &lt;sys/ipc.h&gt; headers are removed from the SYNOPSIS section.

41370 The type of argument *shmaddr* is changed from **char\*** to **const void\***.

41371 The [ENOSYS] error is removed from the ERRORS section.

41372 The DESCRIPTION is clarified in several places.

41373 A FUTURE DIRECTIONS section is added warning application developers about migration to  
41374 IEEE 1003.4 interfaces for interprocess communication.

41375 **Issue 5**

41376 Moved from SHARED MEMORY to BASE.

41377 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
41378 DIRECTIONS to a new APPLICATION USAGE section.

41379 **Issue 6**

41380 The Open Group corrigenda item U021/13 has been applied.

## 41381 NAME

41382 shmctl — XSI shared memory control operations

## 41383 SYNOPSIS

41384 XSI #include &lt;sys/shm.h&gt;

41385 int shmctl(int *shmid*, int *cmd*, struct shmids \**buf*);

41386

## 41387 DESCRIPTION

41388 The *shmctl()* function operates on XSI shared memory (see the Base Definitions volume of  
 41389 IEEE Std. 1003.1-200x, Section 3.342, Shared Memory Object). It is unspecified whether this  
 41390 function interoperates with the realtime interprocess communication facilities defined in Section  
 41391 2.8 (on page 543).

41392 The *shmctl()* function provides a variety of shared memory control operations as specified by  
 41393 *cmd*. The following values for *cmd* are available:

41394 IPC\_STAT Place the current value of each member of the **shmids** data structure  
 41395 associated with *shmid* into the structure pointed to by *buf*. The contents of the  
 41396 structure are defined in <sys/shm.h>.

41397 IPC\_SET Set the value of the following members of the **shmids** data structure  
 41398 associated with *shmid* to the corresponding value found in the structure  
 41399 pointed to by *buf*:

41400 shm\_perm.uid  
 41401 shm\_perm.gid  
 41402 shm\_perm.mode Low-order nine bits.

41403 IPC\_SET can only be executed by a process that has an effective user ID equal  
 41404 to either that of a process with appropriate privileges or to the value of  
 41405 *shm\_perm.cuid* or *shm\_perm.uid* in the **shmids** data structure associated with  
 41406 *shmid*.

41407 IPC\_RMID Remove the shared memory identifier specified by *shmid* from the system and  
 41408 destroy the shared memory segment and **shmids** data structure associated  
 41409 with it. IPC\_RMID can only be executed by a process that has an effective user  
 41410 ID equal to either that of a process with appropriate privileges or to the value  
 41411 of *shm\_perm.cuid* or *shm\_perm.uid* in the **shmids** data structure associated  
 41412 with *shmid*.

## 41413 RETURN VALUE

41414 Upon successful completion, *shmctl()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 41415 indicate the error.

## 41416 ERRORS

41417 The *shmctl()* function shall fail if:

41418 [EACCES] The argument *cmd* is equal to IPC\_STAT and the calling process does not have  
 41419 read permission; see Section 2.7 (on page 541).

41420 [EINVAL] The value of *shmid* is not a valid shared memory identifier, or the value of *cmd*  
 41421 is not a valid command.

41422 [EPERM] The argument *cmd* is equal to IPC\_RMID or IPC\_SET and the effective user ID  
 41423 of the calling process is not equal to that of a process with appropriate  
 41424 privileges and it is not equal to the value of *shm\_perm.cuid* or *shm\_perm.uid* in  
 41425 the data structure associated with *shmid*.



41426 The *shmctl()* function may fail if:

41427 [EOVERFLOW] The *cmd* argument is IPC\_STAT and the *gid* or *uid* value is too large to be  
41428 stored in the structure pointed to by the *buf* argument.

41429 **EXAMPLES**

41430 None.

41431 **APPLICATION USAGE**

41432 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
41433 Application developers who need to use IPC should design their applications so that modules  
41434 using the IPC routines described in Section 2.7 (on page 541) can be easily modified to use the  
41435 alternative interfaces.

41436 **RATIONALE**

41437 None.

41438 **FUTURE DIRECTIONS**

41439 None.

41440 **SEE ALSO**

41441 *shmat()*, *shmdt()*, *shmget()*, *shm\_open()*, *shm\_unlink()*, the Base Definitions volume of  
41442 IEEE Std. 1003.1-200x, <sys/shm.h>, Section 2.7 (on page 541)

41443 **CHANGE HISTORY**

41444 First released in Issue 2. Derived from Issue 2 of the SVID.

41445 **Issue 4**

41446 The function is no longer marked as OPTIONAL FUNCTIONALITY.

41447 The <sys/types.h> and <sys/ipc.h> headers are removed from the SYNOPSIS section.

41448 The [ENOSYS] error is removed from the ERRORS section.

41449 A FUTURE DIRECTIONS section is added warning application developers about migration to  
41450 IEEE 1003.4 interfaces for interprocess communication.

41451 **Issue 4, Version 2**

41452 The ERRORS section is updated for X/OPEN UNIX conformance to include [EOVERFLOW] as  
41453 an optional error.

41454 **Issue 5**

41455 Moved from SHARED MEMORY to BASE.

41456 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
41457 DIRECTIONS to a new APPLICATION USAGE section.

41458 **NAME**

41459 shmdt — XSI shared memory detach operation

41460 **SYNOPSIS**41461 XSI 

```
#include <sys/shm.h>
```

41462 

```
int shmdt(const void *shmaddr);
```

41463

41464 **DESCRIPTION**

41465 The *shmdt()* function operates on XSI shared memory (see the Base Definitions volume of  
 41466 IEEE Std. 1003.1-200x, Section 3.342, Shared Memory Object). It is unspecified whether this  
 41467 function interoperates with the realtime interprocess communication facilities defined in Section  
 41468 2.8 (on page 543).

41469 The *shmdt()* function detaches the shared memory segment located at the address specified by  
 41470 *shmaddr* from the address space of the calling process.

41471 **RETURN VALUE**

41472 Upon successful completion, *shmdt()* shall decrement the value of *shm\_nattch* in the data  
 41473 structure associated with the shared memory ID of the attached shared memory segment and  
 41474 return 0.

41475 Otherwise, the shared memory segment shall not be detached, *shmdt()* shall return  $-1$ , and *errno*  
 41476 shall be set to indicate the error.

41477 **ERRORS**41478 The *shmdt()* function shall fail if:

41479 [EINVAL] The value of *shmaddr* is not the data segment start address of a shared  
 41480 memory segment.

41481 **EXAMPLES**

41482 None.

41483 **APPLICATION USAGE**

41484 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
 41485 Application developers who need to use IPC should design their applications so that modules  
 41486 using the IPC routines described in Section 2.7 (on page 541) can be easily modified to use the  
 41487 alternative interfaces.

41488 **RATIONALE**

41489 None.

41490 **FUTURE DIRECTIONS**

41491 None.

41492 **SEE ALSO**

41493 *exec*, *exit()*, *fork()*, *shmat()*, *shmctl()*, *shmget()*, *shm\_open()*, *shm\_unlink()*, the Base Definitions  
 41494 volume of IEEE Std. 1003.1-200x, <sys/shm.h>, Section 2.7 (on page 541)

41495 **CHANGE HISTORY**

41496 First released in Issue 2. Derived from Issue 2 of the SVID.

41497 **Issue 4**

41498 The function is no longer marked as OPTIONAL FUNCTIONALITY.

41499 The &lt;sys/types.h&gt; and &lt;sys/ipc.h&gt; headers are removed from the SYNOPSIS section.

41500 The type of argument *shmaddr* is changed from **char\*** to **const void\***.

- 41501 The DESCRIPTION is clarified in several places.
- 41502 The [ENOSYS] error is removed from the ERRORS section.
- 41503 A FUTURE DIRECTIONS section is added warning application developers about migration to
- 41504 IEEE 1003.4 interfaces for interprocess communication.
- 41505 **Issue 5**
- 41506 Moved from SHARED MEMORY to BASE.
- 41507 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
- 41508 DIRECTIONS to a new APPLICATION USAGE section.

## 41509 NAME

41510 shmget — get XSI shared memory segment

## 41511 SYNOPSIS

41512 XSI #include &lt;sys/shm.h&gt;

41513 int shmget(key\_t key, size\_t size, int shmflg);

41514

## 41515 DESCRIPTION

41516 The *shmget()* function operates on XSI shared memory (see the Base Definitions volume of  
 41517 IEEE Std. 1003.1-200x, Section 3.342, Shared Memory Object). It is unspecified whether this  
 41518 function interoperates with the realtime interprocess communication facilities defined in Section  
 41519 2.8 (on page 543).

41520 The *shmget()* function shall return the shared memory identifier associated with *key*.

41521 A shared memory identifier, associated data structure, and shared memory segment of at least  
 41522 *size* bytes (see <sys/shm.h>) are created for *key* if one of the following is true:

- 41523 • The argument *key* is equal to `IPC_PRIVATE`.
- 41524 • The argument *key* does not already have a shared memory identifier associated with it and  
 41525 (*shmflg* & `IPC_CREAT`) is non-zero.

41526 Upon creation, the data structure associated with the new shared memory identifier shall be  
 41527 initialized as follows:

- 41528 • The values of *shm\_perm.cuid*, *shm\_perm.uid*, *shm\_perm.cgid*, and *shm\_perm.gid* are set equal to  
 41529 the effective user ID and effective group ID, respectively, of the calling process.
- 41530 • The low-order nine bits of *shm\_perm.mode* are set equal to the low-order nine bits of *shmflg*.  
 41531 The value of *shm\_segsz* is set equal to the value of *size*.
- 41532 • The values of *shm\_lpid*, *shm\_nattch*, *shm\_atime*, and *shm\_dtime* are set equal to 0.
- 41533 • The value of *shm\_ctime* is set equal to the current time.

41534 When the shared memory segment is created, it shall be initialized with all zero values.

## 41535 RETURN VALUE

41536 Upon successful completion, *shmget()* shall return a non-negative integer, namely a shared  
 41537 memory identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

## 41538 ERRORS

41539 The *shmget()* function shall fail if:

- |                                  |          |                                                                                                                                                                                                                                                                                 |
|----------------------------------|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 41540<br>41541<br>41542          | [EACCES] | A shared memory identifier exists for <i>key</i> but operation permission as specified by the low-order nine bits of <i>shmflg</i> would not be granted; see Section 2.7 (on page 541).                                                                                         |
| 41543<br>41544                   | [EEXIST] | A shared memory identifier exists for the argument <i>key</i> but ( <i>shmflg</i> & <code>IPC_CREAT</code> ) && ( <i>shmflg</i> & <code>IPC_EXCL</code> ) is non-zero.                                                                                                          |
| 41545<br>41546<br>41547<br>41548 | [EINVAL] | The value of <i>size</i> is less than the system-imposed minimum or greater than the system-imposed maximum, or a shared memory identifier exists for the argument <i>key</i> but the size of the segment associated with it is less than <i>size</i> and <i>size</i> is not 0. |
| 41549<br>41550                   | [ENOENT] | A shared memory identifier does not exist for the argument <i>key</i> and ( <i>shmflg</i> & <code>IPC_CREAT</code> ) is 0.                                                                                                                                                      |

41551 [ENOMEM] A shared memory identifier and associated shared memory segment shall be  
 41552 created, but the amount of available physical memory is not sufficient to fill  
 41553 the request.

41554 [ENOSPC] A shared memory identifier is to be created, but the system-imposed limit on  
 41555 the maximum number of allowed shared memory identifiers system-wide  
 41556 would be exceeded.

41557 **EXAMPLES**

41558 None.

41559 **APPLICATION USAGE**

41560 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
 41561 Application developers who need to use IPC should design their applications so that modules  
 41562 using the IPC routines described in Section 2.7 (on page 541) can be easily modified to use the  
 41563 alternative interfaces.

41564 **RATIONALE**

41565 None.

41566 **FUTURE DIRECTIONS**

41567 None.

41568 **SEE ALSO**

41569 *shmat()*, *shmctl()*, *shmdt()*, *shm\_open()*, *shm\_unlink()*, the Base Definitions volume of  
 41570 IEEE Std. 1003.1-200x, <sys/shm.h>, Section 2.7 (on page 541)

41571 **CHANGE HISTORY**

41572 First released in Issue 2. Derived from Issue 2 of the SVID.

41573 **Issue 4**

41574 The function is no longer marked as OPTIONAL FUNCTIONALITY.

41575 The <sys/types.h> and <sys/ipc.h> headers are removed from the SYNOPSIS section.

41576 The [ENOSYS] error is removed from the ERRORS section.

41577 A FUTURE DIRECTIONS section is added warning application developers about migration to  
 41578 IEEE 1003.4 interfaces for interprocess communication.

41579 **Issue 5**

41580 Moved from SHARED MEMORY to BASE.

41581 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
 41582 DIRECTIONS to a new APPLICATION USAGE section.

41583 **NAME**

41584 shutdown — shut down socket send and receive operations

41585 **SYNOPSIS**

41586 #include &lt;sys/socket.h&gt;

41587 int shutdown(int *socket*, int *how*);41588 **DESCRIPTION**41589 The *shutdown()* function shall cause all or part of a full-duplex connection on the socket  
41590 associated with the file descriptor *socket* to be shut down.41591 The *shutdown()* function takes the following arguments:

|       |               |                                                            |
|-------|---------------|------------------------------------------------------------|
| 41592 | <i>socket</i> | Specifies the file descriptor of the socket.               |
| 41593 | <i>how</i>    | Specifies the type of shutdown. The values are as follows: |
| 41594 | SHUT_RD       | Disables further receive operations.                       |
| 41595 | SHUT_WR       | Disables further send operations.                          |
| 41596 | SHUT_RDWR     | Disables further send and receive operations.              |

41597 The *shutdown()* function disables subsequent send and/or receive operations on a socket,  
41598 depending on the value of the *how* argument.41599 **RETURN VALUE**41600 Upon successful completion, *shutdown()* shall return 0; otherwise, -1 shall be returned and *errno*  
41601 set to indicate the error.41602 **ERRORS**41603 The *shutdown()* function shall fail if:

|       |            |                                                            |
|-------|------------|------------------------------------------------------------|
| 41604 | [EBADF]    | The <i>socket</i> argument is not a valid file descriptor. |
| 41605 | [EINVAL]   | The <i>how</i> argument is invalid.                        |
| 41606 | [ENOTCONN] | The socket is not connected.                               |
| 41607 | [ENOTSOCK] | The <i>socket</i> argument does not refer to a socket.     |

41608 The *shutdown()* function may fail if:

41609 [ENOBUFS] Insufficient resources were available in the system to perform the operation. |

41610 **EXAMPLES**

41611 None.

41612 **APPLICATION USAGE**

41613 None.

41614 **RATIONALE**

41615 None.

41616 **FUTURE DIRECTIONS**

41617 None.

41618 **SEE ALSO**41619 *getsockopt()*, *read()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *send()*, *sendto()*, *setsockopt()*, *socket()*,  
41620 *write()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/socket.h> |

41621 **CHANGE HISTORY**

41622 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

## 41623 NAME

41624 sigaction — examine and change signal action

## 41625 SYNOPSIS

41626 #include &lt;signal.h&gt;

41627 int sigaction(int *sig*, const struct sigaction \*restrict *act*,  
41628 struct sigaction \*restrict *oact*);

## 41629 DESCRIPTION

41630 The *sigaction()* function allows the calling process to examine and/or specify the action to be  
41631 associated with a specific signal. The argument *sig* specifies the signal; acceptable values are  
41632 defined in <signal.h>.41633 The structure **sigaction**, used to describe an action to be taken, is defined in the header  
41634 <signal.h> to include at least the following members:

41635

41636

| Member Type                           | Member Name                         | Description                                                                                                                                |
|---------------------------------------|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| void(*) (int)<br>sigset_t             | <i>sa_handler</i><br><i>sa_mask</i> | SIG_DFL, SIG_IGN, or pointer to a function.<br>Additional set of signals to be blocked<br>during execution of signal-catching<br>function. |
| int                                   | <i>sa_flags</i>                     | Special flags to affect behavior of signal.                                                                                                |
| void(*) (int,<br>siginfo_t *, void *) | <i>sa_sigaction</i>                 | Signal-catching function.                                                                                                                  |

41643

41644 If the argument *act* is not a null pointer, it points to a structure specifying the action to be  
41645 associated with the specified signal. If the argument *oact* is not a null pointer, the action  
41646 previously associated with the signal is stored in the location pointed to by the argument *oact*. If  
41647 the argument *act* is a null pointer, signal handling is unchanged; thus, the call can be used to  
41648 enquire about the current handling of a given signal. The SIGKILL and SIGSTOP signals shall  
41649 not be added to the signal mask using this mechanism; this restriction shall be enforced by the  
41650 system without causing an error to be indicated.41651 If the SA\_SIGINFO flag (see below) is cleared in the *sa\_flags* field of the **sigaction** structure, the  
41652 *sa\_handler* field identifies the action to be associated with the specified signal. If the  
41653 SA\_SIGINFO flag is set in the *sa\_flags* field, and the implementation supports the Realtime  
41654 Signals Extension option or the X/Open System Interfaces Extension option, the *sa\_sigaction*  
41655 field specifies a signal-catching function. If the SA\_SIGINFO bit is cleared and the *sa\_handler*  
41656 field specifies a signal-catching function, or if the SA\_SIGINFO bit is set, the *sa\_mask* field  
41657 identifies a set of signals that shall be added to the signal mask of the thread before the signal-  
41658 catching function is invoked. If the *sa\_handler* field specifies a signal-catching function, the  
41659 *sa\_mask* field identifies a set of signals that shall be added to the process' signal mask before the  
41660 signal-catching function is invoked.41661 The *sa\_flags* field can be used to modify the behavior of the specified signal.41662 The following flags, defined in the header <signal.h>, can be set in *sa\_flags*:

41663 SA\_NOCLDSTOP Do not generate SIGCHLD when children stop.

41664 If *sig* is SIGCHLD and the SA\_NOCLDSTOP flag is not set in *sa\_flags*, and  
41665 the implementation supports the SIGCHLD signal, then a SIGCHLD  
41666 signal shall be generated for the calling process whenever any of its child  
41667 processes stop. If *sig* is SIGCHLD and the SA\_NOCLDSTOP flag is set in  
41668 *sa\_flags*, then the implementation shall not generate a SIGCHLD signal in



|               |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 41669         |              | this way.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 41670 XSI     | SA_ONSTACK   | If set and an alternate signal stack has been declared with <i>sigaltstack()</i> or <i>sigstack()</i> , the signal shall be delivered to the calling process on that stack. Otherwise, the signal shall be delivered on the current stack.                                                                                                                                                                                                                                                                                                                                |
| 41671         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41672         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41673 XSI     | SA_RESETHAND | If set, the disposition of the signal shall be reset to SIG_DFL and the SA_SIGINFO flag shall be cleared on entry to the signal handler.                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 41674         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41675         |              | <b>Note:</b> SIGILL and SIGTRAP cannot be automatically reset when delivered; the system silently enforces this restriction.                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 41676         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41677         |              | Otherwise, the disposition of the signal shall not be modified on entry to the signal handler.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 41678         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41679         |              | In addition, if this flag is set, <i>sigaction()</i> behaves as if the SA_NODEFER flag were also set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 41680         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41681 XSI     | SA_RESTART   | This flag affects the behavior of interruptible functions; that is, those specified to fail with <i>errno</i> set to [EINTR]. If set, and a function specified as interruptible is interrupted by this signal, the function shall restart and shall not fail with [EINTR] unless otherwise specified. If the flag is not set, interruptible functions interrupted by this signal shall fail with <i>errno</i> set to [EINTR].                                                                                                                                             |
| 41682         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41683         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41684         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41685         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41686         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41687         | SA_SIGINFO   | If cleared and the signal is caught, the signal-catching function shall be entered as:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 41688         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41689         |              | <pre>void func(int signo);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 41690         |              | where <i>signo</i> is the only argument to the signal catching function. In this case, the application shall use the <i>sa_handler</i> member to describe the signal catching function and the application shall not modify the <i>sa_sigaction</i> member.                                                                                                                                                                                                                                                                                                               |
| 41691         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41692         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41693         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41694 XSI RTS |              | If SA_SIGINFO is set and the signal is caught, the signal-catching function shall be entered as:                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 41695         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41696         |              | <pre>void func(int signo, siginfo_t *info, void *context);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 41697         |              | where two additional arguments are passed to the signal catching function. The second argument shall point to an object of type <b>siginfo_t</b> explaining the reason why the signal was generated; the third argument can be cast to a pointer to an object of type <b>ucontext_t</b> to refer to the receiving process' context that was interrupted when the signal was delivered. In this case, the application shall use the <i>sa_sigaction</i> member to describe the signal catching function and the application shall not modify the <i>sa_handler</i> member. |
| 41698         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41699         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41700         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41701         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41702         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41703         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41704         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41705         |              | The <i>si_signo</i> member contains the system-generated signal number.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 41706 XSI     |              | The <i>si_errno</i> member may contain implementation-defined additional error information; if non-zero, it contains an error number identifying the condition that caused the signal to be generated.                                                                                                                                                                                                                                                                                                                                                                    |
| 41707         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41708         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41709         |              | The <i>si_code</i> member contains a code identifying the cause of the signal. If the value of <i>si_code</i> is less than or equal to 0, then the signal was generated by a process and <i>si_pid</i> and <i>si_uid</i> , respectively, indicate the process ID and the real user ID of the sender. The <b>&lt;signal.h&gt;</b> header description contains information about the signal specific contents of the                                                                                                                                                        |
| 41710         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41711         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41712         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41713         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

41714 elements of the **siginfo\_t** type.

41715 XSI **SA\_NOCLDWAIT** If set, and *sig* equals SIGCHLD, child processes of the calling processes  
41716 shall not be transformed into zombie processes when they terminate. If  
41717 the calling process subsequently waits for its children, and the process  
41718 has no unwaited-for children that were transformed into zombie  
41719 processes, it shall block until all of its children terminate, and *wait()*,  
41720 *waitid()*, and *waitpid()* shall fail and set *errno* to [ECHILD]. Otherwise,  
41721 terminating child processes shall be transformed into zombie processes,  
41722 unless SIGCHLD is set to SIG\_IGN.

41723 XSI **SA\_NODEFER** If set and *sig* is caught, *sig* shall not be added to the process' signal mask  
41724 on entry to the signal handler unless it is included in *sa\_mask*. Otherwise,  
41725 *sig* shall always be added to the process' signal mask on entry to the  
41726 signal handler.

41727 When a signal is caught by a signal-catching function installed by *sigaction()*, a new signal mask  
41728 is calculated and installed for the duration of the signal-catching function (or until a call to either  
41729 *sigprocmask()* or *sigsuspend()* is made). This mask is formed by taking the union of the current  
41730 XSI signal mask and the value of the *sa\_mask* for the signal being delivered unless SA\_NODEFER or  
41731 SA\_RESETHAND is set, and then including the signal being delivered. If and when the user's  
41732 signal handler returns normally, the original signal mask is restored.

41733 Once an action is installed for a specific signal, it remains installed until another action is  
41734 XSI explicitly requested (by another call to *sigaction()*), until the SA\_RESETHAND flag causes  
41735 resetting of the handler, or until one of the *exec* functions is called.

41736 If the previous action for *sig* had been established by *signal()*, the values of the fields returned in  
41737 the structure pointed to by *oact* are unspecified, and in particular *oact->sa\_handler* is not  
41738 necessarily the same value passed to *signal()*. However, if a pointer to the same structure or a  
41739 copy thereof is passed to a subsequent call to *sigaction()* via the *act* argument, handling of the  
41740 signal shall be as if the original call to *signal()* were repeated.

41741 If *sigaction()* fails, no new signal handler is installed.

41742 It is unspecified whether an attempt to set the action for a signal that cannot be caught or  
41743 ignored to SIG\_DFL is ignored or causes an error to be returned with *errno* set to [EINVAL].

41744 If SA\_SIGINFO is not set in *sa\_flags*, then the disposition of subsequent occurrences of *sig* when  
41745 it is already pending is implementation-defined; the signal-catching function shall be invoked  
41746 RTS with a single argument. If the implementation supports the Realtime Signals Extension option,  
41747 and if SA\_SIGINFO is set in *sa\_flags*, then subsequent occurrences of *sig* generated by *sigqueue()*  
41748 or as a result of any signal-generating function that supports the specification of an application-  
41749 defined value (when *sig* is already pending) shall be queued in FIFO order until delivered or  
41750 accepted; the signal-catching function shall be invoked with three arguments. The application  
41751 specified value is passed to the signal-catching function as the *si\_value* member of the **siginfo\_t**  
41752 structure.

41753 The result of the use of *sigaction()* and a *sigwait()* function concurrently within a process on the  
41754 same signal is unspecified.

41755 **RETURN VALUE**

41756 Upon successful completion, *sigaction()* shall return 0; otherwise, -1 shall be returned, *errno* shall  
41757 be set to indicate the error, and no new signal-catching function shall be installed.

41758 **ERRORS**41759 The *sigaction()* function shall fail if:41760 [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a  
41761 signal that cannot be caught or ignore a signal that cannot be ignored.41762 [ENOTSUP] The SA\_SIGINFO bit flag is set in the *sa\_flags* field of the **sigaction** structure,  
41763 and the implementation does not support either the Realtime Signals  
41764 Extension option, or the X/Open System Interfaces Extension option.41765 The *sigaction()* function may fail if:41766 [EINVAL] An attempt was made to set the action to SIG\_DFL for a signal that cannot be  
41767 caught or ignored (or both).41768 **EXAMPLES**

41769 None.

41770 **APPLICATION USAGE**41771 The *sigaction()* function supersedes the *signal()* function, and should be used in preference. In  
41772 particular, *sigaction()* and *signal()* should not be used in the same process to control the same  
41773 signal. The behavior of reentrant functions, as defined in the DESCRIPTION, is as specified by  
41774 this volume of IEEE Std. 1003.1-200x, regardless of invocation from a signal-catching function.  
41775 This is the only intended meaning of the statement that reentrant functions may be used in  
41776 signal-catching functions without restrictions. Applications must still consider all effects of such  
41777 functions on such things as data structures, files, and process state. In particular, application  
41778 writers need to consider the restrictions on interactions when interrupting *sleep()* and  
41779 interactions among multiple handles for a file description. The fact that any specific function is  
41780 listed as reentrant does not necessarily mean that invocation of that function from a signal-  
41781 catching function is recommended.41782 In order to prevent errors arising from interrupting non-reentrant function calls, applications  
41783 should protect calls to these functions either by blocking the appropriate signals or through the  
41784 use of some programmatic semaphore (see *semget()*, *sem\_init()*, *sem\_open()*, and so on). Note in  
41785 particular that even the “safe” functions may modify *errno*; the signal-catching function, if not  
41786 executing as an independent thread, may want to save and restore its value. Naturally, the same  
41787 principles apply to the reentrancy of application routines and asynchronous data access. Note  
41788 that *longjmp()* and *siglongjmp()* are not in the list of reentrant functions. This is because the code  
41789 executing after *longjmp()* and *siglongjmp()* can call any unsafe functions with the same danger as  
41790 calling those unsafe functions directly from the signal handler. Applications that use *longjmp()*  
41791 and *siglongjmp()* from within signal handlers require rigorous protection in order to be portable.  
41792 Many of the other functions that are excluded from the list are traditionally implemented using  
41793 either *malloc()* or *free()* functions or the standard I/O library, both of which traditionally use  
41794 data structures in a non-reentrant manner. Because any combination of different functions using  
41795 a common data structure can cause reentrancy problems, this volume of IEEE Std. 1003.1-200x  
41796 does not define the behavior when any unsafe function is called in a signal handler that  
41797 interrupts an unsafe function.41798 If the signal occurs other than as the result of calling *abort()*, *kill()*, or *raise()*, the behavior is  
41799 undefined if the signal handler calls any function in the standard library other than one of the  
41800 functions listed in the table above or refers to any object with static storage duration other than  
41801 by assigning a value to a static storage duration variable of type **volatile sig\_atomic\_t**.  
41802 Furthermore, if such a call fails, the value of *errno* is indeterminate.41803 Usually, the signal is executed on the stack that was in effect before the signal was delivered. An  
41804 alternate stack may be specified to receive a subset of the signals being caught.

41805 When the signal handler returns, the receiving process resumes execution at the point it was  
41806 interrupted unless the signal handler makes other arrangements. If *longjmp()* or *\_longjmp()* is  
41807 used to leave the signal handler, then the signal mask must be explicitly restored by the process.

41808 This volume of IEEE Std. 1003.1-200x defines the third argument of a signal handling function  
41809 when SA\_SIGINFO is set as a **void\*** instead of a **ucontext\_t\***, but without requiring type  
41810 checking. New applications should explicitly cast the third argument of the signal handling  
41811 function to **ucontext\_t\***.

41812 The BSD optional four argument signal handling function is not supported by this volume of  
41813 IEEE Std. 1003.1-200x. The BSD declaration would be:

```
41814 void handler(int sig, int code, struct sigcontext *scp,
41815 char *addr);
```

41816 where *sig* is the signal number, *code* is additional information on certain signals, *scp* is a pointer  
41817 to the sigcontext structure, and *addr* is additional address information. Much the same  
41818 information is available in the objects pointed to by the second argument of the signal handler  
41819 specified when SA\_SIGINFO is set.

#### 41820 RATIONALE

41821 Although this volume of IEEE Std. 1003.1-200x requires that signals that cannot be ignored shall  
41822 not be added to the signal mask when a signal-catching function is entered, there is no explicit  
41823 requirement that subsequent calls to *sigaction()* reflect this in the information returned in the *oact*  
41824 argument. In other words, if SIGKILL is included in the *sa\_mask* field of *act*, it is unspecified  
41825 whether or not a subsequent call to *sigaction()* returns with SIGKILL included in the *sa\_mask*  
41826 field of *oact*.

41827 The SA\_NOCLDSTOP flag, when supplied in the *act->sa\_flags* parameter, allows overloading  
41828 SIGCHLD with the System V semantics that each SIGCLD signal indicates a single terminated  
41829 child. Most portable applications that catch SIGCHLD are expected to install signal-catching  
41830 functions that repeatedly call the *waitpid()* function with the WNOHANG flag set, acting on  
41831 each child for which status is returned, until *waitpid()* returns zero. If stopped children are not of  
41832 interest, the use of the SA\_NOCLDSTOP flag can prevent the overhead from invoking the  
41833 signal-catching routine when they stop.

41834 Some historical implementations also define other mechanisms for stopping processes, such as  
41835 the *ptrace()* function. These implementations usually do not generate a SIGCHLD signal when  
41836 processes stop due to this mechanism; however, that is beyond the scope of this volume of  
41837 IEEE Std. 1003.1-200x.

41838 This volume of IEEE Std. 1003.1-200x requires that calls to *sigaction()* that supply a NULL *act*  
41839 argument succeed, even in the case of signals that cannot be caught or ignored (that is, SIGKILL  
41840 or SIGSTOP). The System V *signal()* and BSD *sigvec()* functions return [EINVAL] in these cases  
41841 and, in this respect, their behavior varies from *sigaction()*.

41842 This volume of IEEE Std. 1003.1-200x requires that *sigaction()* properly save and restore a signal  
41843 action set up by the ISO C standard *signal()* function. However, there is no guarantee that the  
41844 reverse is true, nor could there be given the greater amount of information conveyed by the  
41845 **sigaction** structure. Because of this, applications should avoid using both functions for the same  
41846 signal in the same process. Since this cannot always be avoided in case of general-purpose  
41847 library routines, they should always be implemented with *sigaction()*.

41848 It was intended that the *signal()* function should be implementable as a library routine using  
41849 *sigaction()*.

41850 The POSIX Realtime Extension extends the *sigaction()* function as specified by the POSIX.1-1990  
41851 standard to allow the application to request on a per-signal basis via an additional signal action

41852 flag that the extra parameters, including the application-defined signal value, if any, be passed  
41853 to the signal-catching function.

#### 41854 FUTURE DIRECTIONS

41855 The *fpathconf()* function is marked as an extension in the list of safe functions because it is not  
41856 included in the corresponding list in the ISO POSIX-1 standard, but it is expected to be added in  
41857 a future version.

#### 41858 SEE ALSO

41859 Section 2.4 (on page 528), *bsd\_signal()*, *kill()*, *\_longjmp()*, *longjmp()*, *raise()*, *semget()*, *sem\_init()*,  
41860 *sem\_open()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *signal()*,  
41861 *sigprocmask()*, *sigsuspend()*, *wait()*, *waitid()*, *waitpid()*, the Base Definitions volume of  
41862 IEEE Std. 1003.1-200x, <**signal.h**>, <**ucontext.h**>

#### 41863 CHANGE HISTORY

41864 First released in Issue 3.

41865 Entry included for alignment with the POSIX.1-1988 standard.

#### 41866 Issue 4

41867 The *raise()* and *signal()* functions are added to the list of functions that are either reentrant or not  
41868 interruptible by signals; *fpathconf()* is also added to this list and marked as an extension; *ustat()*  
41869 is removed from the list, as this function is withdrawn from the interface definition. It is no  
41870 longer specified whether *abort()*, *exit()*, and *longjmp()* also fall into this category of functions.

41871 The APPLICATION USAGE section is added. Most of this text is moved from the  
41872 DESCRIPTION in Issue 3.

41873 The FUTURE DIRECTIONS section is added.

41874 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 41875 • The type of argument *act* is changed from **struct sigaction\*** to **const struct sigaction\***.
- 41876 • A statement is added to the DESCRIPTION indicating that the consequence of attempting to  
41877 set SIG\_DFL for a signal that cannot be caught or ignored is unspecified. The [EINVAL] error,  
41878 describing one possible reaction to this condition, is added to the ERRORS section.

#### 41879 Issue 4, Version 2

41880 The following changes are incorporated for X/OPEN UNIX conformance:

- 41881 • The DESCRIPTION describes *sa\_sigaction*, the member of the **sigaction** structure that is the  
41882 signal-catching function.
- 41883 • The DESCRIPTION describes the SA\_ONSTACK, SA\_RESETHAND, SA\_RESTART,  
41884 SA\_SIGINFO, SA\_NOCLDWAIT, and SA\_NODEFER settings of *sa\_flags*. The text describes  
41885 the implications of the use of SA\_SIGINFO for the number of arguments passed to the  
41886 signal-catching function. The text also describes the effects of the SA\_NODEFER and  
41887 SA\_RESETHAND flags on the delivery of a signal and on the permanence of an installed  
41888 action.
- 41889 • The DESCRIPTION specifies the effect if the action for the SIGCHLD signal is set to  
41890 SIG\_IGN.
- 41891 • In the DESCRIPTION, additional text describes the effect if the action is a pointer to a  
41892 function. A new bullet covers the case where SA\_SIGINFO is set. SIGBUS is given as an  
41893 additional signal for which the behavior of a process is undefined following a normal return  
41894 from the signal-catching function.

- 41895           • The APPLICATION USAGE section is updated to describe use of an alternate signal stack;  
41896           resumption of the process receiving the signal; coding for compatibility with POSIX.4-1993;  
41897           and implementation of signal-handling functions in BSD.
- 41898 **Issue 5**
- 41899           The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and POSIX  
41900           Threads Extension.
- 41901           In the DESCRIPTION, the second argument to *func* when SA\_SIGINFO is set is no longer  
41902           permitted to be NULL, and the description of permitted **siginfo\_t** contents is expanded by  
41903           reference to <**signal.h**>.
- 41904           Because the X/OPEN UNIX Extension functionality is now folded into the BASE, the  
41905           [ENOTSUP] error is deleted.
- 41906 **Issue 6**
- 41907           The Open Group corrigenda item U028/7 has been applied. In the paragraph entitled “Signal  
41908           Effects on Other Functions”, a reference to *sigpending()* is added.
- 41909           In the DESCRIPTION, the text “Signal Generation and Delivery” is moved to a separate section  
41910           of this volume of IEEE Std. 1003.1-200x.
- 41911           Text describing functionality from the Realtime Signals option is marked.
- 41912           The following changes are made for alignment with the ISO POSIX-1: 1996 standard:
- 41913           • The [ENOTSUP] error condition is added.
- 41914           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 41915           The **restrict** keyword is added to the *sigaction()* prototype for alignment with the  
41916           ISO/IEC 9899:1999 standard.
- 41917           References to the *wait3()* function are removed.

41918 **NAME**

41919 sigaddset — add a signal to a signal set

41920 **SYNOPSIS**

41921 #include <signal.h>

41922 int sigaddset(sigset\_t \*set, int signo);

41923 **DESCRIPTION**

41924 The *sigaddset()* function adds the individual signal specified by the *signo* to the signal set pointed  
41925 to by *set*.

41926 Applications shall call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
41927 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
41928 nonetheless supplied as an argument to any of *sigaction()*, *sigaddset()*, *sigdelset()*, *sigismember()*,  
41929 *sigpending()*, or *sigprocmask()*, the results are undefined.

41930 **RETURN VALUE**

41931 Upon successful completion, *sigaddset()* shall return 0; otherwise, it shall return -1 and set *errno*  
41932 to indicate the error.

41933 **ERRORS**

41934 The *sigaddset()* function may fail if:

41935 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number. |

41936 **EXAMPLES**

41937 None.

41938 **APPLICATION USAGE**

41939 None.

41940 **RATIONALE**

41941 None.

41942 **FUTURE DIRECTIONS**

41943 None.

41944 **SEE ALSO**

41945 Section 2.4 (on page 528), *sigaction()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*,  
41946 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |  
41947 <signal.h>

41948 **CHANGE HISTORY**

41949 First released in Issue 3.

41950 Entry included for alignment with the POSIX.1-1988 standard.

41951 **Issue 4**

41952 The word “will” is replaced by the word “may” in the ERRORS section.

41953 **Issue 5**

41954 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
41955 previous issues.

41956 **Issue 6**

41957 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

## 41958 NAME

41959 sigaltstack — set and get signal alternate stack context

## 41960 SYNOPSIS

41961 XSI #include &lt;signal.h&gt;

41962 int sigaltstack(const stack\_t \*restrict ss, stack\_t \*restrict oss);

41963

## 41964 DESCRIPTION

41965 The *sigaltstack()* function allows a process to define and examine the state of an alternate stack  
 41966 for signal handlers. Signals that have been explicitly declared to execute on the alternate stack  
 41967 shall be delivered on the alternate stack.

41968 If *ss* is not a null pointer, it points to a **stack\_t** structure that specifies the alternate signal stack  
 41969 that shall take effect upon return from *sigaltstack()*. The *ss\_flags* member specifies the new stack  
 41970 state. If it is set to *SS\_DISABLE*, the stack is disabled and *ss\_sp* and *ss\_size* are ignored.  
 41971 Otherwise, the stack shall be enabled, and the *ss\_sp* and *ss\_size* members specify the new address  
 41972 and size of the stack.

41973 The range of addresses starting at *ss\_sp* up to but not including *ss\_sp+ss\_size*, is available to the  
 41974 implementation for use as the stack. This function makes no assumptions regarding which end  
 41975 is the stack base and in which direction the stack grows as items are pushed.

41976 If *oss* is not a null pointer, on successful completion it shall point to a **stack\_t** structure that  
 41977 specifies the alternate signal stack that was in effect prior to the call to *sigaltstack()*. The *ss\_sp*  
 41978 and *ss\_size* members specify the address and size of that stack. The *ss\_flags* member specifies the  
 41979 stack's state, and may contain one of the following values:

41980 **SS\_ONSTACK** The process is currently executing on the alternate signal stack. Attempts to  
 41981 modify the alternate signal stack while the process is executing on it fail. This  
 41982 flag shall not be modified by processes.

41983 **SS\_DISABLE** The alternate signal stack is currently disabled.

41984 The value *SIGSTKSZ* is a system default specifying the number of bytes that would be used to  
 41985 cover the usual case when manually allocating an alternate stack area. The value *MINSIGSTKSZ*  
 41986 is defined to be the minimum stack size for a signal handler. In computing an alternate stack  
 41987 size, a program should add that amount to its stack requirements to allow for the system  
 41988 implementation overhead. The constants *SS\_ONSTACK*, *SS\_DISABLE*, *SIGSTKSZ*, and  
 41989 *MINSIGSTKSZ* are defined in <signal.h>.

41990 After a successful call to one of the *exec* functions, there are no alternate signal stacks in the new  
 41991 process image.

41992 In some implementations, a signal (whether or not indicated to execute on the alternate stack)  
 41993 shall always execute on the alternate stack if it is delivered while another signal is being caught  
 41994 using the alternate stack.

41995 Use of this function by library threads that are not bound to kernel-scheduled entities results in  
 41996 undefined behavior.

## 41997 RETURN VALUE

41998 Upon successful completion, *sigaltstack()* shall return 0; otherwise, it shall return -1 and set *errno*  
 41999 to indicate the error.



**42000 ERRORS**

- 42001 The *sigaltstack()* function shall fail if:
- 42002 [EINVAL] The *ss* argument is not a null pointer, and the *ss\_flags* member pointed to by *ss* |  
42003 contains flags other than *SS\_DISABLE*.
- 42004 [ENOMEM] The size of the alternate stack area is less than *MINSIGSTKSZ*. |
- 42005 [EPERM] An attempt was made to modify an active stack. |

**42006 EXAMPLES****42007 Allocating Memory for an Alternate Stack**

42008 The following example illustrates a method for allocating memory for an alternate stack.

```
42009 #include <signal.h>
42010 ...
42011 if ((sigstk.ss_sp = malloc(SIGSTKSZ)) == NULL)
42012 /* Error return. */
42013 sigstk.ss_size = SIGSTKSZ;
42014 sigstk.ss_flags = 0;
42015 if (sigaltstack(&sigstk, (stack_t *)0) < 0)
42016 perror("sigaltstack");
```

**42017 APPLICATION USAGE**

- 42018 On some implementations, stack space is automatically extended as needed. On those |  
42019 implementations, automatic extension is typically not available for an alternate stack. If the stack |  
42020 overflows, the behavior is undefined.

**42021 RATIONALE**

42022 None.

**42023 FUTURE DIRECTIONS**

42024 None.

**42025 SEE ALSO**

42026 Section 2.4 (on page 528), *sigaction()*, *sigsetjmp()*, the Base Definitions volume of |  
42027 IEEE Std. 1003.1-200x, <signal.h>

**42028 CHANGE HISTORY**

42029 First released in Issue 4, Version 2.

**42030 Issue 5**

42031 Moved from X/OPEN UNIX extension to BASE.

42032 The last sentence of the DESCRIPTION was included as an APPLICATION USAGE note in |  
42033 previous issues.

**42034 Issue 6**

42035 The DESCRIPTION is updated to avoid use of the term “must” for application requirements. |

42036 The **restrict** keyword is added to the *sigaltstack()* prototype for alignment with the |  
42037 ISO/IEC 9899:1999 standard.

42038 **NAME**

42039 sigdelset — delete a signal from a signal set

42040 **SYNOPSIS**

42041 #include <signal.h>

42042 int sigdelset(sigset\_t \*set, int signo);

42043 **DESCRIPTION**

42044 The *sigdelset()* function deletes the individual signal specified by *signo* from the signal set pointed to by *set*.

42046 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of *sigaction()*, *sigaddset()*, *sigdelset()*, *sigismember()*, *sigpending()*, or *sigprocmask()*, the results are undefined.

42050 **RETURN VALUE**

42051 Upon successful completion, *sigdelset()* shall return 0; otherwise, it shall return -1 and set *errno* to indicate the error.

42053 **ERRORS**

42054 The *sigdelset()* function may fail if:

42055 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal number.

42057 **EXAMPLES**

42058 None.

42059 **APPLICATION USAGE**

42060 None.

42061 **RATIONALE**

42062 None.

42063 **FUTURE DIRECTIONS**

42064 None.

42065 **SEE ALSO**

42066 Section 2.4 (on page 528), *sigaction()*, *sigaddset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <signal.h>

42069 **CHANGE HISTORY**

42070 First released in Issue 3.

42071 Entry included for alignment with the POSIX.1-1988 standard.

42072 **Issue 4**

42073 The word “will” is replaced by the word “may” in the ERRORS section.

42074 **Issue 5**

42075 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in previous issues.

42077 **NAME**

42078 sigemptyset — initialize and empty a signal set

42079 **SYNOPSIS**

42080 #include <signal.h>

42081 int sigemptyset(sigset\_t \*set);

42082 **DESCRIPTION**

42083 The *sigemptyset()* function initializes the signal set pointed to by *set*, such that all signals defined  
42084 in this volume of IEEE Std. 1003.1-200x are excluded.

42085 **RETURN VALUE**

42086 Upon successful completion, *sigemptyset()* shall return 0; otherwise, it shall return -1 and set  
42087 *errno* to indicate the error.

42088 **ERRORS**

42089 No errors are defined.

42090 **EXAMPLES**

42091 None.

42092 **APPLICATION USAGE**

42093 None.

42094 **RATIONALE**

42095 The implementation of the *sigemptyset()* (or *sigfillset()*) function could quite trivially clear (or  
42096 set) all the bits in the signal set. Alternatively, it would be reasonable to initialize part of the  
42097 structure, such as a version field, to permit binary-compatibility between releases where the size  
42098 of the set varies. For such reasons, either *sigemptyset()* or *sigfillset()* must be called prior to any  
42099 other use of the signal set, even if such use is read-only (for example, as an argument to  
42100 *sigpending()*). This function is not intended for dynamic allocation.

42101 The *sigfillset()* and *sigemptyset()* functions require that the resulting signal set include (or  
42102 exclude) all the signals defined in this volume of IEEE Std. 1003.1-200x. Although it is outside  
42103 the scope of this volume of IEEE Std. 1003.1-200x to place this requirement on signals that are  
42104 implemented as extensions, it is recommended that implementation-defined signals also be  
42105 affected by these functions. However, there may be a good reason for a particular signal not to  
42106 be affected. For example, blocking or ignoring an implementation-defined signal may have  
42107 undesirable side effects, whereas the default action for that signal is harmless. In such a case, it  
42108 would be preferable for such a signal to be excluded from the signal set returned by *sigfillset()*.

42109 In early proposals there was no distinction between invalid and unsupported signals (the names  
42110 of optional signals that were not supported by an implementation were not defined by that  
42111 implementation). The [EINVAL] error was thus specified as a required error for invalid signals.  
42112 With that distinction, it is not necessary to require implementations of these functions to  
42113 determine whether an optional signal is actually supported, as that could have a significant  
42114 performance impact for little value. The error could have been required for invalid signals and  
42115 optional for unsupported signals, but this seemed unnecessarily complex. Thus, the error is  
42116 optional in both cases.

42117 **FUTURE DIRECTIONS**

42118 None.

42119 **SEE ALSO**

42120 Section 2.4 (on page 528), *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*, *sigismember()*,  
42121 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
42122 <signal.h>

42123 **CHANGE HISTORY**

42124 First released in Issue 3.

42125 Entry included for alignment with the POSIX.1-1988 standard.

42126 **NAME**

42127 sigfillset — initialize and fill a signal set

42128 **SYNOPSIS**

42129 #include <signal.h>

42130 int sigfillset(sigset\_t \*set);

42131 **DESCRIPTION**

42132 The *sigfillset()* function initializes the signal set pointed to by *set*, such that all signals defined in  
42133 this volume of IEEE Std. 1003.1-200x are included.

42134 **RETURN VALUE**

42135 Upon successful completion, *sigfillset()* shall return 0; otherwise, it shall return -1 and set *errno*  
42136 to indicate the error.

42137 **ERRORS**

42138 No errors are defined.

42139 **EXAMPLES**

42140 None.

42141 **APPLICATION USAGE**

42142 None.

42143 **RATIONALE**

42144 Refer to *sigemptyset()* (on page 1863).

42145 **FUTURE DIRECTIONS**

42146 None.

42147 **SEE ALSO**

42148 Section 2.4 (on page 528), *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigismember()*,  
42149 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
42150 <signal.h>

42151 **CHANGE HISTORY**

42152 First released in Issue 3.

42153 Entry included for alignment with the POSIX.1-1988 standard.

42154 **NAME**

42155           sighold, sigignore — add a signal to the signal mask or set a signal disposition to be ignored

42156 **SYNOPSIS**

```
42157 XSI #include <signal.h>
```

```
42158 int sighold(int sig);
```

```
42159 int sigignore(int sig);
```

42160

42161 **DESCRIPTION**

42162           Refer to *signal()*.

42163 **NAME**

42164 siginterrupt — allow signals to interrupt functions

42165 **SYNOPSIS**

42166 XSI #include &lt;signal.h&gt;

42167 int siginterrupt(int *sig*, int *flag*);

42168

42169 **DESCRIPTION**42170 The *siginterrupt()* function is used to change the restart behavior when a function is interrupted  
42171 by the specified signal. The function *siginterrupt(sig, flag)* has an effect as if implemented as:

```

42172 siginterrupt(int sig, int flag) {
42173 int ret;
42174 struct sigaction act;
42175
42176 (void) sigaction(sig, NULL, &act);
42177 if (flag)
42178 act.sa_flags &= ~SA_RESTART;
42179 else
42180 act.sa_flags |= SA_RESTART;
42181 ret = sigaction(sig, &act, NULL);
42182 return ret;
42183 }

```

42183 **RETURN VALUE**42184 Upon successful completion, *siginterrupt()* shall return 0; otherwise, -1 shall be returned and  
42185 *errno* set to indicate the error.42186 **ERRORS**42187 The *siginterrupt()* function shall fail if:42188 [EINVAL] The *sig* argument is not a valid signal number.42189 **EXAMPLES**

42190 None.

42191 **APPLICATION USAGE**42192 The *siginterrupt()* function supports programs written to historical system interfaces. A portable  
42193 application, when being written or rewritten, should use *sigaction()* with the SA\_RESTART flag  
42194 instead of *siginterrupt()*.42195 **RATIONALE**

42196 None.

42197 **FUTURE DIRECTIONS**

42198 None.

42199 **SEE ALSO**42200 Section 2.4 (on page 528), *sigaction()*, the Base Definitions volume of IEEE Std. 1003.1-200x,

42201 &lt;signal.h&gt;

42202 **CHANGE HISTORY**

42203 First released in Issue 4, Version 2.

42204 **Issue 5**

42205 Moved from X/OPEN UNIX extension to BASE.



42206 **NAME**

42207 sigismember — test for a signal in a signal set

42208 **SYNOPSIS**

42209 #include &lt;signal.h&gt;

42210 int sigismember(const sigset\_t \*set, int signo);

42211 **DESCRIPTION**42212 The *sigismember()* function tests whether the signal specified by *signo* is a member of the set  
42213 pointed to by *set*.42214 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
42215 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
42216 nonetheless supplied as an argument to any of *sigaction()*, *sigaddset()*, *sigdelset()*, *sigismember()*,  
42217 *sigpending()*, or *sigprocmask()*, the results are undefined.42218 **RETURN VALUE**42219 Upon successful completion, *sigismember()* shall return 1 if the specified signal is a member of  
42220 the specified set, or 0 if it is not. Otherwise, it shall return -1 and set *errno* to indicate the error.42221 **ERRORS**42222 The *sigismember()* function may fail if:42223 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal  
42224 number.42225 **EXAMPLES**

42226 None.

42227 **APPLICATION USAGE**

42228 None.

42229 **RATIONALE**

42230 None.

42231 **FUTURE DIRECTIONS**

42232 None.

42233 **SEE ALSO**42234 Section 2.4 (on page 528), *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*, *sigemptyset()*,  
42235 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
42236 <signal.h>42237 **CHANGE HISTORY**

42238 First released in Issue 3.

42239 Entry included for alignment with the POSIX.1-1988 standard.

42240 **Issue 4**

42241 The following changes are incorporated for alignment with the ISO C standard:

- 42242
- The type of the argument *set* is changed from **sigset\_t\*** to type **const sigset\_t\***.
  - The word “will” is replaced by the word “may” in the ERRORS section.
- 42243

42244 **Issue 5**42245 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
42246 previous issues.

42247 **NAME**

42248 siglongjmp — non-local goto with signal handling

42249 **SYNOPSIS**

42250 #include &lt;setjmp.h&gt;

42251 void siglongjmp(sigjmp\_buf env, int val);

42252 **DESCRIPTION**

42253 The *siglongjmp()* function restores the environment saved by the most recent invocation of  
42254 *sigsetjmp()* in the same thread, with the corresponding **sigjmp\_buf** argument. If there is no such  
42255 invocation, or if the function containing the invocation of *sigsetjmp()* has terminated execution in  
42256 the interim, the behavior is undefined.

42257 All accessible objects have values as of the time *siglongjmp()* was called, except that the values of  
42258 objects of automatic storage duration which are local to the function containing the invocation of  
42259 the corresponding *sigsetjmp()* which do not have volatile-qualified type and which are changed  
42260 between the *sigsetjmp()* invocation and *siglongjmp()* call are indeterminate.

42261 As it bypasses the usual function call and return mechanisms, *siglongjmp()* shall execute  
42262 correctly in contexts of interrupts, signals, and any of their associated functions. However, if  
42263 *siglongjmp()* is invoked from a nested signal handler (that is, from a function invoked as a result  
42264 of a signal raised during the handling of another signal), the behavior is undefined.

42265 The *siglongjmp()* function shall restore the saved signal mask if and only if the *env* argument was  
42266 initialized by a call to *sigsetjmp()* with a non-zero *savemask* argument.

42267 The effect of a call to *siglongjmp()* where initialization of the **jmp\_buf** structure was not  
42268 performed in the calling thread is undefined.

42269 **RETURN VALUE**

42270 After *siglongjmp()* is completed, program execution shall continue as if the corresponding  
42271 invocation of *sigsetjmp()* had just returned the value specified by *val*. The *siglongjmp()* function  
42272 shall not cause *sigsetjmp()* to return 0; if *val* is 0, *sigsetjmp()* shall return the value 1.

42273 **ERRORS**

42274 No errors are defined.

42275 **EXAMPLES**

42276 None.

42277 **APPLICATION USAGE**

42278 The distinction between *setjmp()* or *longjmp()* and *sigsetjmp()* or *siglongjmp()* is only significant  
42279 for programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

42280 **RATIONALE**

42281 None.

42282 **FUTURE DIRECTIONS**

42283 None.

42284 **SEE ALSO**

42285 *longjmp()*, *setjmp()*, *sigprocmask()*, *sigsetjmp()*, *sigsuspend()*, the Base Definitions volume of  
42286 IEEE Std. 1003.1-200x, <**setjmp.h**>

42287 **CHANGE HISTORY**

42288 First released in Issue 3.

42289 Entry included for alignment with the ISO POSIX-1 standard.

42290 **Issue 4**

42291 The APPLICATION USAGE section is amended.

42292 An ERRORS section is added.

42293 **Issue 5**

42294 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

## 42295 NAME

42296 sighold, sigignore, signal, sigpause, sigrelse, sigset — signal management

## 42297 SYNOPSIS

42298 #include &lt;signal.h&gt;

42299 void (\*signal(int *sig*, void (\**func*)(int)))(int);42300 XSI int sighold(int *sig*);42301 int sigignore(int *sig*);42302 int sigpause(int *sig*);42303 int sigrelse(int *sig*);42304 void (\*sigset(int *sig*, void (\**disp*)(int)))(int);

42305

## 42306 DESCRIPTION

42307 CX For *signal()*: The functionality described on this reference page is aligned with the ISO C  
 42308 standard. Any conflict between the requirements described here and the ISO C standard is  
 42309 unintentional. This volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

42310 CX Use of any of these functions is unspecified in a multi-threaded process.

42311 The *signal()* function chooses one of three ways in which receipt of the signal number *sig* is to be  
 42312 subsequently handled. If the value of *func* is SIG\_DFL, default handling for that signal shall  
 42313 occur. If the value of *func* is SIG\_IGN, the signal shall be ignored. Otherwise, the application  
 42314 shall ensure that *func* points to a function to be called when that signal occurs. An invocation of  
 42315 such a function because of a signal, or (recursively) of any further functions called by that  
 42316 invocation (other than functions in the standard library), is called a “signal handler”.

42317 When a signal occurs, if *func* points to a function, first the equivalent of a:

42318 `signal(sig, SIG_DFL);`

42319 is executed or an implementation-defined blocking of the signal is performed. (If the value of *sig*  
 42320 is SIGILL, whether the reset to SIG\_DFL occurs is implementation-defined.) Next the equivalent  
 42321 of:

42322 `(*func)(sig);`

42323 is executed. The *func* function may terminate by executing a **return** statement or by calling  
 42324 *abort()*, *exit()*, or *longjmp()*. If *func* executes a **return** statement and the value of *sig* was SIGFPE  
 42325 or any other implementation-defined value corresponding to a computational exception, the  
 42326 behavior is undefined. Otherwise, the program shall resume execution at the point it was  
 42327 interrupted. If the signal occurs as the result of calling the *abort()* or *raise()* function, the signal  
 42328 handler shall not call the *raise()* function.

42329 If the signal occurs other than as the result of calling *abort()*, *kill()*, or *raise()*, the behavior is  
 42330 undefined if the signal handler refers to any object with static storage duration other than by  
 42331 assigning a value to an object declared as volatile **sig\_atomic\_t**, or if the signal handler calls any  
 42332 function in the standard library other than one of the functions listed in Section 2.4 (on page 528)  
 42333 or refers to any object with static storage duration other than by assigning a value to a static  
 42334 storage duration variable of type volatile **sig\_atomic\_t**. Furthermore, if such a call fails, the  
 42335 value of *errno* is indeterminate.

42336 At program start-up, the equivalent of:

42337 `signal(sig, SIG_IGN);`

42338 is executed for some signals, and the equivalent of:

42339 `signal(sig, SIG_DFL);`

42340 is executed for all other signals (see *exec*).

42341 XSI The *sighold()*, *sigignore()*, *sigpause()*, *sigrelse()*, and *sigset()* functions provide simplified signal management.

42343 The *sigset()* function is used to modify signal dispositions. The *sig* argument specifies the signal, which may be any signal except SIGKILL and SIGSTOP. The *disp* argument specifies the signal's disposition, which may be SIG\_DFL, SIG\_IGN, or the address of a signal handler. If *sigset()* is used, and *disp* is the address of a signal handler, the system shall add *sig* to the calling process' signal mask before executing the signal handler; when the signal handler returns, the system shall restore the calling process' signal mask to its state prior the delivery of the signal. In addition, if *sigset()* is used, and *disp* is equal to SIG\_HOLD, *sig* shall be added to the calling process' signal mask and *sig*'s disposition shall remain unchanged. If *sigset()* is used, and *disp* is not equal to SIG\_HOLD, *sig* shall be removed from the calling process' signal mask.

42352 The *sighold()* function adds *sig* to the calling process' signal mask.

42353 The *sigrelse()* function removes *sig* from the calling process' signal mask.

42354 The *sigignore()* function sets the disposition of *sig* to SIG\_IGN.

42355 The *sigpause()* function removes *sig* from the calling process' signal mask and suspends the calling process until a signal is received. The *sigpause()* function restores the process' signal mask to its original state before returning.

42358 If the action for the SIGCHLD signal is set to SIG\_IGN, child processes of the calling processes shall not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited-for children that were transformed into zombie processes, it shall block until all of its children terminate, and *wait()*, *waitid()*, and *waitpid()* shall fail and set *errno* to [ECHILD].

#### 42363 RETURN VALUE

42364 If the request can be honored, *signal()* shall return the value of *func* for the most recent call to *signal()* for the specified signal *sig*. Otherwise, SIG\_ERR shall be returned and a positive value shall be stored in *errno*.

42367 XSI Upon successful completion, *sigset()* shall return SIG\_HOLD if the signal had been blocked and the signal's previous disposition if it had not been blocked. Otherwise, SIG\_ERR shall be returned and *errno* set to indicate the error.

42370 The *sigpause()* function shall suspend execution of the thread until a signal is received, whereupon it shall return -1 and set *errno* to [EINTR].

42372 For all other functions, upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

#### 42374 ERRORS

42375 The *signal()* function shall fail if:

42376 CX [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a signal that cannot be caught or ignore a signal that cannot be ignored.

42378 The *signal()* function may fail if:

42379 CX [EINVAL] An attempt was made to set the action to SIG\_DFL for a signal that cannot be caught or ignored (or both).

42381 The *sighold()*, *sigignore()*, *sigpause()*, *sigrelse()*, and *sigset()* functions shall fail if:

- 42382 XSI [EINVAL] The *sig* argument is an illegal signal number.
- 42383 The *sigset()* and *sigignore()* functions shall fail if:
- 42384 XSI [EINVAL] An attempt is made to catch a signal that cannot be caught, or to ignore a  
42385 signal that cannot be ignored.
- 42386 **EXAMPLES**
- 42387 None.
- 42388 **APPLICATION USAGE**
- 42389 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling  
42390 signals; new applications should use *sigaction()* rather than *signal()*.
- 42391 The *sighold()* function, in conjunction with *sigrelse()* or *sigpause()*, may be used to establish  
42392 critical regions of code that require the delivery of a signal to be temporarily deferred.
- 42393 The *sigsuspend()* function should be used in preference to *sigpause()* for broader portability.
- 42394 **RATIONALE**
- 42395 None.
- 42396 **FUTURE DIRECTIONS**
- 42397 None.
- 42398 **SEE ALSO**
- 42399 Section 2.4 (on page 528), *exec*, *pause()*, *sigaction()*, *sigsuspend()*, *waitid()*, the Base Definitions  
42400 volume of IEEE Std. 1003.1-200x, <signal.h>
- 42401 **CHANGE HISTORY**
- 42402 First released in Issue 1. Derived from Issue 1 of the SVID.
- 42403 **Issue 4**
- 42404 The APPLICATION USAGE section is added.
- 42405 The following changes are incorporated for alignment with the ISO C standard:
- 42406 • The function is no longer marked as an extension.
- 42407 • The argument **int** is added to the definition of *func* in the SYNOPSIS section.
- 42408 • In Issue 3, this function cross-referred to *sigaction()*. This issue provides a complete  
42409 description of the function as defined in ISO C standard.
- 42410 **Issue 4, Version 2**
- 42411 The following changes are incorporated for X/OPEN UNIX conformance:
- 42412 • The *sighold()*, *sigignore()*, *sigpause()*, *sigrelse()*, and *sigset()* functions are added to the  
42413 SYNOPSIS.
- 42414 • The DESCRIPTION is updated to describe semantics of the above functions.
- 42415 • Additional text is added to the RETURN VALUE section to describe possible returns from  
42416 the *sigset()* function specifically, and all of the above functions in general.
- 42417 • The ERRORS section is restructured to describe possible error returns from each of the above  
42418 functions individually.
- 42419 • The APPLICATION USAGE section is updated to describe certain programming  
42420 considerations associated with the X/OPEN UNIX functions.

42421 **Issue 5**

42422 Moved from X/OPEN UNIX extension to BASE.

42423 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the process' signal mask to its original state before returning.

42425 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to [EINTR].

42428 **Issue 6**

42429 Extensions beyond the ISO C standard are now marked.

42430 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

42431 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

42432 References to the *wait3()* function are removed.

42433 **NAME**

42434 signbit — test sign

42435 **SYNOPSIS**

42436 #include &lt;math.h&gt;

42437 int signbit(real-floating x);

42438 **DESCRIPTION**

42439 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
42440 conflict between the requirements described here and the ISO C standard is unintentional. This  
42441 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

42442 The *signbit()* macro shall determine whether the sign of its argument value is negative.42443 **RETURN VALUE**

42444 The *signbit()* macro shall return a non-zero value if and only if the sign of its argument value is  
42445 negative.

42446 **ERRORS**

42447 No errors are defined.

42448 **EXAMPLES**

42449 None.

42450 **APPLICATION USAGE**

42451 None.

42452 **RATIONALE**

42453 None.

42454 **FUTURE DIRECTIONS**

42455 None.

42456 **SEE ALSO**

42457 *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *isnormal()*, the Base Definitions volume of  
42458 IEEE Std. 1003.1-200x, <math.h>

42459 **CHANGE HISTORY**

42460 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.



42461 **NAME**

42462 sigpause — remove a signal from the signal mask and suspend the thread

42463 **SYNOPSIS**

42464 XSI #include <signal.h>

42465 int sigpause(int *sig*);

42466

42467 **DESCRIPTION**

42468 Refer to *signal()*.

42469 **NAME**

42470 sigpending — examine pending signals

42471 **SYNOPSIS**

42472 #include <signal.h>

42473 int sigpending(sigset\_t \*set);

42474 **DESCRIPTION**

42475 The *sigpending()* function stores, in the location referenced by the *set* argument, the set of signals  
42476 that are blocked from delivery to the calling thread and that are pending on the process or the  
42477 calling thread.

42478 **RETURN VALUE**

42479 Upon successful completion, *sigpending()* shall return 0; otherwise, -1 shall be returned and  
42480 *errno* set to indicate the error.

42481 **ERRORS**

42482 No errors are defined.

42483 **EXAMPLES**

42484 None.

42485 **APPLICATION USAGE**

42486 None.

42487 **RATIONALE**

42488 None.

42489 **FUTURE DIRECTIONS**

42490 None.

42491 **SEE ALSO**

42492 *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *sigprocmask()*, the Base Definitions  
42493 volume of IEEE Std. 1003.1-200x, <signal.h>

42494 **CHANGE HISTORY**

42495 First released in Issue 3.

42496 **Issue 5**

42497 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42498 **NAME**

42499 sigprocmask — examine and change blocked signals

42500 **SYNOPSIS**

42501 #include &lt;signal.h&gt;

42502 int sigprocmask(int *how*, const sigset\_t \*restrict *set*,  
42503 sigset\_t \*restrict *oset*);42504 **DESCRIPTION**42505 Refer to *pthread\_sigmask()*.

## 42506 NAME

42507 sigqueue — queue a signal to a process (**REALTIME**)

## 42508 SYNOPSIS

42509 RTS `#include <signal.h>`42510 `int sigqueue(pid_t pid, int signo, const union sigval value);`

42511

## 42512 DESCRIPTION

42513 The *sigqueue()* function shall cause the signal specified by *signo* to be sent with the value  
 42514 specified by *value* to the process specified by *pid*. If *signo* is zero (the null signal), error checking  
 42515 is performed but no signal is actually sent. The null signal can be used to check the validity of  
 42516 *pid*.

42517 The conditions required for a process to have permission to queue a signal to another process  
 42518 are the same as for the *kill()* function.

42519 The *sigqueue()* function returns immediately. If SA\_SIGINFO is set for *signo* and if the resources  
 42520 were available to queue the signal, the signal is queued and sent to the receiving process. If  
 42521 SA\_SIGINFO is not set for *signo*, then *signo* is sent at least once to the receiving process; it is  
 42522 unspecified whether *value* shall be sent to the receiving process as a result of this call.

42523 If the value of *pid* causes *signo* to be generated for the sending process, and if *signo* is not blocked  
 42524 for the calling thread and if no other thread has *signo* unblocked or is waiting in a *sigwait()*  
 42525 function for *signo*, either *signo* or at least the pending, unblocked signal shall be delivered to the  
 42526 calling thread before the *sigqueue()* function returns. Should any multiple pending signals in the  
 42527 range SIGRTMIN to SIGRTMAX be selected for delivery, it shall be the lowest numbered one.  
 42528 The selection order between realtime and non-realtime signals, or between multiple pending  
 42529 non-realtime signals, is unspecified.

## 42530 RETURN VALUE

42531 Upon successful completion, the specified signal shall have been queued, and the *sigqueue()*  
 42532 function shall return a value of zero. Otherwise, the function shall return a value of  $-1$  and set  
 42533 *errno* to indicate the error.

## 42534 ERRORS

42535 The *sigqueue()* function shall fail if:

|       |          |                                                                              |  |
|-------|----------|------------------------------------------------------------------------------|--|
| 42536 | [EAGAIN] | No resources available to queue the signal. The process has already queued   |  |
| 42537 |          | SIGQUEUE_MAX signals that are still pending at the receiver(s), or a system- |  |
| 42538 |          | wide resource limit has been exceeded.                                       |  |

|       |          |                                                                                    |  |
|-------|----------|------------------------------------------------------------------------------------|--|
| 42539 | [EINVAL] | The value of the <i>signo</i> argument is an invalid or unsupported signal number. |  |
|-------|----------|------------------------------------------------------------------------------------|--|

|       |         |                                                                               |  |
|-------|---------|-------------------------------------------------------------------------------|--|
| 42540 | [EPERM] | The process does not have the appropriate privilege to send the signal to the |  |
| 42541 |         | receiving process.                                                            |  |

|       |         |                                        |  |
|-------|---------|----------------------------------------|--|
| 42542 | [ESRCH] | The process <i>pid</i> does not exist. |  |
|-------|---------|----------------------------------------|--|

42543 **EXAMPLES**

42544 None.

42545 **APPLICATION USAGE**

42546 None.

42547 **RATIONALE**

42548 The *sigqueue()* function allows an application to queue a realtime signal to itself or to another  
 42549 process, specifying the application-defined value. This is common practice in realtime  
 42550 applications on existing realtime systems. It was felt that specifying another function in the  
 42551 *sig...* name space already carved out for signals was preferable to extending the function to  
 42552 *kill()*.

42553 Such a function became necessary when the put/get event function of the message queues was  
 42554 removed. It should be noted that the *sigqueue()* function implies reduced performance in a  
 42555 security-conscious implementation as the access permissions between the sender and receiver  
 42556 have to be checked on each send when the *pid* is resolved into a target process. Such access  
 42557 checks were necessary only at message queue open in the previous function.

42558 The standard developers required that *sigqueue()* have the same semantics with respect to the  
 42559 null signal as *kill()*, and that the same permission checking be used. But because of the difficulty  
 42560 of implementing the “broadcast” semantic of *kill()* (for example, to process groups) and the  
 42561 interaction with resource allocation, this semantic was not adopted. The *sigqueue()* function  
 42562 queues a signal to a single process specified by the *pid* argument.

42563 The *sigqueue()* function can fail if the system has insufficient resources to queue the signal. An  
 42564 explicit limit on the number of queued signals that a process could send was introduced. While  
 42565 the limit is “per-sender”, this volume of IEEE Std. 1003.1-200x does not specify that the  
 42566 resources be part of the state of the sender. This would require either that the sender be  
 42567 maintained after exit until all signals that it had sent to other processes were handled or that all  
 42568 such signals that had not yet been acted upon be removed from the queue(s) of the receivers.  
 42569 This volume of IEEE Std. 1003.1-200x does not preclude this behavior, but an implementation  
 42570 that allocated queuing resources from a system-wide pool (with per-sender limits) and that  
 42571 leaves queued signals pending after the sender exits is also permitted.

42572 **FUTURE DIRECTIONS**

42573 None.

42574 **SEE ALSO**

42575 Section 2.8.1 (on page 543), the Base Definitions volume of IEEE Std. 1003.1-200x, &lt;signal.h&gt;

42576 **CHANGE HISTORY**

42577 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
 42578 POSIX Threads Extension.

42579 **Issue 6**42580 The *sigqueue()* function is marked as part of the Realtime Signals Extension option.

42581 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 42582 implementation does not support the Realtime Signals Extension option.

42583 **NAME**

42584 sigrelse, sigset — remove a signal from signal mask or modify signal disposition

42585 **SYNOPSIS**

42586 XSI #include <signal.h>

42587 int sigrelse(int *sig*);

42588 void (\*sigset(int *sig*, void (\**disp*)(int)))(int);

42589

42590 **DESCRIPTION**

42591 Refer to *signal()*.

42592 **NAME**

42593 sigsetjmp — set jump point for a non-local goto

42594 **SYNOPSIS**

42595 #include &lt;setjmp.h&gt;

42596 int sigsetjmp(sigjmp\_buf env, int savemask);

42597 **DESCRIPTION**

42598 A call to *sigsetjmp()* saves the calling environment in its *env* argument for later use by  
 42599 *siglongjmp()*. It is unspecified whether *sigsetjmp()* is a macro or a function. If a macro definition  
 42600 is suppressed in order to access an actual function, or a program defines an external identifier  
 42601 with the name *sigsetjmp*, the behavior is undefined.

42602 If the value of the *savemask* argument is not 0, *sigsetjmp()* shall also save the current signal mask  
 42603 of the calling thread as part of the calling environment.

42604 All accessible objects have values as of the time *siglongjmp()* was called, except that the values of  
 42605 objects of automatic storage duration which are local to the function containing the invocation of  
 42606 the corresponding *sigsetjmp()* which do not have volatile-qualified type and which are changed  
 42607 between the *sigsetjmp()* invocation and *siglongjmp()* call are indeterminate.

42608 The application shall ensure that an invocation of *sigsetjmp()* appears in one of the following  
 42609 contexts only:

- 42610 • The entire controlling expression of a selection or iteration statement
- 42611 • One operand of a relational or equality operator with the other operand an integral constant  
 42612 expression, with the resulting expression being the entire controlling expression of a  
 42613 selection or iteration statement
- 42614 • The operand of a unary ('!') operator with the resulting expression being the entire  
 42615 controlling expression of a selection or iteration
- 42616 • The entire expression of an expression statement (possibly cast to **void**)

42617 **RETURN VALUE**

42618 If the return is from a successful direct invocation, *sigsetjmp()* shall return 0. If the return is from  
 42619 a call to *siglongjmp()*, *sigsetjmp()* shall return a non-zero value.

42620 **ERRORS**

42621 No errors are defined.

42622 **EXAMPLES**

42623 None.

42624 **APPLICATION USAGE**

42625 The distinction between *setjmp()/longjmp()* and *sigsetjmp()/siglongjmp()* is only significant for  
 42626 programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

42627 **RATIONALE**

42628 The ISO C standard specifies various restrictions on the usage of the *setjmp()* macro in order to  
 42629 permit implementors to recognize the name in the compiler and not implement an actual  
 42630 function. These same restrictions apply to the *sigsetjmp()* macro.

42631 There are processors that cannot easily support these calls, but this was not considered a  
 42632 sufficient reason to exclude them.

42633 4.2 BSD and 4.3 BSD systems provide functions named *\_setjmp()* and *\_longjmp()* that, together  
 42634 with *setjmp()* and *longjmp()*, provide the same functionality as *sigsetjmp()* and *siglongjmp()*. On  
 42635 those systems, *setjmp()* and *longjmp()* save and restore signal masks, while *\_setjmp()* and

42636 *\_longjmp()* do not. On System V, Release 3 and in corresponding issues of the SVID, *setjmp()* and  
42637 *longjmp()* are explicitly defined not to save and restore signal masks. In order to permit existing  
42638 practice in both cases, the relation of *setjmp()* and *longjmp()* to signal masks is not specified, and  
42639 a new set of functions is defined instead.

42640 The *longjmp()* and *siglongjmp()* functions operate as in the previous issue provided the matching  
42641 *setjmp()* or *sigsetjmp()* has been performed in the same thread. Non-local jumps into contexts  
42642 saved by other threads would be at best a questionable practice and were not considered worthy  
42643 of standardization.

#### 42644 FUTURE DIRECTIONS

42645 None.

#### 42646 SEE ALSO

42647 *siglongjmp()*, *signal()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of  
42648 IEEE Std. 1003.1-200x, <*setjmp.h*>

#### 42649 CHANGE HISTORY

42650 First released in Issue 3.

42651 Entry included for alignment with the POSIX.1-1988 standard.

#### 42652 Issue 4

42653 The DESCRIPTION states that *sigsetjmp()* is a macro or a function. Issue 3 states that it is a  
42654 macro. Warnings are also added about the suppression of a *sigsetjmp()* macro definition.

42655 A statement is added to the DESCRIPTION about the accessibility of objects after a *siglongjmp()*  
42656 call.

42657 Text is added to the DESCRIPTION describing the contexts in which calls to *sigsetjmp()* are  
42658 valid.

#### 42659 Issue 5

42660 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

#### 42661 Issue 6

42662 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.



42663 **NAME**

42664 sigsuspend — wait for a signal

42665 **SYNOPSIS**

42666 #include &lt;signal.h&gt;

42667 int sigsuspend(const sigset\_t \*sigmask);

42668 **DESCRIPTION**

42669 The *sigsuspend()* function replaces the current signal mask of the calling thread with the set of  
 42670 signals pointed to by *sigmask* and then suspends the thread until delivery of a signal whose  
 42671 action is either to execute a signal-catching function or to terminate the process. This shall not  
 42672 cause any other signals that may have been pending on the process to become pending on the  
 42673 thread.

42674 If the action is to terminate the process then *sigsuspend()* shall never return. If the action is to  
 42675 execute a signal-catching function, then *sigsuspend()* shall return after the signal-catching  
 42676 function returns, with the signal mask restored to the set that existed prior to the *sigsuspend()*  
 42677 call.

42678 It is not possible to block signals that cannot be ignored. This is enforced by the system without  
 42679 causing an error to be indicated.

42680 **RETURN VALUE**

42681 Since *sigsuspend()* suspends thread execution indefinitely, there is no successful completion  
 42682 return value. If a return occurs, -1 shall be returned and *errno* set to indicate the error.

42683 **ERRORS**42684 The *sigsuspend()* function shall fail if:

|       |         |                                                                            |
|-------|---------|----------------------------------------------------------------------------|
| 42685 | [EINTR] | A signal is caught by the calling process and control is returned from the |
| 42686 |         | signal-catching function.                                                  |

42687 **EXAMPLES**

42688 None.

42689 **APPLICATION USAGE**

42690 Normally, at the beginning of a critical code section, a specified set of signals is blocked using  
 42691 the *sigprocmask()* function. When the thread has completed the critical section and needs to wait  
 42692 for the previously blocked signal(s), it pauses by calling *sigsuspend()* with the mask that was  
 42693 returned by the *sigprocmask()* call.

42694 **RATIONALE**

42695 None.

42696 **FUTURE DIRECTIONS**

42697 None.

42698 **SEE ALSO**

42699 Section 2.4 (on page 528), *pause()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, the  
 42700 Base Definitions volume of IEEE Std. 1003.1-200x, <signal.h>

42701 **CHANGE HISTORY**

42702 First released in Issue 3.

42703 Entry included for alignment with the POSIX.1-1988 standard.

42704 **Issue 4**

42705 The term “signal handler” is changed to “signal-catching function”.

42706 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 42707 • The type of the argument *sigmask* is changed from **sigset\_t\*** to **const sigset\_t\***.

42708 **Issue 5**

42709 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42710 **Issue 6**

42711 The text in the RETURN VALUE section has been changed from “suspends process execution”  
42712 to “suspends thread execution”. This reflects IEEE PASC Interpretation 1003.1c #40.

42713 Text in the APPLICATION USAGE section has been replaced.

42714 **NAME**42715 sigtimedwait, sigwaitinfo — wait for queued signals (**REALTIME**)42716 **SYNOPSIS**42717 RTS 

```
#include <signal.h>
```

42718 

```
int sigtimedwait(const sigset_t *restrict set, siginfo_t *restrict info,
```

42719 

```
const struct timespec *restrict timeout);
```

42720 

```
int sigwaitinfo(const sigset_t *restrict set, siginfo_t *restrict info);
```

42721

42722 **DESCRIPTION**

42723 The *sigtimedwait()* function behaves the same as *sigwaitinfo()* except that if none of the signals  
 42724 specified by *set* are pending, *sigtimedwait()* waits for the time interval specified in the **timespec**  
 42725 structure referenced by *timeout*. If the **timespec** structure pointed to by *timeout* is zero-valued  
 42726 and if none of the signals specified by *set* are pending, then *sigtimedwait()* returns immediately  
 42727 with an error. If *timeout* is the NULL pointer, the behavior is unspecified. If the Monotonic Clock  
 42728 option is supported, the **CLOCK\_MONOTONIC** clock shall be used to measure the time interval  
 42729 specified by the *timeout* argument.

42730 The *sigwaitinfo()* function selects the pending signal from the set specified by *set*. Should any of  
 42731 multiple pending signals in the range **SIGRTMIN** to **SIGRTMAX** be selected, it shall be the  
 42732 lowest numbered one. The selection order between realtime and non-realtime signals, or  
 42733 between multiple pending non-realtime signals, is unspecified. If no signal in *set* is pending at  
 42734 the time of the call, the calling thread is suspended until one or more signals in *set* become  
 42735 pending or until it is interrupted by an unblocked, caught signal.

42736 The *sigwaitinfo()* function behaves the same as the *sigwait()* function if the *info* argument is  
 42737 NULL. If the *info* argument is non-NULL, the *sigwaitinfo()* function behaves the same as  
 42738 *sigwait()*, except that the selected signal number shall be stored in the *si\_signo* member, and the  
 42739 cause of the signal shall be stored in the *si\_code* member. If any value is queued to the selected  
 42740 signal, the first such queued value shall be dequeued and, if the *info* argument is non-NULL, the  
 42741 value shall be stored in the *si\_value* member of *info*. The system resource used to queue the  
 42742 signal shall be released and returned to the system for other use. If no value is queued, the  
 42743 content of the *si\_value* member is undefined. If no further signals are queued for the selected  
 42744 signal, the pending indication for that signal shall be reset.

42745 **RETURN VALUE**

42746 Upon successful completion (that is, one of the signals specified by *set* is pending or is  
 42747 generated) *sigwaitinfo()* and *sigtimedwait()* shall return the selected signal number. Otherwise,  
 42748 the function shall return a value of **-1** and set *errno* to indicate the error.

42749 **ERRORS**42750 The *sigtimedwait()* function shall fail if:42751 **Notes to Reviewers**42752 *This section with side shading will not appear in the final copy. - Ed.*

42753 D1, XSH, ERN 345 proposes that the [EAGAIN] error condition ought to be [ETIMEDOUT] as  
 42754 per the same condition on *pthread\_cond\_timedwait()*.

42755 [EAGAIN] No signal specified by *set* was generated within the specified timeout period.

42756 The *sigtimedwait()* and *sigwaitinfo()* functions may fail if:

42757 [EINTR] The wait was interrupted by an unblocked, caught signal. It shall be  
 42758 documented in system documentation whether this error causes these

42759 functions to fail.

42760 The *sigtimedwait()* function may also fail if:

42761 [EINVAL] The *timeout* argument specified a *tv\_nsec* value less than zero or greater than  
42762 or equal to 1 000 million.

42763 An implementation only checks for this error if no signal is pending in *set* and it is necessary to  
42764 wait.

#### 42765 EXAMPLES

42766 None.

#### 42767 APPLICATION USAGE

42768 None.

#### 42769 RATIONALE

42770 Existing programming practice on realtime systems uses the ability to pause waiting for a  
42771 selected set of events and handle the first event that occurs in-line instead of in a signal-handling  
42772 function. This allows applications to be written in an event-directed style similar to a state  
42773 machine. This style of programming is useful for largescale transaction processing in which the  
42774 overall throughput of an application and the ability to clearly track states are more important  
42775 than the ability to minimize the response time of individual event handling.

42776 It is possible to construct a signal-waiting macro function out of the realtime signal function  
42777 mechanism defined in this volume of IEEE Std. 1003.1-200x. However, such a macro has to  
42778 include the definition of a generalized handler for all signals to be waited on. A significant  
42779 portion of the overhead of handler processing can be avoided if the signal-waiting function is  
42780 provided by the kernel. This volume of IEEE Std. 1003.1-200x therefore provides two signal-  
42781 waiting functions—one that waits indefinitely and one with a timeout—as part of the overall  
42782 realtime signal function specification.

42783 The specification of a function with a timeout allows an application to be written that can be  
42784 broken out of a wait after a set period of time if no event has occurred. It was argued that setting  
42785 a timer event before the wait and recognizing the timer event in the wait would also implement  
42786 the same functionality, but at a lower performance level. Because of the performance  
42787 degradation associated with the user-level specification of a timer event and the subsequent  
42788 cancelation of that timer event after the wait completes for a valid event, and the complexity  
42789 associated with handling potential race conditions associated with the user-level method, the  
42790 separate function has been included.

42791 Note that the semantics of the *sigwaitinfo()* function are nearly identical to that of the *sigwait()*  
42792 function defined by this volume of IEEE Std. 1003.1-200x. The only difference is that *sigwaitinfo()*  
42793 returns the queued signal value in the *value* argument. The return of the queued value is  
42794 required so that applications can differentiate between multiple events queued to the same  
42795 signal number.

42796 The two distinct functions are being maintained because some implementations may choose to  
42797 implement the POSIX Threads Extension functions and not implement the queued signals  
42798 extensions. Note, though, that *sigwaitinfo()* does not return the queued value if the *value*  
42799 argument is NULL, so the POSIX Threads Extension *sigwait()* function can be implemented as a  
42800 macro on *sigwaitinfo()*.

42801 The *sigtimedwait()* function was separated from the *sigwaitinfo()* function to address concerns  
42802 regarding the overloading of the *timeout* pointer to indicate indefinite wait (no timeout), timed  
42803 wait, and immediate return, and concerns regarding consistency with other functions where the  
42804 conditional and timed waits were separate functions from the pure blocking function. The  
42805 semantics of *sigtimedwait()* are specified such that *sigwaitinfo()* could be implemented as a

42806 macro with a NULL pointer for *timeout*.

42807 The *sigwait* functions provide a synchronous mechanism for threads to wait for asynchronously  
42808 generated signals. One important question was how many threads that are suspended in a call to  
42809 a *sigwait()* function for a signal should return from the call when the signal is sent. Four choices  
42810 were considered:

- 42811 1. Return an error for multiple simultaneous calls to *sigwait* functions for the same signal.
- 42812 2. One or more threads return.
- 42813 3. All waiting threads return.
- 42814 4. Exactly one thread returns.

42815 Prohibiting multiple calls to *sigwait()* for the same signal was felt to be overly restrictive. The  
42816 “one or more” behavior made implementation of conforming packages easy at the expense of  
42817 forcing POSIX threads clients to protect against multiple simultaneous calls to *sigwait()* in  
42818 application code in order to achieve predictable behavior. There was concern that the “all  
42819 waiting threads” behavior would result in “signal broadcast storms”, consuming excessive CPU  
42820 resources by replicating the signals in the general case. Furthermore, no convincing examples  
42821 could be presented that delivery to all was either simpler or more powerful than delivery to one.

42822 Thus, the consensus was that exactly one thread that was suspended in a call to a *sigwait*  
42823 function for a signal should return when that signal occurs. This is not an onerous restriction as:

- 42824 • A multi-way signal wait can be built from the single-way wait.
- 42825 • Signals should only be handled by application-level code, as library routines cannot guess  
42826 what the application wants to do with signals generated for the entire process.
- 42827 • Applications can thus arrange for a single thread to wait for any given signal and call any  
42828 needed routines upon its arrival.

42829 In an application that is using signals for XSI interprocess communication, signal processing is  
42830 typically done in one place. Alternatively, if the signal is being caught so that process cleanup  
42831 can be done, the signal handler thread can call separate process cleanup routines for each  
42832 portion of the application. Since the application main line started each portion of the application,  
42833 it is at the right abstraction level to tell each portion of the application to clean up.

42834 Certainly, there exist programming styles where it is logical to consider waiting for a single  
42835 signal in multiple threads. A simple *sigwait\_multiple()* routine can be constructed to achieve this  
42836 goal. A possible implementation would be to have each *sigwait\_multiple()* caller registered as  
42837 having expressed interest in a set of signals. The caller then waits on a thread-specific condition  
42838 variable. A single server thread calls a *sigwait()* function on the union of all registered signals.  
42839 When the *sigwait()* function returns, the appropriate state is set and condition variables are  
42840 broadcast. New *sigwait\_multiple()* callers may cause the pending *sigwait()* call to be canceled  
42841 and reissued in order to update the set of signals being waited for.

#### 42842 **FUTURE DIRECTIONS**

42843 None.

#### 42844 **SEE ALSO**

42845 Section 2.8.1 (on page 543), *pause()*, *pthread\_sigmask()*, *sigaction()*, *sigpending()*, *sigsuspend()*,  
42846 *sigwait()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**signal.h**>, <**time.h**>

42847 **CHANGE HISTORY**

42848 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
42849 POSIX Threads Extension.

42850 **Issue 6**

42851 These functions are marked as part of the Realtime Signals Extension option.

42852 The Open Group corrigenda item U035/3 has been applied. The SYNOPSIS of the *sigwaitinfo()*  
42853 function has been corrected so that the second argument is of type **siginfo\_t\***.

42854 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
42855 implementation does not support the Realtime Signals Extension option.

42856 The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by specifying that the  
42857 CLOCK\_MONOTONIC clock, if supported, is used to measure timeout intervals.

42858 The **restrict** keyword is added to the *sigtimedwait()* and *sigwaitinfo()* prototypes for alignment  
42859 with the ISO/IEC 9899:1999 standard.

42860 **NAME**

42861 sigwait — wait for queued signals

42862 **SYNOPSIS**

42863 #include &lt;signal.h&gt;

42864 int sigwait(const sigset\_t \*restrict set, int \*restrict sig);

42865 **DESCRIPTION**

42866 The *sigwait()* function selects a pending signal from *set*, atomically clears it from the system's set  
 42867 of pending signals, and returns that signal number in the location referenced by *sig*. If prior to  
 42868 the call to *sigwait()* there are multiple pending instances of a single signal number, it is  
 42869 implementation-defined whether upon successful return there are any remaining pending  
 42870 signals for that signal number. If the implementation supports queued signals and there are  
 42871 multiple signals queued for the signal number selected, the first such queued signal shall cause a  
 42872 return from *sigwait()* and the remainder shall remain queued. If no signal in *set* is pending at the  
 42873 time of the call, the thread is suspended until one or more becomes pending. The signals defined  
 42874 by *set* shall have been blocked at the time of the call to *sigwait()*; otherwise, the behavior is  
 42875 undefined. The effect of *sigwait()* on the signal actions for the signals in *set* is unspecified.

42876 If more than one thread is using *sigwait()* to wait for the same signal, no more than one of these  
 42877 threads shall return from *sigwait()* with the signal number. Which thread returns from *sigwait()*  
 42878 if more than a single thread is waiting is unspecified.

42879 **RTS** Should any of the multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it  
 42880 shall be the lowest numbered one. The selection order between realtime and non-realtime  
 42881 signals, or between multiple pending non-realtime signals, is unspecified.

42882 **RETURN VALUE**

42883 Upon successful completion, *sigwait()* shall store the signal number of the received signal at the  
 42884 location referenced by *sig* and return zero. Otherwise, an error number shall be returned to  
 42885 indicate the error.

42886 **ERRORS**42887 The *sigwait()* function may fail if:42888 [EINVAL] The *set* argument contains an invalid or unsupported signal number.42889 **EXAMPLES**

42890 None.

42891 **APPLICATION USAGE**

42892 None.

42893 **RATIONALE**

42894 To provide a convenient way for a thread to wait for a signal, this volume of  
 42895 IEEE Std. 1003.1-200x provides the *sigwait()* function. For most cases where a thread has to wait  
 42896 for a signal, the *sigwait()* function should be quite convenient, efficient, and adequate.

42897 However, requests were made for a lower-level primitive than *sigwait()* and for semaphores that  
 42898 could be used by threads. After some consideration, threads were allowed to use semaphores  
 42899 and *sem\_post()* was defined to be async-signal and async-cancel-safe.

42900 In summary, when it is necessary for code run in response to an asynchronous signal to notify a  
 42901 thread, *sigwait()* should be used to handle the signal. Alternatively, if the implementation  
 42902 provides semaphores, they also can be used, either following *sigwait()* or from within a signal  
 42903 handling routine previously registered with *sigaction()*.

42904 **FUTURE DIRECTIONS**

42905           None.

42906 **SEE ALSO**

42907           Section 2.4 (on page 528), Section 2.8.1 (on page 543), *pause()*, *pthread\_sigmask()*, *sigaction()*,  
42908           *sigpending()*, *sigsuspend()*, *sigwaitinfo()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
42909           <**signal.h**>, <**time.h**>

42910 **CHANGE HISTORY**

42911           First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
42912           POSIX Threads Extension.

42913 **Issue 6**

42914           The RATIONALE section is added.

42915           The **restrict** keyword is added to the *sigwait()* prototype for alignment with the  
42916           ISO/IEC 9899:1999 standard.



42917 **NAME**42918 sigwaitinfo — wait for queued signals (**REALTIME**)42919 **SYNOPSIS**42920 RTS `#include <signal.h>`42921 `int sigwaitinfo(const sigset_t *restrict set, siginfo_t *restrict info);` |

42922

42923 **DESCRIPTION**42924 Refer to *sigtimedwait()*.

42925 **NAME**

42926 sin, sinf, sinl — sine function

42927 **SYNOPSIS**

42928 #include &lt;math.h&gt;

42929 double sin(double x);

42930 float sinf(float x);

42931 long double sinl(long double x);

42932 **DESCRIPTION**

42933 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 42934 conflict between the requirements described here and the ISO C standard is unintentional. This  
 42935 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

42936 These functions shall compute the sine of its argument *x*, measured in radians.

42937 An application wishing to check for error situations should set *errno* to 0 before calling *sin()*. If  
 42938 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

42939 The *sin()* function may lose accuracy when its argument is far from 0.0.42940 **RETURN VALUE**42941 Upon successful completion, these functions shall return the sine of *x*.42942 XSI If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

42943 XSI If *x* is  $\pm\text{Inf}$ , either 0.0 shall be returned and *errno* set to [EDOM], or NaN shall be returned and  
 42944 *errno* may be set to [EDOM].

42945 If the correct result would cause underflow, 0.0 shall be returned and *errno* may be set to  
 42946 [ERANGE].

42947 **ERRORS**

42948 These functions may fail if:

42949 XSI [EDOM] The value of *x* is NaN, or *x* is  $\pm\text{Inf}$ .

42950 [ERANGE] The result underflows.

42951 XSI No other errors shall occur.

42952 **EXAMPLES**42953 **Taking the Sine of a 45-Degree Angle**

42954 #include &lt;math.h&gt;

42955 ...

42956 double radians = 45.0 \* M\_PI / 180;

42957 double result;

42958 ...

42959 result = sin(radians);

42960 **APPLICATION USAGE**

42961 None.

42962 **RATIONALE**

42963 None.

42964 **FUTURE DIRECTIONS**

42965 None.

42966 **SEE ALSO**42967 *asin()*, *isnan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>42968 **CHANGE HISTORY**

42969 First released in Issue 1. Derived from Issue 1 of the SVID.

42970 **Issue 4**42971 References to *matherr()* are removed.42972 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the  
42973 ISO C standard and to rationalize error handling in the mathematics functions.

42974 The return value specified for [EDOM] is marked as an extension.

42975 **Issue 5**42976 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
42977 in previous issues.42978 **Issue 6**42979 The *sinf()* and *sinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

42980 **NAME**

42981       sinh, sinhf, sinhl — hyperbolic sine function

42982 **SYNOPSIS**

42983       #include <math.h>

42984       double sinh(double x);

42985       float sinhf(float x);

42986       long double sinhl(long double x);

42987 **DESCRIPTION**

42988 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
42989       conflict between the requirements described here and the ISO C standard is unintentional. This  
42990       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

42991       These functions shall compute the hyperbolic sine of  $x$ .

42992       An application wishing to check for error situations should set *errno* to 0 before calling *sinh()*. If  
42993       *errno* is non-zero on return, or the return value is NaN, an error has occurred.

42994 **RETURN VALUE**

42995       Upon successful completion, these functions shall return the hyperbolic sine of  $x$ .

42996       If the result would cause an overflow,  $\pm$ HUGE\_VAL shall be returned and *errno* set to  
42997       [ERANGE].

42998       If the result would cause underflow, 0.0 shall be returned and *errno* may be set to [ERANGE].

42999 **XSI**       If  $x$  is NaN, NaN shall be returned and *errno* may be set to [EDOM].

43000 **ERRORS**

43001       These functions shall fail if:

43002       [ERANGE]       The result would cause overflow.

43003       These functions may fail if:

43004 **XSI**       [EDOM]       The value of  $x$  is NaN.

43005       [ERANGE]       The result would cause underflow.

43006 **XSI**       No other errors shall occur.

43007 **EXAMPLES**

43008       None.

43009 **APPLICATION USAGE**

43010       None.

43011 **RATIONALE**

43012       None.

43013 **FUTURE DIRECTIONS**

43014       None.

43015 **SEE ALSO**

43016       *asinh()*, *cosh()*, *isnan()*, *tanh()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

43017 **CHANGE HISTORY**

43018       First released in Issue 1. Derived from Issue 1 of the SVID.

43019 **Issue 4**

43020 References to *matherr()* are removed.

43021 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the  
43022 ISO C standard and to rationalize error handling in the mathematics functions.

43023 The return value specified for [EDOM] is marked as an extension.

43024 **Issue 5**

43025 The DESCRIPTION is updated to indicate how an application should check for an error. This  
43026 text was previously published in the APPLICATION USAGE section.

43027 **Issue 6**

43028 The *sinhf()* and *sinhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

43029 **NAME**

43030 sleep — suspend execution for an interval of time

43031 **SYNOPSIS**

43032 #include <unistd.h>

43033 unsigned sleep(unsigned *seconds*);

43034 **DESCRIPTION**

43035 The *sleep()* function shall cause the calling thread to be suspended from execution until either  
43036 the number of realtime seconds specified by the argument *seconds* has elapsed or a signal is  
43037 delivered to the calling thread and its action is to invoke a signal-catching function or to  
43038 terminate the process. The suspension time may be longer than requested due to the scheduling  
43039 of other activity by the system.

43040 If a SIGALRM signal is generated for the calling process during execution of *sleep()* and if the  
43041 SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *sleep()*  
43042 returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also  
43043 unspecified whether it remains pending after *sleep()* returns or it is discarded.

43044 If a SIGALRM signal is generated for the calling process during execution of *sleep()*, except as a  
43045 result of a prior call to *alarm()*, and if the SIGALRM signal is not being ignored or blocked from  
43046 delivery, it is unspecified whether that signal has any effect other than causing *sleep()* to return.

43047 If a signal-catching function interrupts *sleep()* and examines or changes either the time a  
43048 SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or  
43049 whether the SIGALRM signal is blocked from delivery, the results are unspecified.

43050 If a signal-catching function interrupts *sleep()* and calls *siglongjmp()* or *longjmp()* to restore an  
43051 environment saved prior to the *sleep()* call, the action associated with the SIGALRM signal and  
43052 the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also  
43053 unspecified whether the SIGALRM signal is blocked, unless the process' signal mask is restored  
43054 as part of the environment.

43055 XSI Interactions between *sleep()* and any of *setitimer()*, *ualarm()*, or *usleep()* are unspecified.

43056 **RETURN VALUE**

43057 If *sleep()* returns because the requested time has elapsed, the value returned shall be 0. If *sleep()*  
43058 returns because of premature arousal due to delivery of a signal, the return value shall be the  
43059 “unslept” amount (the requested time minus the time actually slept) in seconds.

43060 **ERRORS**

43061 No errors are defined.

43062 **EXAMPLES**

43063 None.

43064 **APPLICATION USAGE**

43065 None.

43066 **RATIONALE**

43067 There are two general approaches to the implementation of the *sleep()* function. One is to use the  
43068 *alarm()* function to schedule a SIGALRM signal and then suspend the process waiting for that  
43069 signal. The other is to implement an independent facility. This volume of IEEE Std. 1003.1-200x  
43070 permits either approach.

43071 In order to comply with the requirement that no primitive shall change a process attribute unless  
43072 explicitly described by this volume of IEEE Std. 1003.1-200x, an implementation using SIGALRM  
43073 must carefully take into account any SIGALRM signal scheduled by previous *alarm()* calls, the

43074 action previously established for SIGALRM, and whether SIGALRM was blocked. If a SIGALRM  
 43075 has been scheduled before the *sleep()* would ordinarily complete, the *sleep()* must be shortened  
 43076 to that time and a SIGALRM generated (possibly simulated by direct invocation of the signal-  
 43077 catching function) before *sleep()* returns. If a SIGALRM has been scheduled after the *sleep()*  
 43078 would ordinarily complete, it must be rescheduled for the same time before *sleep()* returns. The  
 43079 action and blocking for SIGALRM must be saved and restored.

43080 Historical implementations often implement the SIGALRM-based version using *alarm()* and  
 43081 *pause()*. One such implementation is prone to infinite hangups, as described in *pause()*. Another  
 43082 such implementation uses the C-language *setjmp()* and *longjmp()* functions to avoid that  
 43083 window. That implementation introduces a different problem: when the SIGALRM signal  
 43084 interrupts a signal-catching function installed by the user to catch a different signal, the  
 43085 *longjmp()* aborts that signal-catching function. An implementation based on *sigprocmask()*,  
 43086 *alarm()*, and *sigsuspend()* can avoid these problems.

43087 Despite all reasonable care, there are several very subtle, but detectable and unavoidable,  
 43088 differences between the two types of implementations. These are the cases mentioned in this  
 43089 volume of IEEE Std. 1003.1-200x where some other activity relating to SIGALRM takes place,  
 43090 and the results are stated to be unspecified. All of these cases are sufficiently unusual as not to  
 43091 be of concern to most applications.

43092 See also the discussion of the term *realtime* in *alarm()*.

43093 Because *sleep()* can be implemented using *alarm()*, the discussion about alarms occurring early  
 43094 under *alarm()* applies to *sleep()* as well.

43095 Application writers should note that the type of the argument *seconds* and the return value of  
 43096 *sleep()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application  
 43097 cannot pass a value greater than the minimum guaranteed value for {UINT\_MAX}, which the  
 43098 ISO C standard sets as 65 535, and any application passing a larger value is restricting its  
 43099 portability. A different type was considered, but historical implementations, including those  
 43100 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

43101 Scheduling delays may cause the process to return from the *sleep()* function significantly after  
 43102 the requested time. In such cases, the return value should be set to zero, since the formula  
 43103 (requested time minus the time actually spent) yields a negative number and *sleep()* returns an  
 43104 **unsigned**.

#### 43105 FUTURE DIRECTIONS

43106 None.

#### 43107 SEE ALSO

43108 *alarm()*, *getitimer()*, *nanosleep()*, *pause()*, *sigaction()*, *sigsetjmp()*, *ualarm()*, *usleep()*, the Base  
 43109 Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>

#### 43110 CHANGE HISTORY

43111 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 43112 Issue 4

43113 The <**unistd.h**> header is added to the SYNOPSIS section.

#### 43114 Issue 4, Version 2

43115 The DESCRIPTION is updated to indicate possible interactions with the *setitimer()*, *ualarm()*,  
 43116 and *usleep()* functions.

43117 **Issue 5**

43118

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.



43119 **NAME**

43120 socket — create an endpoint for communication

43121 **SYNOPSIS**

43122 #include &lt;sys/socket.h&gt;

43123 int socket(int *domain*, int *type*, int *protocol*);43124 **DESCRIPTION**43125 The *socket()* function shall create an unbound socket in a communications domain, and return a  
43126 file descriptor that can be used in later function calls that operate on sockets.43127 The *socket()* function takes the following arguments:43128 *domain* Specifies the communications domain in which a socket is to be created.43129 *type* Specifies the type of socket to be created.43130 *protocol* Specifies a particular protocol to be used with the socket. Specifying a *protocol*  
43131 of 0 causes *socket()* to use an unspecified default protocol appropriate for the  
43132 requested socket type.43133 The *domain* argument specifies the address family used in the communications domain. The  
43134 address families supported by the system are implementation-defined.43135 Symbolic constants that can be used for the domain argument are defined in the <sys/socket.h>  
43136 header.43137 The *type* argument specifies the socket type, which determines the semantics of communication  
43138 over the socket. The socket types supported by the system are implementation-defined. Possible  
43139 socket types include:43140 SOCK\_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte  
43141 streams, and may provide a transmission mechanism for out-of-band  
43142 data.43143 SOCK\_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages  
43144 of fixed maximum length.43145 SOCK\_SEQPACKET Provides sequenced, reliable, bidirectional, connection-mode  
43146 transmission path for records. A record can be sent using one or more  
43147 output operations and received using one or more input operations, but a  
43148 single operation never transfers part of more than one record. Record  
43149 boundaries are visible to the receiver via the MSG\_EOR flag.43150 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address  
43151 family. The protocols supported by the system are implementation-defined.43152 The process may need to have appropriate privileges to use the *socket()* function or to create  
43153 some sockets.43154 **RETURN VALUE**43155 Upon successful completion, *socket()* shall return a non-negative integer, the socket file  
43156 descriptor. Otherwise, a value of -1 shall be returned and *errno* set to indicate the error.43157 **ERRORS**43158 The *socket()* function shall fail if:

43159 [EAFNOSUPPORT]

43160 The implementation does not support the specified address family.

- 43161 [EMFILE] No more file descriptors are available for this process.
- 43162 [ENFILE] No more file descriptors are available for the system.
- 43163 [EPROTONOSUPPORT]  
43164 The protocol is not supported by the address family, or the protocol is not  
43165 supported by the implementation.
- 43166 [EPROTOTYPE] The socket type is not supported by the protocol.
- 43167 The *socket()* function may fail if:
- 43168 [EACCES] The process does not have appropriate privileges.
- 43169 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 43170 [ENOMEM] Insufficient memory was available to fulfill the request.
- 43171 **EXAMPLES**
- 43172 None.
- 43173 **APPLICATION USAGE**
- 43174 The documentation for specific address families specifies which protocols each address family  
43175 supports. The documentation for specific protocols specifies which socket types each protocol  
43176 supports.
- 43177 The application can determine if an address family is supported by trying to create a socket with  
43178 *domain* set to the protocol in question.
- 43179 **RATIONALE**
- 43180 None.
- 43181 **FUTURE DIRECTIONS**
- 43182 None.
- 43183 **SEE ALSO**
- 43184 *accept()*, *bind()*, *connect()*, *getsockname()*, *getsockopt()*, *listen()*, *recv()*, *recvfrom()*, *recvmsg()*,  
43185 *send()*, *sendmsg()*, *setsockopt()*, *shutdown()*, *socketpair()*, the Base Definitions volume of  
43186 IEEE Std. 1003.1-200x, <netinet/in.h>, <sys/socket.h>
- 43187 **CHANGE HISTORY**
- 43188 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

43189 **NAME**

43190 socketpair — create a pair of connected sockets

43191 **SYNOPSIS**

43192 #include &lt;sys/socket.h&gt;

43193 int socketpair(int *domain*, int *type*, int *protocol*,  
43194 int *socket\_vector*[2]);43195 **DESCRIPTION**43196 The *socketpair()* function creates an unbound pair of connected sockets in a specified *domain*, of a  
43197 specified *type*, under the protocol optionally specified by the *protocol* argument. The two sockets  
43198 are identical. The file descriptors used in referencing the created sockets are returned in  
43199 *socket\_vector*[0] and *socket\_vector*[1].43200 The *socketpair()* function takes the following arguments:

|       |                      |                                                                                                                                                                                                                               |
|-------|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 43201 | <i>domain</i>        | Specifies the communications domain in which the sockets are to be created.                                                                                                                                                   |
| 43202 | <i>type</i>          | Specifies the type of sockets to be created.                                                                                                                                                                                  |
| 43203 | <i>protocol</i>      | Specifies a particular protocol to be used with the sockets. Specifying a<br>43204 <i>protocol</i> of 0 causes <i>socketpair()</i> to use an unspecified default protocol<br>43205 appropriate for the requested socket type. |
| 43206 | <i>socket_vector</i> | Specifies a 2-integer array to hold the file descriptors of the created socket<br>43207 pair.                                                                                                                                 |

43208 The *type* argument specifies the socket type, which determines the semantics of communications  
43209 over the socket. The socket types supported by the system are implementation-defined. Possible  
43210 socket types include:

|       |                |                                                                                                                                                                                                                                                                                                                                                                                     |
|-------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 43211 | SOCK_STREAM    | Provides sequenced, reliable, bidirectional, connection-mode byte<br>43212 streams, and may provide a transmission mechanism for out-of-band<br>43213 data.                                                                                                                                                                                                                         |
| 43214 | SOCK_DGRAM     | Provides datagrams, which are connectionless-mode, unreliable messages<br>43215 of fixed maximum length.                                                                                                                                                                                                                                                                            |
| 43216 | SOCK_SEQPACKET | Provides sequenced, reliable, bidirectional, connection-mode<br>43217 transmission paths for records. A record can be sent using one or more<br>43218 output operations and received using one or more input operations, but a<br>43219 single operation never transfers part of more than one record. Record<br>43220 boundaries are visible to the receiver via the MSG_EOR flag. |

43221 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address  
43222 family. The protocols supported by the system are implementation-defined.43223 The process may need to have appropriate privileges to use the *socketpair()* function or to create  
43224 some sockets.43225 **RETURN VALUE**43226 Upon successful completion, this function shall return 0; otherwise, -1 shall be returned and  
43227 *errno* set to indicate the error.43228 **ERRORS**43229 The *socketpair()* function shall fail if:

43230 [EAFNOSUPPORT]

43231 The implementation does not support the specified address family.

- 43232 [EMFILE] No more file descriptors are available for this process.
- 43233 [ENFILE] No more file descriptors are available for the system.
- 43234 [EOPNOTSUPP] The specified protocol does not permit creation of socket pairs.
- 43235 [EPROTONOSUPPORT]  
43236 The protocol is not supported by the address family, or the protocol is not  
43237 supported by the implementation.
- 43238 [EPROTOTYPE] The socket type is not supported by the protocol.
- 43239 The *socketpair()* function may fail if:
- 43240 [EACCES] The process does not have appropriate privileges.
- 43241 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 43242 [ENOMEM] Insufficient memory was available to fulfill the request.

**43243 EXAMPLES**

43244 None.

**43245 APPLICATION USAGE**

43246 The documentation for specific address families specifies which protocols each address family  
43247 supports. The documentation for specific protocols specifies which socket types each protocol  
43248 supports.

43249 The *socketpair()* function is used primarily with UNIX domain sockets and need not be  
43250 supported for other domains.

**43251 RATIONALE**

43252 None.

**43253 FUTURE DIRECTIONS**

43254 None.

**43255 SEE ALSO**

43256 *socket()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/socket.h>

**43257 CHANGE HISTORY**

43258 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

43259 **NAME**

43260        sprintf, snprintf — print formatted output

43261 **SYNOPSIS**

43262        #include &lt;stdio.h&gt;

43263        int snprintf(char \*restrict *s*, size\_t *n*,43264            const char \*restrict *format*, /\* args \*/ ...);43265        int sprintf(char \*restrict *s*, const char \*restrict *format*, ...);43266 **DESCRIPTION**43267        Refer to *fprintf()*.

43268 **NAME**

43269 sqrt, sqrtf, sqrtl — square root function

43270 **SYNOPSIS**

43271 #include &lt;math.h&gt;

43272 double sqrt(double x);

43273 float sqrtf(float x);

43274 long double sqrtl(long double x);

43275 **DESCRIPTION**

43276 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 43277 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43278 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

43279 These functions shall compute the square root of  $x$ ,  $\sqrt{x}$ .

43280 An application wishing to check for error situations should set *errno* to 0 before calling *sqrt()*. If  
 43281 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

43282 **RETURN VALUE**43283 Upon successful completion, these functions shall return the square root of  $x$ .43284 XSI If  $x$  is NaN, NaN shall be returned and *errno* may be set to [EDOM].43285 XSI If  $x$  is negative, 0.0 or NaN shall be returned and *errno* shall be set to [EDOM].43286 **ERRORS**

43287 These functions shall fail if:

43288 [EDOM] The value of  $x$  is negative.

43289 These functions may fail if:

43290 XSI [EDOM] The value of  $x$  is NaN.

43291 XSI No other errors shall occur.

43292 **EXAMPLES**43293 **Taking the Square Root of 9.0**

43294 #include &lt;math.h&gt;

43295 ...

43296 double x = 9.0;

43297 double result;

43298 ...

43299 result = sqrt(x);

43300 **APPLICATION USAGE**

43301 None.

43302 **RATIONALE**

43303 None.

43304 **FUTURE DIRECTIONS**

43305 None.

43306 **SEE ALSO**

43307 *isnan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>, <stdio.h>

43308 **CHANGE HISTORY**

43309 First released in Issue 1. Derived from Issue 1 of the SVID.

43310 **Issue 4**

43311 References to *matherr()* are removed.

43312 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the  
43313 ISO C standard and to rationalize error handling in the mathematics functions.

43314 The return value specified for [EDOM] is marked as an extension.

43315 **Issue 5**

43316 The DESCRIPTION is updated to indicate how an application should check for an error. This  
43317 text was previously published in the APPLICATION USAGE section.

43318 **Issue 6**

43319 The *sqrtrf()* and *sqrtrl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

43320 **NAME**

43321           srand — pseudo-random number generator

43322 **SYNOPSIS**

43323           #include <stdlib.h>

43324           void srand(unsigned *seed*);

43325 **DESCRIPTION**

43326           Refer to *rand()*.



43327 **NAME**

43328           srand48 — seed uniformly distributed double-precision pseudo-random number generator

43329 **SYNOPSIS**

43330 XSI        #include &lt;stdlib.h&gt;

43331            void srand48(long *seedval*);

43332

43333 **DESCRIPTION**43334            Refer to *drand48()*.

43335 **NAME**

43336           srandom — seed pseudo-random number generator

43337 **SYNOPSIS**

43338 XSI       #include <stdlib.h>

43339           void srandom(unsigned *seed*);

43340

43341 **DESCRIPTION**

43342           Refer to *initstate()*.

43343 **NAME**

43344        sscanf — convert formatted input

43345 **SYNOPSIS**

43346        #include &lt;stdio.h&gt;

43347        int sscanf(const char \*restrict *s*, const char \*restrict *format*, ...); |43348 **DESCRIPTION** |43349        Refer to *fscanf()*.

## 43350 NAME

43351 stat — get file status

## 43352 SYNOPSIS

43353 #include &lt;sys/stat.h&gt;

43354 int stat(const char \*restrict path, struct stat \*restrict buf);

## 43355 DESCRIPTION

43356 The *stat()* function obtains information about the named file and writes it to the area pointed to  
 43357 by the *buf* argument. The *path* argument points to a path name naming a file. Read, write, or  
 43358 execute permission of the named file is not required. An implementation that provides  
 43359 additional or alternate file access control mechanisms may, under implementation-defined  
 43360 conditions, cause *stat()* to fail. In particular, the system may deny the existence of the file  
 43361 specified by *path*.

43362 If the named file is a symbolic link, the *stat()* function shall continue path name resolution using  
 43363 the contents of the symbolic link, and shall return information pertaining to the resulting file if  
 43364 the file exists.

43365 The *buf* argument is a pointer to a **stat** structure, as defined in the header <sys/stat.h>, into  
 43366 which information is placed concerning the file.

43367 The *stat()* function updates any time-related fields (as described in the definition of **File Times**  
 43368 **Update** in the Base Definitions volume of IEEE Std. 1003.1-200x), before writing into the **stat**  
 43369 structure.

43370 The structure members *st\_mode*, *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atime*, *st\_ctime*, and *st\_mtime*  
 43371 shall have meaningful values for all file types defined in this volume of IEEE Std. 1003.1-200x.  
 43372 The value of the member *st\_nlink* shall be set to the number of links to the file.

## 43373 RETURN VALUE

43374 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 43375 indicate the error.

## 43376 ERRORS

43377 The *stat()* function shall fail if:

43378 [EACCES] Search permission is denied for a component of the path prefix.

43379 [EIO] An error occurred while reading from the file system.

43380 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
43381 argument.

43382 [ENAMETOOLONG]

43383 The length of the *path* argument exceeds {PATH\_MAX} or a path name  
43384 component is longer than {NAME\_MAX}.43385 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

43386 [ENOTDIR] A component of the path prefix is not a directory.

43387 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file  
43388 serial number cannot be represented correctly in the structure pointed to by  
43389 *buf*.43390 The *stat()* function may fail if:43391 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
43392 resolution of the *path* argument.

43393 [ENAMETOOLONG]  
 43394 As a result of encountering a symbolic link in resolution of the *path* argument,  
 43395 the length of the substituted path name string exceeded {PATH\_MAX}.

43396 [EOVERFLOW] A value to be stored would overflow one of the members of the **stat** structure.

43397 **EXAMPLES**43398 **Obtaining File Status Information**

43399 The following example shows how to obtain file status information for a file named  
 43400 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure.

```
43401 #include <sys/types.h>
43402 #include <sys/stat.h>
43403 #include <fcntl.h>
43404 struct stat buffer;
43405 int status;
43406 ...
43407 status = stat("/home/cnd/mod1", &buffer);
```

43408 **Getting Directory Information**

43409 The following example fragment gets status information for each entry in a directory. The call to  
 43410 the *stat()* function stores file information in the **stat** structure pointed to by *statbuf*. The lines  
 43411 that follow the *stat()* call format the fields in the **stat** structure for presentation to the user of the  
 43412 program.

```
43413 #include <sys/types.h>
43414 #include <sys/stat.h>
43415 #include <dirent.h>
43416 #include <pwd.h>
43417 #include <grp.h>
43418 #include <time.h>
43419 #include <locale.h>
43420 #include <langinfo.h>
43421 struct dirent *dp;
43422 struct stat statbuf;
43423 struct passwd *pwd;
43424 struct group *grp;
43425 struct tm *tm;
43426 char datestring[256];
43427 ...
43428 /* Loop through directory entries */
43429 while ((dp = readdir(dir)) != NULL) {
43430 /* Get entry's information. */
43431 if (stat(dp->d_name, &statbuf) == -1)
43432 continue;
43433 /* Print out type, permissions, and number of links. */
43434 printf("%10.10s", sperm (statbuf.st_mode));
43435 printf("%4d", statbuf.st_nlink);
```

```

43436 /* Print out owners name if it is found using getpwuid(). */
43437 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
43438 printf(" %-8.8s", pwd->pw_name);
43439 else
43440 printf(" %-8d", statbuf.st_uid);
43441
43442 /* Print out group name if it's found using getgrgid(). */
43443 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
43444 printf(" %-8.8s", grp->gr_name);
43445 else
43446 printf(" %-8d", statbuf.st_gid);
43447
43448 /* Print size of file. */
43449 printf("%9ld", statbuf.st_size);
43450
43451 tm = localtime(&statbuf.st_mtime);
43452
43453 /* Get localized date string. */
43454 strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
43455
43456 printf(" %s %s\n", datestring, dp->d_name);
43457 }

```

#### 43453 APPLICATION USAGE

43454 None.

#### 43455 RATIONALE

43456 The intent of the paragraph describing “additional or alternate file access control mechanisms”  
 43457 is to allow a secure implementation where a process with a label that does not dominate the  
 43458 file’s label cannot perform a *stat()* function. This is not related to read permission; a process with  
 43459 a label that dominates the file’s label does not need read permission. An implementation that  
 43460 supports write-up operations could fail *fstat()* function calls even though it has a valid file  
 43461 descriptor open for writing.

#### 43462 FUTURE DIRECTIONS

43463 None.

#### 43464 SEE ALSO

43465 *fstat()*, *lstat()*, *readlink()*, *symlink()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
 43466 `<sys/stat.h>`, `<sys/types.h>`

#### 43467 CHANGE HISTORY

43468 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 43469 Issue 4

43470 The `<sys/types.h>` header is now marked as optional (OH); this header need not be included on  
 43471 XSI-conformant systems.

43472 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 43473 • The type of argument *path* is changed from **char\*** to **const char\***.
- 43474 • In the DESCRIPTION is changed as follows:
  - 43475 — Statements indicating the purpose of this function and a paragraph defining the contents
  - 43476 of **stat** structure members are added.
  - 43477 — The words “extended security controls” are replaced by “additional or alternate file
  - 43478 access control mechanisms”.

- 43479 The following change is incorporated for alignment with the FIPS requirements:
- 43480 • In the **ERRORS** section, the condition whereby [ENAMETOOLONG] is returned if a path
  - 43481 name component is larger than {NAME\_MAX} is now defined as mandatory and marked as
  - 43482 an extension.
- 43483 **Issue 4, Version 2**
- 43484 The **ERRORS** section is updated for X/OPEN UNIX conformance as follows:
- 43485 • In the mandatory section, [EIO] is added to indicate that a physical I/O error has occurred,
  - 43486 and [ELOOP] to indicate that too many symbolic links were encountered during path name
  - 43487 resolution.
  - 43488 • In the optional section, a second [ENAMETOOLONG] condition is defined that may report
  - 43489 excessive length of an intermediate result of path name resolution of a symbolic link.
  - 43490 • In the optional section, [EOVERFLOW] is added to indicate that a value to be stored in a
  - 43491 member of the **stat** structure would cause overflow.
- 43492 **Issue 5**
- 43493 Large File Summit extensions are added.
- 43494 **Issue 6**
- 43495 In the **SYNOPSIS**, the inclusion of `<sys/types.h>` is no longer required.
- 43496 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:
- 43497 • The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.
  - 43498 This is since behavior may vary from one file system to another.
- 43499 The following new requirements on POSIX implementations derive from alignment with the
- 43500 Single UNIX Specification:
- 43501 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
  - 43502 required for conforming implementations of previous POSIX specifications, it was not
  - 43503 required for UNIX applications.
  - 43504 • The [EIO] mandatory error condition is added.
  - 43505 • The [ELOOP] mandatory error condition is added.
  - 43506 • The [EOVERFLOW] mandatory error condition is added. This change is to support large
  - 43507 files.
  - 43508 • The [ENAMETOOLONG] and the second [EOVERFLOW] optional error conditions are
  - 43509 added.
- 43510 The following changes were made to align with the IEEE P1003.1a draft standard:
- 43511 • Details are added regarding the treatment of symbolic links.
  - 43512 • The [ELOOP] optional error condition is added.
- 43513 The **DESCRIPTION** is updated to avoid use of the term “must” for application requirements.
- 43514 The **restrict** keyword is added to the `stat()` prototype for alignment with the ISO/IEC 9899: 1999
- 43515 standard.

43516 **NAME**

43517           statvfs — get file system information

43518 **SYNOPSIS**

43519 XSI       #include <sys/statvfs.h>

43520           int statvfs(const char \*restrict path, struct statvfs \*restrict buf);

43521

43522 **DESCRIPTION**

43523           Refer to *fstatvfs()*.



43524 **NAME**

43525 stderr, stdin, stdout — standard I/O streams

43526 **SYNOPSIS**

43527 #include &lt;stdio.h&gt;

43528 extern FILE \**stderr*, \**stdin*, \**stdout*;43529 **DESCRIPTION**

43530 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 43531 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43532 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

43533 A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type  
 43534 **FILE**. The *fopen()* function creates certain descriptive data for a stream and returns a pointer to  
 43535 designate the stream in all further transactions. Normally, there are three open streams with  
 43536 constant pointers declared in the <stdio.h> header and associated with the standard open files.

43537 At program start-up, three streams are predefined and need not be opened explicitly: *standard*  
 43538 *input* (for reading conventional input), *standard output* (for writing conventional output), and  
 43539 *standard error* (for writing diagnostic output). When opened, the standard error stream is not  
 43540 fully buffered; the standard input and standard output streams are fully buffered if and only if  
 43541 the stream can be determined not to refer to an interactive device.

43542 CX The following symbolic values in <unistd.h> define the file descriptors that shall be associated  
 43543 with the C-language *stdin*, *stdout*, and *stderr* when the application is started:

43544 STDIN\_FILENO Standard input value, *stdin*. Its value is 0.

43545 STDOUT\_FILENO Standard output value, *stdout*. Its value is 1.

43546 STDERR\_FILENO Standard error value, *stderr*. Its value is 2.

43547

43548 *stderr* is expected to be open for reading and writing.

43549 **RETURN VALUE**

43550 None.

43551 **ERRORS**

43552 No errors are defined.

43553 **EXAMPLES**

43554 None.

43555 **APPLICATION USAGE**

43556 None.

43557 **RATIONALE**

43558 None.

43559 **FUTURE DIRECTIONS**

43560 None.

43561 **SEE ALSO**

43562 *fclose()*, *feof()*, *ferror()*, *fileno()*, *fopen()*, *fread()*, *fseek()*, *getc()*, *gets()*, *popen()*, *printf()*, *putc()*,  
 43563 *puts()*, *read()*, *scanf()*, *setbuf()*, *setvbuf()*, *tmpfile()*, *ungetc()*, *vprintf()*, the Base Definitions  
 43564 volume of IEEE Std. 1003.1-200x, <stdio.h>, <unistd.h>

43565 **CHANGE HISTORY**

43566 First released in Issue 1.

43567 **Issue 6**

43568 Extensions beyond the ISO C standard are now marked.

43569 **NAME**

43570           strcasecmp, strncasecmp — case-insensitive string comparisons

43571 **SYNOPSIS**

43572 XSI       #include &lt;strings.h&gt;

43573           int strcmp(const char \*s1, const char \*s2);

43574           int strncasecmp(const char \*s1, const char \*s2, size\_t n);

43575

43576 **DESCRIPTION**

43577       The *strcasecmp()* function compares, while ignoring differences in case, the string pointed to by *s1* to the string pointed to by *s2*. The *strncasecmp()* function compares, while ignoring differences in case, not more than *n* bytes from the string pointed to by *s1* to the string pointed to by *s2*.

43581       In the POSIX locale, *strcasecmp()* and *strncasecmp()* do upper to lower conversions, then a byte comparison. The results are unspecified in other locales.

43583 **RETURN VALUE**

43584       Upon completion, *strcasecmp()* shall return an integer greater than, equal to, or less than 0, if the string pointed to by *s1* is, ignoring case, greater than, equal to, or less than the string pointed to by *s2*, respectively.

43587       Upon successful completion, *strncasecmp()* shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by *s1* is, ignoring case, greater than, equal to, or less than the possibly null-terminated array pointed to by *s2*, respectively.

43590 **ERRORS**

43591       No errors are defined.

43592 **EXAMPLES**

43593       None.

43594 **APPLICATION USAGE**

43595       None.

43596 **RATIONALE**

43597       None.

43598 **FUTURE DIRECTIONS**

43599       None.

43600 **SEE ALSO**

43601       The Base Definitions volume of IEEE Std. 1003.1-200x, &lt;strings.h&gt;

43602 **CHANGE HISTORY**

43603       First released in Issue 4, Version 2.

43604 **Issue 5**

43605       Moved from X/OPEN UNIX extension to BASE.

43606 **NAME**

43607           strcat — concatenate two strings

43608 **SYNOPSIS**

43609           #include &lt;string.h&gt;

43610           char \*strcat(char \*restrict *s1*, const char \*restrict *s2*);43611 **DESCRIPTION**

43612 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
43613 conflict between the requirements described here and the ISO C standard is unintentional. This  
43614 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

43615       The *strcat()* function shall append a copy of the string pointed to by *s2* (including the  
43616 terminating null byte) to the end of the string pointed to by *s1*. The initial byte of *s2* overwrites  
43617 the null byte at the end of *s1*. If copying takes place between objects that overlap, the behavior is  
43618 undefined.

43619 **RETURN VALUE**43620       The *strcat()* function shall return *s1*; no return value is reserved to indicate an error.43621 **ERRORS**

43622       No errors are defined.

43623 **EXAMPLES**

43624       None.

43625 **APPLICATION USAGE**

43626       This issue is aligned with the ISO C standard; this does not affect compatibility with XPG3  
43627 applications. Reliable error detection by this function was never guaranteed.

43628 **RATIONALE**

43629       None.

43630 **FUTURE DIRECTIONS**

43631       None.

43632 **SEE ALSO**43633       *strncat()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <string.h>43634 **CHANGE HISTORY**

43635       First released in Issue 1. Derived from Issue 1 of the SVID.

43636 **Issue 4**

43637       The **DESCRIPTION** is changed to make it clear that the function manipulates bytes rather than  
43638 (possibly multi-byte) characters.

43639       The following change is incorporated for alignment with the ISO C standard:

- 43640
  - The type of argument *s2* is changed from **char\*** to **const char\***.

43641 **Issue 6**43642       The *strcat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

43643 **NAME**

43644           strchr — string scanning operation

43645 **SYNOPSIS**

43646           #include &lt;string.h&gt;

43647           char \*strchr(const char \*s, int c);

43648 **DESCRIPTION**43649 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
43650 conflict between the requirements described here and the ISO C standard is unintentional. This  
43651 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.43652 **CX**       The *strchr()* function shall locate the first occurrence of *c* (converted to an **unsigned char**) in the  
43653 string pointed to by *s*. The terminating null byte is considered to be part of the string.43654 **RETURN VALUE**43655           Upon completion, *strchr()* shall return a pointer to the byte, or a null pointer if the byte was not  
43656 found.43657 **ERRORS**

43658           No errors are defined.

43659 **EXAMPLES**

43660           None.

43661 **APPLICATION USAGE**

43662           None.

43663 **RATIONALE**

43664           None.

43665 **FUTURE DIRECTIONS**

43666           None.

43667 **SEE ALSO**43668           *strchr()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <string.h>43669 **CHANGE HISTORY**

43670           First released in Issue 1. Derived from Issue 1 of the SVID.

43671 **Issue 4**43672           The **DESCRIPTION** and **RETURN VALUE** sections are changed to make it clear that the function  
43673 manipulates bytes rather than (possibly multi-byte) characters.43674           The **APPLICATION USAGE** section is removed.

43675           The following change is incorporated for alignment with the ISO C standard:

- 43676
- The type of argument *s* is changed from **char\*** to **const char\***.

43677 **Issue 6**

43678           Extensions beyond the ISO C standard are now marked.

43679 **NAME**43680 `stricmp` — compare two strings43681 **SYNOPSIS**43682 `#include <string.h>`43683 `int stricmp(const char *s1, const char *s2);`43684 **DESCRIPTION**

43685 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 43686 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43687 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

43688 The `stricmp()` function shall compare the string pointed to by `s1` to the string pointed to by `s2`.

43689 The sign of a non-zero return value shall be determined by the sign of the difference between the  
 43690 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings  
 43691 being compared.

43692 **RETURN VALUE**

43693 Upon completion, `stricmp()` shall return an integer greater than, equal to, or less than 0, if the  
 43694 string pointed to by `s1` is greater than, equal to, or less than the string pointed to by `s2`,  
 43695 respectively.

43696 **ERRORS**

43697 No errors are defined.

43698 **EXAMPLES**43699 **Checking a Password Entry**

43700 The following example compares the information read from standard input to the value of the  
 43701 name of the user entry. If the `stricmp()` function returns 0 (indicating a match), a further check  
 43702 will be made to see if the user entered the proper old password. The `crypt()` function is used to  
 43703 encrypt the old password entered by the user, using the value of the encrypted password in the  
 43704 **passwd** structure as the salt. If this value matches the value of the encrypted **passwd** in the  
 43705 structure, the entered password `oldpasswd` is the correct user's password. Finally, the program  
 43706 encrypts the new password so that it can store the information in the **passwd** structure.

```

43707 #include <string.h>
43708 #include <unistd.h>
43709 #include <stdio.h>
43710 ...
43711 int valid_change;
43712 struct passwd *p;
43713 char user[100];
43714 char oldpasswd[100];
43715 char newpasswd[100];
43716 char savepasswd[100];
43717 ...
43718 if (stricmp(p->pw_name, user) == 0) {
43719 if (stricmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
43720 strcpy(savepasswd, crypt(newpasswd, user));
43721 p->pw_passwd = savepasswd;
43722 valid_change = 1;
43723 }
43724 else {

```

```
43725 fprintf(stderr, "Old password is not valid\n");
43726 }
43727 }
43728 ...
```

**43729 APPLICATION USAGE**

43730 None.

**43731 RATIONALE**

43732 None.

**43733 FUTURE DIRECTIONS**

43734 None.

**43735 SEE ALSO**

43736 *strncmp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**string.h**>

**43737 CHANGE HISTORY**

43738 First released in Issue 1. Derived from Issue 1 of the SVID.

**43739 Issue 4**

43740 The DESCRIPTION is changed to make it clear that *strcmp()* compares bytes rather than  
43741 (possibly multi-byte) characters.

43742 The following change is incorporated for alignment with the ISO C standard:

- 43743 • The type of arguments *s1* and *s2* is changed from **char\*** to **const char\***.

**43744 Issue 6**

43745 Extensions beyond the ISO C standard are now marked.

43746 **NAME**

43747 strcoll — string comparison using collating information

43748 **SYNOPSIS**

43749 #include &lt;string.h&gt;

43750 int strcoll(const char \*s1, const char \*s2);

43751 **DESCRIPTION**

43752 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 43753 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43754 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

43755 The *strcoll()* function shall compare the string pointed to by *s1* to the string pointed to by *s2*,  
 43756 both interpreted as appropriate to the *LC\_COLLATE* category of the current locale.

43757 CX The *strcoll()* function shall not change the setting of *errno* if successful.

43758 Because no return value is reserved to indicate an error, an application wishing to check for error  
 43759 situations should set *errno* to 0, then call *strcoll()*, then check *errno*.

43760 **RETURN VALUE**

43761 Upon successful completion, *strcoll()* shall return an integer greater than, equal to, or less than 0,  
 43762 according to whether the string pointed to by *s1* is greater than, equal to, or less than the string  
 43763 CX pointed to by *s2* when both are interpreted as appropriate to the current locale. On error,  
 43764 *strcoll()* may set *errno*, but no return value is reserved to indicate an error.

43765 **ERRORS**43766 The *strcoll()* function may fail if:

43767 CX [EINVAL] The *s1* or *s2* arguments contain characters outside the domain of the collating  
 43768 sequence.

43769 **EXAMPLES**43770 **Comparing Nodes**

43771 The following example uses an application-defined function, *node\_compare()*, to compare two  
 43772 nodes based on an alphabetical ordering of the *string* field.

```
43773 #include <string.h>
43774 ...
43775 struct node { /* These are stored in the table. */
43776 char *string;
43777 int length;
43778 };
43779 ...
43780 int node_compare(const void *node1, const void *node2)
43781 {
43782 return strcoll(((const struct node *)node1)->string,
43783 ((const struct node *)node2)->string);
43784 }
43785 ...
```

43786 **APPLICATION USAGE**43787 The *strxfrm()* and *strcmp()* functions should be used for sorting large lists.



43788 **RATIONALE**

43789 None.

43790 **FUTURE DIRECTIONS**

43791 None.

43792 **SEE ALSO**43793 *strcmp()*, *strxfrm()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <string.h>43794 **CHANGE HISTORY**

43795 First released in Issue 3.

43796 **Issue 4**43797 A paragraph describing how the sign of the return value should be determined is removed from  
43798 the DESCRIPTION.

43799 The [EINVAL] error is marked as an extension.

43800 The following changes are incorporated for alignment with the ISO C standard:

- 43801 • The function is no longer marked as an extension.
- 43802 • The type of arguments *s1* and *s2* are changed from **char\*** to **const char\***.

43803 **Issue 5**43804 The DESCRIPTION is updated to indicate that *errno* does not be changed if the function is  
43805 successful.43806 **Issue 6**

43807 Extensions beyond the ISO C standard are now marked.

43808 The following new requirements on POSIX implementations derive from alignment with the  
43809 Single UNIX Specification:

- 43810 • The [EINVAL] optional error condition is added.

## 43811 NAME

43812 strcpy — copy a string

## 43813 SYNOPSIS

43814 #include &lt;string.h&gt;

43815 char \*strcpy(char \*restrict *s1*, const char \*restrict *s2*);

## 43816 DESCRIPTION

43817 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 43818 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43819 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

43820 The *strcpy()* function shall copy the string pointed to by *s2* (including the terminating null byte)  
 43821 into the array pointed to by *s1*. If copying takes place between objects that overlap, the behavior  
 43822 is undefined.

## 43823 RETURN VALUE

43824 The *strcpy()* function shall return *s1*; no return value is reserved to indicate an error.

## 43825 ERRORS

43826 No errors are defined.

## 43827 EXAMPLES

43828 **Initializing a String**43829 The following example copies the string "-----" into the *permstring* variable.

```
43830 #include <string.h>
43831 ...
43832 static char permstring[11];
43833 ...
43834 strcpy(permstring, "-----");
43835 ...
```

43836 **Storing a Key and Data**

43837 The following example allocates space for a key using *malloc()* then uses *strcpy()* to place the  
 43838 key there. Then it allocates space for data using *malloc()*, and uses *strcpy()* to place data there.  
 43839 (The user-defined function *dbfree()* frees memory previously allocated to an array of type **struct**  
 43840 **element**.)

```
43841 #include <string.h>
43842 #include <stdlib.h>
43843 #include <stdio.h>
43844 ...
43845 /* Structure used to read data and store it. */
43846 struct element {
43847 char *key;
43848 char *data;
43849 };
43850 struct element *tbl, *curtbl;
43851 char *key, *data;
43852 int count;
43853 ...
43854 void dbfree(struct element *, int);
```

```

43855 ...
43856 if ((curtbl->key = malloc(strlen(key) + 1)) == NULL) {
43857 perror("malloc"); dbfree(tbl, count); return NULL;
43858 }
43859 strcpy(curtbl->key, key);

43860 if ((curtbl->data = malloc(strlen(data) + 1)) == NULL) {
43861 perror("malloc"); free(curtbl->key); dbfree(tbl, count); return NULL;
43862 }
43863 strcpy(curtbl->data, data);
43864 ...

```

#### 43865 APPLICATION USAGE

43866 Character movement is performed differently in different implementations. Thus, overlapping  
 43867 moves may yield surprises.

43868 This issue is aligned with the ISO C standard; this does not affect compatibility with XPG3  
 43869 applications. Reliable error detection by this function was never guaranteed.

#### 43870 RATIONALE

43871 None.

#### 43872 FUTURE DIRECTIONS

43873 None.

#### 43874 SEE ALSO

43875 *strncpy()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**string.h**>

#### 43876 CHANGE HISTORY

43877 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 43878 Issue 4

43879 The DESCRIPTION is changed to make it clear that the function manipulates bytes rather than  
 43880 (possibly multi-byte) characters.

43881 The following change is incorporated for alignment with the ISO C standard:

- 43882 • The type of argument *s2* is changed from **char\*** to **const char\***.

#### 43883 Issue 6

43884 The *strcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

43885 **NAME**

43886 strcspn — get length of a complementary substring

43887 **SYNOPSIS**

43888 #include &lt;string.h&gt;

43889 size\_t strcspn(const char \*s1, const char \*s2);

43890 **DESCRIPTION**

43891 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
43892 conflict between the requirements described here and the ISO C standard is unintentional. This  
43893 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

43894 The *strcspn()* function shall compute the length of the maximum initial segment of the string  
43895 pointed to by *s1* which consists entirely of bytes *not* from the string pointed to by *s2*.

43896 **RETURN VALUE**

43897 The *strcspn()* function shall return the length of the computed segment of the string pointed to  
43898 by *s1*; no return value is reserved to indicate an error.

43899 **ERRORS**

43900 No errors are defined.

43901 **EXAMPLES**

43902 None.

43903 **APPLICATION USAGE**

43904 None.

43905 **RATIONALE**

43906 None.

43907 **FUTURE DIRECTIONS**

43908 None.

43909 **SEE ALSO**43910 *strspn()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <string.h>43911 **CHANGE HISTORY**

43912 First released in Issue 1. Derived from Issue 1 of the SVID.

43913 **Issue 4**

43914 The DESCRIPTION is changed to make it clear that the function manipulates bytes rather than  
43915 (possibly multi-byte) characters.

43916 The following change is incorporated for alignment with the ISO C standard:

- 43917 • The type of arguments *s1* and *s2* is changed from **char\*** to **const char\***.

43918 **Issue 5**

43919 The RETURN VALUE section is updated to indicated that *strcspn()* returns the length of *s1*, and  
43920 not *s1* itself as was previously stated.

43921 **Issue 6**

43922 The Open Group corrigenda item U030/1 has been applied. The text of the RETURN VALUE  
43923 section is updated to indicate that the computed segment length is returned, not the *s1* length.

43924 **NAME**

43925            strdup — duplicate a string

43926 **SYNOPSIS**

43927 XSI        #include &lt;string.h&gt;

43928            char \*strdup(const char \*s1);

43929

43930 **DESCRIPTION**

43931            The *strdup()* function shall return a pointer to a new string, which is a duplicate of the string pointed to by *s1*. The returned pointer can be passed to *free()*. A null pointer is returned if the new string cannot be created.

43934 **RETURN VALUE**

43935            The *strdup()* function shall return a pointer to a new string on success. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

43937 **ERRORS**43938            The *strdup()* function may fail if:

43939            [ENOMEM]       Storage space available is insufficient.

43940 **EXAMPLES**

43941            None.

43942 **APPLICATION USAGE**

43943            None.

43944 **RATIONALE**

43945            None.

43946 **FUTURE DIRECTIONS**

43947            None.

43948 **SEE ALSO**43949            *free()*, *malloc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <string.h>43950 **CHANGE HISTORY**

43951            First released in Issue 4, Version 2.

43952 **Issue 5**

43953            Moved from X/OPEN UNIX extension to BASE.

43954 **NAME**

43955            strerror, strerror\_r — get error message string

43956 **SYNOPSIS**

43957            #include &lt;string.h&gt;

43958            char \*strerror(int *errnum*);43959            int strerror\_r(int *errnum*, char \**strerrbuf*, size\_t *buflen*);43960 **DESCRIPTION**

43961            The *strerror()* function maps the error number in *errnum* to a locale-dependent error message string and returns a pointer to it. The string pointed to must not be modified by the application, but may be overwritten by a subsequent call to *strerror()* or *perror()*.

43962            The contents of the error message strings returned by *strerror()* should be determined by the setting of the *LC\_MESSAGES* category in the current locale.

43963            The implementation shall behave as if no function defined in this volume of IEEE Std. 1003.1-200x calls *strerror()*.

43964            The *strerror()* function shall not change the setting of *errno* if successful.

43965            Because no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strerror()*, then check *errno*.

43966            The *strerror()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

43967            The *strerror\_r()* function maps the error number in *errnum* to a locale-dependent error message string and returns the string in the buffer pointed to by *strerrbuf*, with length *buflen*.

43975 **RETURN VALUE**

43976            Upon successful completion, *strerror()* shall return a pointer to the generated message string. On error *errno* may be set, but no return value is reserved to indicate an error.

43977            Upon successful completion, *strerror\_r()* returns 0. Otherwise, an error number is returned to indicate the error.

43980 **ERRORS**

43981            These functions may fail if:

43982            [EINVAL]            The value of *errnum* is not a valid error number.

43983            The *strerror\_r()* function may fail if:

43984            [ERANGE]            Insufficient storage was supplied via *strerrbuf* and *buflen* to contain the generated message string.

43986 **EXAMPLES**

43987            None.

43988 **APPLICATION USAGE**

43989            None.

43990 **RATIONALE**

43991            None.

43992 **FUTURE DIRECTIONS**

43993            None.

43994 **SEE ALSO**

43995 *perror()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <string.h>

43996 **CHANGE HISTORY**

43997 First released in Issue 3.

43998 **Issue 4**

43999 The DESCRIPTION is changed as follows:

- 44000 • The term “language-dependent” is replaced by “locale-dependent”.
- 44001 • A statement about the use of the *LC\_MESSAGES* category for determining the language of
- 44002 error messages is added and marked as an extension.

44003 The fact that *strerror()* can return a null pointer on failure and set *errno* is marked as an

44004 extension.

44005 The [EINVAL] error is marked as an extension.

44006 The FUTURE DIRECTIONS section is removed.

44007 The following change is incorporated for alignment with the ISO C standard:

- 44008 • The function is no longer marked as an extension.

44009 **Issue 5**

44010 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

44011 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

44012 **Issue 6**

44013 Extensions beyond the ISO C standard are now marked.

44014 The following new requirements on POSIX implementations derive from alignment with the

44015 Single UNIX Specification:

- 44016 • In the RETURN VALUE section, the fact that *errno* may be set is added.
- 44017 • The [EINVAL] optional error condition is added.

44018 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

44019 The *strerror\_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.

## 44020 NAME

44021 strfmon — convert monetary value to a string

## 44022 SYNOPSIS

44023 XSI 

```
#include <monetary.h>
```

44024 

```
ssize_t strfmon(char *restrict s, size_t maxsize,
44025 const char *restrict format, ...);
```

44026

## 44027 DESCRIPTION

44028 The *strfmon()* function shall place characters into the array pointed to by *s* as controlled by the  
44029 string pointed to by *format*. No more than *maxsize* bytes are placed into the array.44030 The format is a character string that contains two types of objects: *plain characters*, which are  
44031 simply copied to the output stream, and *conversion specifications*, each of which results in the  
44032 fetching of zero or more arguments which are converted and formatted. The results are  
44033 undefined if there are insufficient arguments for the format. If the format is exhausted while  
44034 arguments remain, the excess arguments are simply ignored.

44035 The application shall ensure that a conversion specification consists of the following sequence:

- 44036 • A '%' character
- 44037 • Optional flags
- 44038 • Optional field width
- 44039 • Optional left precision
- 44040 • Optional right precision
- 44041 • A required conversion character that determines the conversion to be performed

44042 **Flags**

44043 One or more of the following optional flags can be specified to control the conversion:

- 44044 =*f* An '=' followed by a single character *f* which is used as the numeric fill character. The  
44045 application shall ensure that the fill character is representable in a single byte in order  
44046 to work with precision and width counts. The default numeric fill character is the  
44047 <space> character. This flag does not affect field width filling which always uses the  
44048 <space> character. This flag is ignored unless a left precision (see below) is specified.
- 44049 ^ Do not format the currency amount with grouping characters. The default is to insert  
44050 the grouping characters if defined for the current locale.
- 44051 + or ( Specify the style of representing positive and negative currency amounts. Only one of  
44052 '+' or '(' may be specified. If '+' is specified, the locale's equivalent of '+' and '-'  
44053 are used (for example, in the U.S., the empty string if positive and '-' if negative). If  
44054 '(' is specified, negative amounts are enclosed within parentheses. If neither flag is  
44055 specified, the '+' style is used.
- 44056 ! Suppress the currency symbol from the output conversion.
- 44057 - Specify the alignment. If this flag is present all fields are left-justified (padded to the  
44058 right) rather than right-justified.



44059 **Field Width**

44060 *w* A decimal digit string *w* specifying a minimum field width in bytes in which the result  
 44061 of the conversion is right-justified (or left-justified if the flag '-' is specified). The  
 44062 default is 0.

44063 **Left Precision**

44064 *#n* A '#' followed by a decimal digit string *n* specifying a maximum number of digits  
 44065 expected to be formatted to the left of the radix character. This option can be used to  
 44066 keep the formatted output from multiple calls to the *strfmon()* function aligned in the  
 44067 same columns. It can also be used to fill unused positions with a special character as in  
 44068 "\$\*\*\*123.45". This option causes an amount to be formatted as if it has the number  
 44069 of digits specified by *n*. If more than *n* digit positions are required, this conversion  
 44070 specification is ignored. Digit positions in excess of those actually required are filled  
 44071 with the numeric fill character (see the *=f* flag above).

44072 If grouping has not been suppressed with the '^' flag, and it is defined for the current  
 44073 locale, grouping separators are inserted before the fill characters (if any) are added.  
 44074 Grouping separators are not applied to fill characters even if the fill character is a digit.

44075 To ensure alignment, any characters appearing before or after the number in the  
 44076 formatted output such as currency or sign symbols are padded as necessary with  
 44077 <space> characters to make their positive and negative formats an equal length.

44078 **Right Precision**

44079 *.p* A period followed by a decimal digit string *p* specifying the number of digits after the  
 44080 radix character. If the value of the right precision *p* is 0, no radix character appears. If a  
 44081 right precision is not included, a default specified by the current locale is used. The  
 44082 amount being formatted is rounded to the specified number of digits prior to  
 44083 formatting.

44084 **Conversion Characters**

44085 The conversion characters and their meanings are:

44086 *i* The **double** argument is formatted according to the locale's international currency  
 44087 format (for example, in the U.S.: USD 1,234.56).

44088 *n* The **double** argument is formatted according to the locale's national currency format  
 44089 (for example, in the U.S.: \$1,234.56).

44090 *%* Convert to a '%'; no argument is converted. The entire conversion specification shall  
 44091 be "%%".

44092 **Locale Information**

44093 The *LC\_MONETARY* category of the program's locale affects the behavior of this function  
 44094 including the monetary radix character (which may be different from the numeric radix  
 44095 character affected by the *LC\_NUMERIC* category), the grouping separator, the currency  
 44096 symbols, and formats. The international currency symbol should be conformant with the  
 44097 ISO 4217:1995 standard.

44098 If the value of *maxsize* is greater than {SSIZE\_MAX}, the result is implementation-defined.

## 44099 RETURN VALUE

44100 If the total number of resulting bytes including the terminating null byte is not more than  
 44101 *maxsize*, *strfmon()* shall return the number of bytes placed into the array pointed to by *s*, not  
 44102 including the terminating null byte. Otherwise, -1 shall be returned, the contents of the array are  
 44103 indeterminate, and *errno* shall be set to indicate the error.

## 44104 ERRORS

44105 The *strfmon()* function shall fail if:

44106 [E2BIG] Conversion stopped due to lack of space in the buffer.

## 44107 EXAMPLES

44108 Given a locale for the U.S. and the values 123.45, -123.45, and 3456.781:

| Conversion Specification | Output                                      | Comments                                                                   |
|--------------------------|---------------------------------------------|----------------------------------------------------------------------------|
| %n                       | \$123.45<br>-\$123.45<br>\$3,456.78         | Default formatting.                                                        |
| %11n                     | \$123.45<br>-\$123.45<br>\$3,456.78         | Right align within an 11 character field.                                  |
| ##5n                     | \$ 123.45<br>-\$ 123.45<br>\$ 3,456.78      | Aligned columns for values up to 99,999.                                   |
| %=*#5n                   | \$***123.45<br>-\$***123.45<br>\$*3,456.78  | Specify a fill character.                                                  |
| %=#5n                    | \$000123.45<br>-\$000123.45<br>\$03,456.78  | Fill characters do not use grouping even if the fill character is a digit. |
| %^#5n                    | \$ 123.45<br>-\$ 123.45<br>\$ 3456.78       | Disable the grouping separator.                                            |
| %^#5.0n                  | \$ 123<br>-\$ 123<br>\$ 3457                | Round off to whole units.                                                  |
| %^#5.4n                  | \$ 123.4500<br>-\$ 123.4500<br>\$ 3456.7810 | Increase the precision.                                                    |
| %(#5n                    | 123.45<br>(\$ 123.45)<br>\$ 3,456.78        | Use an alternative pos/neg style.                                          |
| %(!#5n                   | 123.45<br>( 123.45)<br>3,456.78             | Disable the currency symbol.                                               |

## 44141 APPLICATION USAGE

44142 None.

44143 **RATIONALE**

44144           None.

44145 **FUTURE DIRECTIONS**44146           Lowercase conversion characters are reserved for future standards use and uppercase for  
44147           implementation-defined use.44148 **SEE ALSO**44149           *localeconv()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**monetary.h**>44150 **CHANGE HISTORY**

44151           First released in Issue 4.

44152 **Issue 5**

44153           Moved from ENHANCED I18N to BASE.

44154           The [ENOSYS] error is removed.

44155           A sentence is added to the DESCRIPTION warning about values of *maxsize* that are greater than  
44156           {SSIZE\_MAX}.44157 **Issue 6**

44158           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

44159           The **restrict** keyword is added to the *strfmon()* prototype for alignment with the  
44160           ISO/IEC 9899:1999 standard.

## 44161 NAME

44162 strftime — convert date and time to a string

## 44163 SYNOPSIS

44164 #include &lt;time.h&gt;

```
44165 size_t strftime(char *restrict s, size_t maxsize,
44166 const char *restrict format, const struct tm *restrict timptr);
```

## 44167 DESCRIPTION

44168 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 44169 conflict between the requirements described here and the ISO C standard is unintentional. This  
 44170 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

44171 The *strftime()* function shall place bytes into the array pointed to by *s* as controlled by the string  
 44172 pointed to by *format*. The *format* string consists of zero or more conversion specifications and  
 44173 ordinary characters. A conversion specification consists of a '%' character, possibly followed by  
 44174 an *E* or *O* modifier, and a terminating conversion character that determines the conversion  
 44175 specification's behavior. All ordinary characters (including the terminating null byte) are copied  
 44176 unchanged into the array. If copying takes place between objects that overlap, the behavior is  
 44177 undefined. No more than *maxsize* bytes are placed into the array. Each conversion specification  
 44178 is replaced by appropriate characters as described in the following list. The appropriate  
 44179 characters are determined by the program's locale and by the values contained in the structure  
 44180 pointed to by *timptr*.

44181 CX Local timezone information is used as though *strftime()* called *tzset()*.

|       |    |                                                                                         |
|-------|----|-----------------------------------------------------------------------------------------|
| 44182 | %a | Replaced by the locale's abbreviated weekday name.                                      |
| 44183 | %A | Replaced by the locale's full weekday name.                                             |
| 44184 | %b | Replaced by the locale's abbreviated month name.                                        |
| 44185 | %B | Replaced by the locale's full month name.                                               |
| 44186 | %c | Replaced by the locale's appropriate date and time representation.                      |
| 44187 | %C | Replaced by the century number (the year divided by 100 and truncated to an integer)    |
| 44188 |    | as a decimal number [00-99].                                                            |
| 44189 | %d | Replaced by the day of the month as a decimal number [01,31].                           |
| 44190 | %D | Same as %m/%d/%y.                                                                       |
| 44191 | %e | Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded |
| 44192 |    | by a space.                                                                             |
| 44193 | %F | Equivalent to %Y-%m-%d (the ISO 8601: 1988 standard date format).                       |
| 44194 | %g | Replaced by the last 2 digits of the week-based year (see below) as a decimal number    |
| 44195 |    | (00-99).                                                                                |
| 44196 | %G | Replaced by the week-based year (see below) as a decimal number (for example, 1977).    |
| 44197 | %h | Same as %b.                                                                             |
| 44198 | %H | Replaced by the hour (24-hour clock) as a decimal number [00,23].                       |
| 44199 | %I | Replaced by the hour (12-hour clock) as a decimal number [01,12].                       |
| 44200 | %j | Replaced by the day of the year as a decimal number [001,366].                          |

|       |                 |                                                                                                   |
|-------|-----------------|---------------------------------------------------------------------------------------------------|
| 44201 | <code>%m</code> | Replaced by the month as a decimal number [01,12].                                                |
| 44202 | <code>%M</code> | Replaced by the minute as a decimal number [00,59].                                               |
| 44203 | <code>%n</code> | Replaced by a <newline> character.                                                                |
| 44204 | <code>%p</code> | Replaced by the locale's equivalent of either a.m. or p.m.                                        |
| 44205 | <code>%r</code> | Replaced by the time in a.m. and p.m. notation; in the POSIX locale this is equivalent to         |
| 44206 |                 | <code>%I:%M:%S %p</code> .                                                                        |
| 44207 | <code>%R</code> | Replaced by the time in 24 hour notation ( <code>%H:%M</code> ).                                  |
| 44208 | <code>%S</code> | Replaced by the second as a decimal number [00,61].                                               |
| 44209 | <code>%t</code> | Replaced by a <tab> character.                                                                    |
| 44210 | <code>%T</code> | Replaced by the time ( <code>%H:%M:%S</code> ).                                                   |
| 44211 | <code>%u</code> | Replaced by the weekday as a decimal number [1,7], with 1 representing Monday.                    |
| 44212 | <code>%U</code> | Replaced by the week number of the year (Sunday as the first day of the week) as a                |
| 44213 |                 | decimal number [00,53].                                                                           |
| 44214 | <code>%V</code> | Replaced by the week number of the year (Monday as the first day of the week) as a                |
| 44215 |                 | decimal number [01,53]. If the week containing 1 January has four or more days in the             |
| 44216 |                 | new year, then it is considered week 1. Otherwise, it is the last week of the previous            |
| 44217 |                 | year, and the next week is week 1.                                                                |
| 44218 | <code>%w</code> | Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday.                    |
| 44219 | <code>%W</code> | Replaced by the week number of the year (Monday as the first day of the week) as a                |
| 44220 |                 | decimal number [00,53]. All days in a new year preceding the first Monday are                     |
| 44221 |                 | considered to be in week 0.                                                                       |
| 44222 | <code>%x</code> | Replaced by the locale's appropriate date representation.                                         |
| 44223 | <code>%X</code> | Replaced by the locale's appropriate time representation.                                         |
| 44224 | <code>%y</code> | Replaced by the year without century as a decimal number [00,99].                                 |
| 44225 | <code>%Y</code> | Replaced by the year with century as a decimal number.                                            |
| 44226 | <code>%z</code> | Replaced by the offset from UTC in the ISO 8601:1988 standard format <code>-0430</code> (meaning  |
| 44227 |                 | 4 hours 30 minutes behind UTC, west of Greenwich), or by no characters if no timezone             |
| 44228 |                 | is determinable.                                                                                  |
| 44229 | <code>%Z</code> | Replaced by the timezone name or abbreviation, or by no bytes if no timezone                      |
| 44230 |                 | information exists.                                                                               |
| 44231 | <code>%%</code> | Replaced by <code>'%'</code> .                                                                    |
| 44232 |                 | If a conversion specification does not correspond to any of the above, the behavior is undefined. |

#### 44233 **Modified Conversion Specifiers**

|       |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 44234 | <code>CX</code>  | Some conversion specifiers can be modified by the <i>E</i> or <i>O</i> modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale, (see ERA in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 7.3.5, LC_TIME) the behavior shall be as if the unmodified conversion specification were used. |
| 44235 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 44236 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 44237 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 44238 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 44239 | <code>%Ec</code> | Replaced by the locale's alternative appropriate date and time representation.                                                                                                                                                                                                                                                                                                                                                                                                                         |

|       |            |                                                                                                                                                                                                    |
|-------|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 44240 | <b>%EC</b> | Replaced by the name of the base year (period) in the locale's alternative representation.                                                                                                         |
| 44241 |            |                                                                                                                                                                                                    |
| 44242 | <b>%Ex</b> | Replaced by the locale's alternative date representation.                                                                                                                                          |
| 44243 | <b>%EX</b> | Replaced by the locale's alternative time representation.                                                                                                                                          |
| 44244 | <b>%Ey</b> | Replaced by the offset from <b>%EC</b> (year only) in the locale's alternative representation.                                                                                                     |
| 44245 | <b>%EY</b> | Replaced by the full alternative year representation.                                                                                                                                              |
| 44246 | <b>%Od</b> | Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero; otherwise, with leading spaces. |
| 44247 |            |                                                                                                                                                                                                    |
| 44248 |            |                                                                                                                                                                                                    |
| 44249 | <b>%Oe</b> | Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading spaces.                                                                            |
| 44250 |            |                                                                                                                                                                                                    |
| 44251 | <b>%OH</b> | Replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.                                                                                                               |
| 44252 | <b>%OI</b> | Replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.                                                                                                               |
| 44253 | <b>%Om</b> | Replaced by the month using the locale's alternative numeric symbols.                                                                                                                              |
| 44254 | <b>%OM</b> | Replaced by the minutes using the locale's alternative numeric symbols.                                                                                                                            |
| 44255 | <b>%OS</b> | Replaced by the seconds using the locale's alternative numeric symbols.                                                                                                                            |
| 44256 | <b>%Ou</b> | Replaced by the weekday as a number in the locale's alternative representation (Monday=1).                                                                                                         |
| 44257 |            |                                                                                                                                                                                                    |
| 44258 | <b>%OU</b> | Replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to <b>%U</b> ) using the locale's alternative numeric symbols.                                   |
| 44259 |            |                                                                                                                                                                                                    |
| 44260 | <b>%OV</b> | Replaced by the week number of the year (Monday as the first day of the week, rules corresponding to <b>%V</b> ) using the locale's alternative numeric symbols.                                   |
| 44261 |            |                                                                                                                                                                                                    |
| 44262 | <b>%Ow</b> | Replaced by the number of the weekday (Sunday=0) using the locale's alternative numeric symbols.                                                                                                   |
| 44263 |            |                                                                                                                                                                                                    |
| 44264 | <b>%OW</b> | Replaced by the week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.                                                                      |
| 44265 |            |                                                                                                                                                                                                    |
| 44266 | <b>%Oy</b> | Replaced by the year (offset from <b>%C</b> ) using the locale's alternative numeric symbols.                                                                                                      |
| 44267 |            |                                                                                                                                                                                                    |
| 44268 |            |                                                                                                                                                                                                    |
| 44269 |            |                                                                                                                                                                                                    |
| 44270 |            |                                                                                                                                                                                                    |
| 44271 |            |                                                                                                                                                                                                    |
| 44272 |            |                                                                                                                                                                                                    |
| 44273 |            |                                                                                                                                                                                                    |
| 44274 |            |                                                                                                                                                                                                    |
| 44275 |            |                                                                                                                                                                                                    |
| 44276 |            |                                                                                                                                                                                                    |
| 44277 |            |                                                                                                                                                                                                    |
| 44278 |            |                                                                                                                                                                                                    |
| 44279 |            |                                                                                                                                                                                                    |
| 44280 |            |                                                                                                                                                                                                    |

44281 **ERRORS**

44282 No errors are defined.

44283 **EXAMPLES**44284 **Getting a Localized Date String**

44285 The following example first sets the locale to the user's default. The locale information will be  
 44286 used in the `nl_langinfo()` and `strptime()` functions. The `nl_langinfo()` function returns the localized  
 44287 date string which specifies how the date is laid out. The `strptime()` function takes this information  
 44288 and, using the `tm` structure for values, places the date and time information into `datestring`.

```
44289 #include <time.h>
44290 #include <locale.h>
44291 #include <langinfo.h>
44292 ...
44293 struct tm *tm;
44294 char datestring[256];
44295 ...
44296 setlocale (LC_ALL, "");
44297 ...
44298 strptime (datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
44299 ...
```

44300 **APPLICATION USAGE**

44301 The range of values for `%S` is [00,61] rather than [00,59] to allow for the occasional leap second  
 44302 and even more infrequent double leap second.

44303 Some of the conversion specifications marked EX are duplicates of others. They are included for  
 44304 compatibility with `nl_cxtime()` and `nl_ascxtime()`, which were published in Issue 2.

44305 Applications should use `%Y` (4-digit years) in preference to `%y` (2-digit years).

44306 In the C locale, the `E` and `O` modifiers are ignored and the replacement strings for the following  
 44307 specifiers are:

|       |                 |                                                 |
|-------|-----------------|-------------------------------------------------|
| 44308 | <code>%a</code> | The first three characters of <code>%A</code> . |
| 44309 | <code>%A</code> | One of Sunday, Monday, ..., Saturday.           |
| 44310 | <code>%b</code> | The first three characters of <code>%B</code> . |
| 44311 | <code>%B</code> | One of January, February, ..., December.        |
| 44312 | <code>%c</code> | Equivalent to <code>%a %b %e %T %Y</code> .     |
| 44313 | <code>%p</code> | One of AM or PM.                                |
| 44314 | <code>%r</code> | Equivalent to <code>%I:%M:%S %p</code> .        |
| 44315 | <code>%x</code> | Equivalent to <code>%m/%d/%y</code> .           |
| 44316 | <code>%X</code> | Equivalent to <code>%T</code> .                 |
| 44317 | <code>%Z</code> | Implementation-defined.                         |

44318 **RATIONALE**

44319 None.

44320 **FUTURE DIRECTIONS**

44321 None.

44322 **SEE ALSO**

44323 *asctime()*, *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*, *time()*, *utime()*,  
44324 the Base Definitions volume of IEEE Std. 1003.1-200x, <**time.h**>

44325 **CHANGE HISTORY**

44326 First released in Issue 3.

44327 **Issue 4**

44328 The DESCRIPTION is expanded to describe modified conversion specifiers.

44329 %C, %e, %R, %u, and %V are added to the list of valid conversion specifications.

44330 The DESCRIPTION and RETURN VALUE sections are changed to make it clear when the  
44331 function uses byte values rather than (possibly multi-byte) character values.

44332 The following changes are incorporated for alignment with the ISO C standard:

- 44333 • The type of argument *format* is changed from **char\*** to **const char\***, and the type of argument  
44334 *tm\_ptr* is changed from **struct tm\*** to **const struct tm\***.
- 44335 • In the description of the %Z conversion specification, the words “or abbreviation” are added  
44336 to indicate that *strptime()* does not necessarily return a full timezone name.

44337 **Issue 5**

44338 The description of %OV is changed to be consistent with %V and defines Monday as the first  
44339 day of the week.

44340 The description of %Oy is clarified.

44341 **Issue 6**

44342 Extensions beyond the ISO C standard are now marked.

44343 The Open Group corrigenda item U033/8 has been applied. The %V conversion specifier is  
44344 changed from “Otherwise, it is week 53 of the previous year, and the next week is week 1” to  
44345 “Otherwise, it is the last week of the previous year, and the next week is week 1”.

44346 The following new requirements on POSIX implementations derive from alignment with the  
44347 Single UNIX Specification:

- 44348 • The %C, %D, %e, %h, %n, %r, %R, %t, and %T conversion specifiers are added.
- 44349 • The modified conversion specifiers are added for consistency with the ISO POSIX-2 standard  
44350 *date* utility.

44351 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 44352 • The *strptime()* prototype is updated.
- 44353 • The DESCRIPTION is extensively revised.



44354 **NAME**

44355            strlen — get string length

44356 **SYNOPSIS**

44357            #include &lt;string.h&gt;

44358            size\_t strlen(const char \*s);

44359 **DESCRIPTION**

44360 cx        The functionality described on this reference page is aligned with the ISO C standard. Any  
44361        conflict between the requirements described here and the ISO C standard is unintentional. This  
44362        volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

44363        The *strlen()* function shall compute the number of bytes in the string to which *s* points, not  
44364        including the terminating null byte.

44365 **RETURN VALUE**

44366        The *strlen()* function shall return the length of *s*; no return value shall be reserved to indicate an  
44367        error.

44368 **ERRORS**

44369        No errors are defined.

44370 **EXAMPLES**44371            **Getting String Lengths**

44372        The following example sets the maximum length of *key* and *data* by using *strlen()* to get the  
44373        lengths of those strings.

```
44374 #include <string.h>
44375 ...
44376 struct element {
44377 char *key;
44378 char *data;
44379 };
44380 ...
44381 char *key, *data;
44382 int len;
44383 *keylength = *datalength = 0;
44384 ...
44385 if ((len = strlen(key)) > *keylength)
44386 *keylength = len;
44387 if ((len = strlen(data)) > *datalength)
44388 *datalength = len;
44389 ...
```

44390 **APPLICATION USAGE**

44391        None.

44392 **RATIONALE**

44393        None.

44394 **FUTURE DIRECTIONS**

44395        None.

44396 **SEE ALSO**44397       The Base Definitions volume of IEEE Std. 1003.1-200x, <**string.h**>44398 **CHANGE HISTORY**

44399       First released in Issue 1. Derived from Issue 1 of the SVID.

44400 **Issue 4**44401       The DESCRIPTION is changed to make it clear that the function works in units of bytes rather  
44402       than (possibly multi-byte) characters.

44403       The following change is incorporated for alignment with the ISO C standard:

- 44404
- The type of argument *s* is changed from **char\*** to **const char\***.

44405 **Issue 5**44406       The RETURN VALUE section is updated to indicate that *strlen()* returns the length of *s*, and not  
44407       *s* itself as was previously stated.

44408 **NAME**

44409       strncasecmp — case-insensitive string comparison

44410 **SYNOPSIS**

44411 xSI       #include &lt;strings.h&gt;

44412       int strncasecmp(const char \*s1, const char \*s2, size\_t n);

44413

44414 **DESCRIPTION**44415       Refer to *strcasecmp()*.

44416 **NAME**

44417 strncat — concatenate a string with part of another

44418 **SYNOPSIS**

44419 #include &lt;string.h&gt;

44420 char \*strncat(char \*restrict *s1*, const char \*restrict *s2*, size\_t *n*);44421 **DESCRIPTION**

44422 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
44423 conflict between the requirements described here and the ISO C standard is unintentional. This  
44424 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

44425 The *strncat()* function shall append not more than *n* bytes (a null byte and bytes that follow it  
44426 are not appended) from the array pointed to by *s2* to the end of the string pointed to by *s1*. The  
44427 initial byte of *s2* overwrites the null byte at the end of *s1*. A terminating null byte is always  
44428 appended to the result. If copying takes place between objects that overlap, the behavior is  
44429 undefined.

44430 **RETURN VALUE**44431 The *strncat()* function shall return *s1*; no return value shall be reserved to indicate an error.44432 **ERRORS**

44433 No errors are defined.

44434 **EXAMPLES**

44435 None.

44436 **APPLICATION USAGE**

44437 None.

44438 **RATIONALE**

44439 None.

44440 **FUTURE DIRECTIONS**

44441 None.

44442 **SEE ALSO**44443 *strcat()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**string.h**>44444 **CHANGE HISTORY**

44445 First released in Issue 1. Derived from Issue 1 of the SVID.

44446 **Issue 4**

44447 The **DESCRIPTION** is changed to make it clear that the function manipulates bytes rather than  
44448 (possibly multi-byte) characters.

44449 The following change is incorporated for alignment with the ISO C standard:

- 44450 • The type of argument *s2* is changed from **char\*** to **const char\***.

44451 **Issue 6**44452 The *strncat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

44453 **NAME**

44454 strncmp — compare part of two strings

44455 **SYNOPSIS**

44456 #include &lt;string.h&gt;

44457 int strncmp(const char \*s1, const char \*s2, size\_t n);

44458 **DESCRIPTION**

44459 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
44460 conflict between the requirements described here and the ISO C standard is unintentional. This  
44461 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

44462 The *strncmp()* function shall compare not more than *n* bytes (bytes that follow a null byte are not  
44463 compared) from the array pointed to by *s1* to the array pointed to by *s2*.

44464 The sign of a non-zero return value is determined by the sign of the difference between the  
44465 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings  
44466 being compared.

44467 **RETURN VALUE**

44468 Upon successful completion, *strncmp()* shall return an integer greater than, equal to, or less than  
44469 0, if the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the  
44470 possibly null-terminated array pointed to by *s2* respectively.

44471 **ERRORS**

44472 No errors are defined.

44473 **EXAMPLES**

44474 None.

44475 **APPLICATION USAGE**

44476 None.

44477 **RATIONALE**

44478 None.

44479 **FUTURE DIRECTIONS**

44480 None.

44481 **SEE ALSO**44482 *strcmp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**string.h**>44483 **CHANGE HISTORY**

44484 First released in Issue 1. Derived from Issue 1 of the SVID.

44485 **Issue 4**

44486 The DESCRIPTION is changed to make it clear that the function manipulates bytes rather than  
44487 (possibly multi-byte) characters.

44488 The following change is incorporated for alignment with the ISO C standard:

- 44489 • The type of arguments *s1* and *s2* are changed from **char\*** to **const char\***.

44490 **Issue 6**

44491 Extensions beyond the ISO C standard are now marked.

44492 **NAME**

44493       strncpy — copy part of a string

44494 **SYNOPSIS**

44495       #include &lt;string.h&gt;

44496       char \*strncpy(char \*restrict *s1*, const char \*restrict *s2*, size\_t *n*);44497 **DESCRIPTION**

44498 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
44499 conflict between the requirements described here and the ISO C standard is unintentional. This  
44500 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

44501       The *strncpy()* function shall copy not more than *n* bytes (bytes that follow a null byte are not  
44502 copied) from the array pointed to by *s2* to the array pointed to by *s1*. If copying takes place  
44503 between objects that overlap, the behavior is undefined.

44504       If the array pointed to by *s2* is a string that is shorter than *n* bytes, null bytes shall be appended  
44505 to the copy in the array pointed to by *s1*, until *n* bytes in all are written.

44506 **RETURN VALUE**44507       The *strncpy()* function shall return *s1*; no return value is reserved to indicate an error.44508 **ERRORS**

44509       No errors are defined.

44510 **EXAMPLES**

44511       None.

44512 **APPLICATION USAGE**

44513       Character movement is performed differently in different implementations. Thus, overlapping  
44514 moves may yield surprises.

44515       If there is no null byte in the first *n* bytes of the array pointed to by *s2*, the result is not null-  
44516 terminated.

44517 **RATIONALE**

44518       None.

44519 **FUTURE DIRECTIONS**

44520       None.

44521 **SEE ALSO**44522       *strcpy()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <string.h>44523 **CHANGE HISTORY**

44524       First released in Issue 1. Derived from Issue 1 of the SVID.

44525 **Issue 4**

44526       The **DESCRIPTION** is changed to make it clear that the function manipulates bytes rather than  
44527 (possibly multi-byte) characters.

44528       The following change is incorporated for alignment with the ISO C standard:

- 44529       • The type of argument *s2* is changed from **char\*** to **const char\***.

44530 **Issue 6**44531       The *strncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

44532 **NAME**

44533 strpbrk — scan string for byte

44534 **SYNOPSIS**

44535 #include &lt;string.h&gt;

44536 char \*strpbrk(const char \*s1, const char \*s2);

44537 **DESCRIPTION**

44538 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
44539 conflict between the requirements described here and the ISO C standard is unintentional. This  
44540 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

44541 The *strpbrk()* function shall locate the first occurrence in the string pointed to by *s1* of any byte  
44542 from the string pointed to by *s2*.

44543 **RETURN VALUE**

44544 Upon successful completion, *strpbrk()* shall return a pointer to the byte or a null pointer if no  
44545 byte from *s2* occurs in *s1*.

44546 **ERRORS**

44547 No errors are defined.

44548 **EXAMPLES**

44549 None.

44550 **APPLICATION USAGE**

44551 None.

44552 **RATIONALE**

44553 None.

44554 **FUTURE DIRECTIONS**

44555 None.

44556 **SEE ALSO**44557 *strchr()*, *strchr()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <string.h>44558 **CHANGE HISTORY**

44559 First released in Issue 1. Derived from Issue 1 of the SVID.

44560 **Issue 4**

44561 The DESCRIPTION and RETURN VALUE sections are changed to make it clear that the function  
44562 works in units of bytes rather than (possibly multi-byte) characters.

44563 The following change is incorporated for alignment with the ISO C standard:

- 44564 • The type of arguments *s1* and *s2* is changed from **char\*** to **const char\***.

## 44565 NAME

44566 strptime — date and time conversion

## 44567 SYNOPSIS

44568 XSI 

```
#include <time.h>
```

```
44569 char *strptime(const char *restrict buf, const char *restrict format,
44570 struct tm *restrict tm);
44571
```

## 44572 DESCRIPTION

44573 The *strptime()* function shall convert the character string pointed to by *buf* to values which are  
 44574 stored in the **tm** structure pointed to by *tm*, using the format specified by *format*.

44575 The *format* is composed of zero or more directives. Each directive is composed of one of the  
 44576 following: one or more white-space characters (as specified by *isspace()*); an ordinary character  
 44577 (neither '%' nor a white-space character); or a conversion specification. Each conversion  
 44578 specification is composed of a '%' character followed by a conversion character which specifies  
 44579 the replacement required. The application shall ensure that there is white-space or other non-  
 44580 alphanumeric characters between any two conversion specifications. The following conversion  
 44581 specifications are supported:

|       |    |                                                                                                                                                                                               |
|-------|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 44582 | %a | The day of the week, using the locale's weekday names; either the abbreviated or full name may be specified.                                                                                  |
| 44583 |    |                                                                                                                                                                                               |
| 44584 | %A | The same as %a.                                                                                                                                                                               |
| 44585 | %b | The month, using the locale's month names; either the abbreviated or full name may be specified.                                                                                              |
| 44586 |    |                                                                                                                                                                                               |
| 44587 | %B | The same as %b.                                                                                                                                                                               |
| 44588 | %c | Replaced by the locale's appropriate date and time representation.                                                                                                                            |
| 44589 | %C | The century number [0,99]; leading zeros are permitted but not required.                                                                                                                      |
| 44590 | %d | The day of the month [1,31]; leading zeros are permitted but not required.                                                                                                                    |
| 44591 | %D | The date as %m/%d/%y.                                                                                                                                                                         |
| 44592 | %e | The same as %d.                                                                                                                                                                               |
| 44593 | %h | The same as %b.                                                                                                                                                                               |
| 44594 | %H | The hour (24-hour clock) [0,23]; leading zeros are permitted but not required.                                                                                                                |
| 44595 | %I | The hour (12-hour clock) [1,12]; leading zeros are permitted but not required.                                                                                                                |
| 44596 | %j | The day number of the year [1,366]; leading zeros are permitted but not required.                                                                                                             |
| 44597 | %m | The month number [1,12]; leading zeros are permitted but not required.                                                                                                                        |
| 44598 | %M | The minute [0-59]; leading zeros are permitted but not required.                                                                                                                              |
| 44599 | %n | Any white space.                                                                                                                                                                              |
| 44600 | %p | The locale's equivalent of a.m or p.m.                                                                                                                                                        |
| 44601 | %r | 12-hour clock time using the AM/PM notation if <b>t_fmt_ampm</b> is not an empty string in the LC_TIME portion of the current locale; in the POSIX locale, this is equivalent to %I:%M:%S %p. |
| 44602 |    |                                                                                                                                                                                               |
| 44603 |    |                                                                                                                                                                                               |
| 44604 | %R | The time as %H:%M.                                                                                                                                                                            |



|       |                 |                                                                                                                                                                                                                                                                                                              |
|-------|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 44605 | <code>%S</code> | The seconds [0,61]; leading zeros are permitted but not required.                                                                                                                                                                                                                                            |
| 44606 | <code>%t</code> | Any white space.                                                                                                                                                                                                                                                                                             |
| 44607 | <code>%T</code> | The time as <code>%H:%M:%S</code> .                                                                                                                                                                                                                                                                          |
| 44608 | <code>%U</code> | The week number of the year (Sunday as the first day of the week) as a decimal number [00,53]; leading zeros are permitted but not required.                                                                                                                                                                 |
| 44609 |                 |                                                                                                                                                                                                                                                                                                              |
| 44610 | <code>%w</code> | The weekday as a decimal number [0,6], with 0 representing Sunday; leading zeros are permitted but not required.                                                                                                                                                                                             |
| 44611 |                 |                                                                                                                                                                                                                                                                                                              |
| 44612 | <code>%W</code> | The week number of the year (Monday as the first day of the week) as a decimal number [00,53]; leading zeros are permitted but not required.                                                                                                                                                                 |
| 44613 |                 |                                                                                                                                                                                                                                                                                                              |
| 44614 | <code>%x</code> | The date, using the locale's date format.                                                                                                                                                                                                                                                                    |
| 44615 | <code>%X</code> | The time, using the locale's time format.                                                                                                                                                                                                                                                                    |
| 44616 | <code>%y</code> | The year within century. When a century is not otherwise specified, values in the range 69-99 refer to years in the twentieth century (1969 to 1999 inclusive); values in the range 00-68 refer to years in the twenty-first century (2000 to 2068 inclusive); leading zeros are permitted but not required. |
| 44617 |                 |                                                                                                                                                                                                                                                                                                              |
| 44618 |                 |                                                                                                                                                                                                                                                                                                              |
| 44619 |                 |                                                                                                                                                                                                                                                                                                              |
| 44620 | <code>%Y</code> | The year, including the century (for example, 1988).                                                                                                                                                                                                                                                         |
| 44621 | <code>%%</code> | Replaced by <code>'%'</code> .                                                                                                                                                                                                                                                                               |

#### 44622 **Modified Directives**

|       |                  |                                                                                                                                                                                                                                                                                                                                                                         |
|-------|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 44623 |                  | Some directives can be modified by the <i>E</i> and <i>O</i> modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified directive. If the alternative format or specification does not exist in the current locale, the behavior shall be as if the unmodified directive were used. |
| 44624 |                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 44625 |                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 44626 |                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 44627 | <code>%Ec</code> | The locale's alternative appropriate date and time representation.                                                                                                                                                                                                                                                                                                      |
| 44628 | <code>%EC</code> | The name of the base year (period) in the locale's alternative representation.                                                                                                                                                                                                                                                                                          |
| 44629 | <code>%Ex</code> | The locale's alternative date representation.                                                                                                                                                                                                                                                                                                                           |
| 44630 | <code>%EX</code> | The locale's alternative time representation.                                                                                                                                                                                                                                                                                                                           |
| 44631 | <code>%Ey</code> | The offset from <code>%EC</code> (year only) in the locale's alternative representation.                                                                                                                                                                                                                                                                                |
| 44632 | <code>%EY</code> | The full alternative year representation.                                                                                                                                                                                                                                                                                                                               |
| 44633 | <code>%Od</code> | The day of the month using the locale's alternative numeric symbols; leading zeros are permitted but not required.                                                                                                                                                                                                                                                      |
| 44634 |                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 44635 | <code>%Oe</code> | The same as <code>%Od</code> .                                                                                                                                                                                                                                                                                                                                          |
| 44636 | <code>%OH</code> | The hour (24-hour clock) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                |
| 44637 | <code>%OI</code> | The hour (12-hour clock) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                |
| 44638 | <code>%Om</code> | The month using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                               |
| 44639 | <code>%OM</code> | The minutes using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                             |
| 44640 | <code>%OS</code> | The seconds using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                             |
| 44641 | <code>%OU</code> | The week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                       |
| 44642 |                  |                                                                                                                                                                                                                                                                                                                                                                         |

- 44643        %Ow    The number of the weekday (Sunday=0) using the locale's alternative numeric symbols.
- 44644        %OW    The week number of the year (Monday as the first day of the week) using the locale's  
44645 alternative numeric symbols.
- 44646        %Oy    The year (offset from %C) using the locale's alternative numeric symbols.
- 44647        A directive composed of white-space characters is executed by scanning input up to the first  
44648 character that is not white-space (which remains unscanned), or until no more characters can be  
44649 scanned.
- 44650        A directive that is an ordinary character is executed by scanning the next character from the  
44651 buffer. If the character scanned from the buffer differs from the one comprising the directive, the  
44652 directive fails, and the differing and subsequent characters remain unscanned.
- 44653        A series of directives composed of %n, %t, white-space characters, or any combination is  
44654 executed by scanning up to the first character that is not white space (which remains  
44655 unscanned), or until no more characters can be scanned.
- 44656        Any other conversion specification is executed by scanning characters until a character matching  
44657 the next directive is scanned, or until no more characters can be scanned. These characters,  
44658 except the one matching the next directive, are then compared to the locale values associated  
44659 with the conversion specifier. If a match is found, values for the appropriate **tm** structure  
44660 members are set to values corresponding to the locale information. Case is ignored when  
44661 matching items in *buf* such as month or weekday names. If no match is found, *strptime()* fails  
44662 and no more characters are scanned.
- 44663 **RETURN VALUE**
- 44664        Upon successful completion, *strptime()* shall return a pointer to the character following the last  
44665 character parsed. Otherwise, a null pointer shall be returned.
- 44666 **ERRORS**
- 44667        No errors are defined.
- 44668 **EXAMPLES**
- 44669        None.
- 44670 **APPLICATION USAGE**
- 44671        Several “same as” formats and the special processing of white-space characters are provided in  
44672 order to ease the use of identical *format* strings for *strftime()* and *strptime()*.
- 44673        Applications should use %Y (4-digit years) in preference to %y (2-digit years).
- 44674        It is unspecified whether multiple calls to *strptime()* using the same **tm** structure will update the  
44675 current contents of the structure or overwrite all contents of the structure. Portable applications  
44676 should make a single call to *strptime()* with a format and all data needed to completely specify  
44677 the date and time being converted.
- 44678 **RATIONALE**
- 44679        None.
- 44680 **FUTURE DIRECTIONS**
- 44681        The *strptime()* function is expected to be mandatory in the next version of this volume of  
44682 IEEE Std. 1003.1-200x.
- 44683 **SEE ALSO**
- 44684        *scanf()*, *strftime()*, *time()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**time.h**>

44685 **CHANGE HISTORY**

44686 First released in Issue 4.

44687 **Issue 5**

44688 Moved from ENHANCED I18N to BASE.

44689 The [ENOSYS] error is removed.

44690 The exact meaning of the %y and %Oy specifiers are clarified in the DESCRIPTION.

44691 **Issue 6**

44692 The Open Group corrigenda item U033/5 has been applied. The %r specifier description is reworded.

44694 The DESCRIPTION is updated to avoid use of the term “must” for application requirements. |

44695 The **restrict** keyword is added to the *strptime()* prototype for alignment with the |  
44696 ISO/IEC 9899:1999 standard. |

44697 The Open Group corrigenda item U047/2 has been applied. |

44698 **NAME**

44699       strchr — string scanning operation

44700 **SYNOPSIS**

44701       #include &lt;string.h&gt;

44702       char \*strchr(const char \*s, int c);

44703 **DESCRIPTION**

44704 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
44705 conflict between the requirements described here and the ISO C standard is unintentional. This  
44706 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

44707 **CX**       The *strchr()* function shall locate the last occurrence of *c* (converted to an **unsigned char**) in the  
44708 string pointed to by *s*. The terminating null byte is considered to be part of the string.

44709 **RETURN VALUE**

44710       Upon successful completion, *strchr()* shall return a pointer to the byte or a null pointer if *c* does  
44711 not occur in the string.

44712 **ERRORS**

44713       No errors are defined.

44714 **EXAMPLES**44715       **Finding the Base Name of a File**

44716       The following example uses *strchr()* to get a pointer to the base name of a file. The *strchr()*  
44717 function searches backwards through the name of the file to find the last '/' character in *name*.  
44718 This pointer (plus one) will point to the base name of the file.

44719       #include &lt;string.h&gt;

44720       ...

44721       const char \*name;

44722       char \*basename;

44723       ...

44724       basename = strchr(name, '/') + 1;

44725       ...

44726 **APPLICATION USAGE**

44727       None.

44728 **RATIONALE**

44729       None.

44730 **FUTURE DIRECTIONS**

44731       None.

44732 **SEE ALSO**44733       *strchr()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <string.h>44734 **CHANGE HISTORY**

44735       First released in Issue 1. Derived from Issue 1 of the SVID.

44736 **Issue 4**

44737       The **DESCRIPTION** and **RETURN VALUE** sections are changed to make it clear that the function  
44738 works in units of bytes rather than (possibly multi-byte) characters.

44739       The following change is incorporated for alignment with the ISO C standard:

44740

- The type of argument *s* is changed from **char\*** to **const char\***.

44741 **NAME**

44742 strspn — get length of a substring

44743 **SYNOPSIS**

44744 #include &lt;string.h&gt;

44745 size\_t strspn(const char \*s1, const char \*s2);

44746 **DESCRIPTION**

44747 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
44748 conflict between the requirements described here and the ISO C standard is unintentional. This  
44749 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

44750 The *strspn()* function shall compute the length of the maximum initial segment of the string  
44751 pointed to by *s1* which consists entirely of bytes from the string pointed to by *s2*.

44752 **RETURN VALUE**

44753 The *strspn()* function shall return the length of *s1*; no return value is reserved to indicate an  
44754 error.

44755 **ERRORS**

44756 No errors are defined.

44757 **EXAMPLES**

44758 None.

44759 **APPLICATION USAGE**

44760 None.

44761 **RATIONALE**

44762 None.

44763 **FUTURE DIRECTIONS**

44764 None.

44765 **SEE ALSO**44766 *strcspn()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <string.h>44767 **CHANGE HISTORY**

44768 First released in Issue 1. Derived from Issue 1 of the SVID.

44769 **Issue 4**

44770 The DESCRIPTION is changed to make it clear that the function works in units of bytes rather  
44771 than (possibly multi-byte) characters.

44772 The following change is incorporated for alignment with the ISO C standard:

- 44773 • The type of arguments *s1* and *s2* are changed from **char\*** to **const char\***.

44774 **Issue 5**

44775 The RETURN VALUE section is updated to indicate that *strspn()* returns the length of *s*, and not  
44776 *s* itself as was previously stated.

44777 **NAME**

44778            strstr — find a substring

44779 **SYNOPSIS**

44780            #include &lt;string.h&gt;

44781            char \*strstr(const char \*s1, const char \*s2);

44782 **DESCRIPTION**

44783 cx        The functionality described on this reference page is aligned with the ISO C standard. Any  
44784        conflict between the requirements described here and the ISO C standard is unintentional. This  
44785        volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

44786        The *strstr()* function shall locate the first occurrence in the string pointed to by *s1* of the  
44787        sequence of bytes (excluding the terminating null byte) in the string pointed to by *s2*.

44788 **RETURN VALUE**

44789        Upon successful completion, *strstr()* shall return a pointer to the located string or a null pointer  
44790        if the string is not found.

44791        If *s2* points to a string with zero length, the function shall return *s1*.

44792 **ERRORS**

44793        No errors are defined.

44794 **EXAMPLES**

44795        None.

44796 **APPLICATION USAGE**

44797        None.

44798 **RATIONALE**

44799        None.

44800 **FUTURE DIRECTIONS**

44801        None.

44802 **SEE ALSO**44803        *strchr()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <string.h>44804 **CHANGE HISTORY**

44805        First released in Issue 3.

44806        Entry included for alignment with the ANSI C standard.

44807 **Issue 4**

44808        The DESCRIPTION is changed to make it clear that the function works in units of bytes rather  
44809        than (possibly multi-byte) characters.

44810        The following change is incorporated for alignment with the ISO C standard:

- 44811        • The type of arguments *s1* and *s2* are changed from **char\*** to **const char\***.

## 44812 NAME

44813 strtod, strtodf, strtold — convert string to a double-precision number

## 44814 SYNOPSIS

44815 #include <stdlib.h>

44816 double strtod(const char \*restrict *nptr*, char \*\*restrict *endptr*);

44817 float strtodf(const char \*restrict *nptr*, char \*\*restrict *endptr*);

44818 long double strtold(const char \*restrict *nptr*, char \*\*restrict *endptr*);

## 44819 DESCRIPTION

44820 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
44821 conflict between the requirements described here and the ISO C standard is unintentional. This  
44822 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

44823 These functions shall convert the initial portion of the string pointed to by *nptr* to **double**, **float**,  
44824 and **long double** representation, respectively. First, they decompose the input string into three  
44825 parts:

- 44826 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 44827 2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
- 44828 3. A final string of one or more unrecognized characters, including the terminating null byte  
44829 of the input string

44830 Then it attempts to convert the subject sequence to a floating-point number, and returns the  
44831 result.

44832 The expected form of the subject sequence is an optional plus or minus sign, then one of the  
44833 following:

- 44834 • A non-empty sequence of decimal digits optionally containing a radix character, then an  
44835 optional exponent part
- 44836 • A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix  
44837 character, then an optional binary exponent part
- 44838 • One of INF or INFINITY, ignoring case
- 44839 • One of NAN or NAN(*n-char-sequence<sub>opt</sub>*), ignoring case in the NAN part, where:

44840 n-char-sequence:

44841 digit

44842 nondigit

44843 n-char-sequence digit

44844 n-char-sequence nondigit

44845 The subject sequence is defined as the longest initial subsequence of the input string, starting  
44846 with the first non-white-space character, that is of the expected form. The subject sequence  
44847 contains no characters if the input string is not of the expected form.

44848 If the subject sequence has the expected form for a floating-point number, the sequence of  
44849 characters starting with the first digit or the decimal-point character (whichever occurs first) is  
44850 interpreted as a floating constant of the C language, except that the radix character is used in  
44851 place of a period, and that if neither an exponent part nor a radix character appears in a decimal  
44852 floating-point number, or if a binary exponent part does not appear in a hexadecimal floating-  
44853 point number, an exponent part of the appropriate type with value zero is assumed to follow the  
44854 last digit in the string. If the subject sequence begins with a minus sign, the sequence is  
44855 interpreted as negated. A character sequence INF or INFINITY is interpreted as an infinity, if



44856 representable in the return type, else like a floating constant that is too large for the range of the  
 44857 return type. A character sequence NAN or NAN(*n-char-sequence<sub>opt</sub>*), is interpreted as a quiet NaN, if  
 44858 supported in the return type, else like a subject sequence part that does not have the expected form; the  
 44859 meaning of the *n-char* sequences is implementation-defined. A pointer to the final string is stored  
 44860 in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

44861 If the subject sequence has the hexadecimal form and FLT\_RADIX is a power of 2, the value  
 44862 resulting from the conversion is correctly rounded.

44863 CX The radix character is defined in the program's locale (category *LC\_NUMERIC*). In the POSIX  
 44864 locale, or in a locale where the radix character is not defined, the radix character defaults to a  
 44865 period ('.').

44866 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
 44867 accepted.

44868 If the subject sequence is empty or does not have the expected form, no conversion is performed;  
 44869 the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null  
 44870 pointer.

44871 The *strtod()* function shall not change the setting of *errno* if successful.

44872 Because 0 is returned on error and is also a valid return on success, an application wishing to  
 44873 check for error situations should set *errno* to 0, then call *strtod()*, then check *errno*.

#### 44874 RETURN VALUE

44875 Upon successful completion, these functions shall return the converted value. If no conversion  
 44876 could be performed, 0 shall be returned, and *errno* may be set to [EINVAL].

44877 If the correct value is outside the range of representable values, HUGE\_VAL, HUGE\_VALF, or  
 44878 HUGE\_VALL shall be returned (according to the sign of the value), and *errno* shall be set to  
 44879 [ERANGE].

44880 If the correct value would cause an underflow, a value whose magnitude is no greater than the  
 44881 smallest normalized positive number in the return type shall be returned and *errno* set to  
 44882 [ERANGE].

#### 44883 ERRORS

44884 The *strtod()* function shall fail if:

44885 CX [ERANGE] The value to be returned would cause overflow or underflow.

44886 The *strtod()* function may fail if:

44887 CX [EINVAL] No conversion could be performed.

#### 44888 Notes to Reviewers

44889 *This section with side shading will not appear in the final copy. - Ed.*

44890 There is a query outstanding over the question of [EINVAL] being an allowable extension to  
 44891 C99. This error may be removed in a future draft.

44892 **EXAMPLES**

44893 None.

44894 **APPLICATION USAGE**

44895 If the subject sequence has the hexadecimal form and FLT\_RADIX is not a power of 2, the result  
 44896 should be one of the two numbers in the appropriate internal format that are adjacent to the  
 44897 hexadecimal floating source value, with the extra stipulation that the error should have a correct  
 44898 sign for the current rounding direction.

44899 If the subject sequence has the decimal form and at most DECIMAL\_DIG (defined in <float.h>)  
 44900 significant digits, the result should be correctly rounded. If the subject sequence *D* has the  
 44901 decimal form and more than DECIMAL\_DIG significant digits, consider the two bounding,  
 44902 adjacent decimal strings *L* and *U*, both having DECIMAL\_DIG significant digits, such that the  
 44903 values of *L*, *D*, and *U* satisfy " $L \leq D \leq U$ ". The result should be one of the (equal or  
 44904 adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current  
 44905 rounding direction, with the extra stipulation that the error with respect to *D* should have a  
 44906 correct sign for the current rounding direction.

44907 **RATIONALE**

44908 None.

44909 **FUTURE DIRECTIONS**

44910 None.

44911 **SEE ALSO**

44912 *isspace()*, *localeconv()*, *scanf()*, *setlocale()*, *strtol()*, the Base Definitions volume of  
 44913 IEEE Std. 1003.1-200x, <float.h>, <stdlib.h>, the Base Definitions volume of  
 44914 IEEE Std. 1003.1-200x, Chapter 7, Locale

44915 **CHANGE HISTORY**

44916 First released in Issue 1. Derived from Issue 1 of the SVID.

44917 **Issue 4**

44918 The DESCRIPTION is changed to make it clear when the function manipulates bytes and when  
 44919 it manipulates characters.

44920 The [EINVAL] error is added to the ERRORS section and marked as an extension.

44921 The following changes are incorporated for alignment with the ISO C standard:

- 44922 • The function is no longer marked as an extension.
- 44923 • The type of argument *str* is changed from **char\*** to **const char\***.
- 44924 • The name of the second argument is changed from *ptr* to *endptr*.
- 44925 • The precise conditions under which the [ERANGE] error can be set have been defined in the  
 44926 RETURN VALUE section.

44927 **Issue 5**44928 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.44929 **Issue 6**

44930 Extensions beyond the ISO C standard are now marked.

44931 The following new requirements on POSIX implementations derive from alignment with the  
 44932 Single UNIX Specification:

- 44933 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
 44934 added if no conversion could be performed.

44935 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 44936 • The *strtod()* function is updated.
- 44937 • The *strtof()* and *strtold()* functions are added.
- 44938 • The DESCRIPTION is extensively revised.

44939 **NAME**

44940 strtoimax, strtoumax — convert string to integer type

44941 **SYNOPSIS**

44942 #include <inttypes.h>

44943 intmax\_t strtoimax(const char \*restrict nptr, char \*\*restrict endptr,  
44944 int base);

44945 uintmax\_t strtoumax(const char \*restrict nptr, char \*\*restrict endptr,  
44946 int base);

44947 **DESCRIPTION**

44948 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
44949 conflict between the requirements described here and the ISO C standard is unintentional. This  
44950 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

44951 These functions shall be equivalent to the *strtol()*, *strtoll()*, *strtoul()*, and *strtoull()* functions,  
44952 except that the initial portion of the string shall be converted to **intmax\_t** and **uintmax\_t**  
44953 representation, respectively.

44954 **RETURN VALUE**

44955 These functions shall return the converted value, if any.

44956 If no conversion could be performed, zero shall be returned.

44957 If the correct value is outside the range of representable values, {INTMAX\_MAX},  
44958 {INTMAX\_MIN}, or {UINTMAX\_MAX} shall be returned (according to the return type and sign  
44959 of the value, if any), and *errno* shall be set to [ERANGE].

44960 **ERRORS**

44961 These functions shall fail if:

44962 [ERANGE] The value to be returned is not representable.

44963 These functions may fail if:

44964 [EINVAL] The value of *base* is not supported.

44965 **EXAMPLES**

44966 None.

44967 **APPLICATION USAGE**

44968 None.

44969 **RATIONALE**

44970 None.

44971 **FUTURE DIRECTIONS**

44972 None.

44973 **SEE ALSO**

44974 *strtol()*, *strtoul()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <inttypes.h>

44975 **CHANGE HISTORY**

44976 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

## 44977 NAME

44978 strtok, strtok\_r — split string into tokens

## 44979 SYNOPSIS

44980 #include &lt;string.h&gt;

44981 char \*strtok(char \*restrict *s1*, const char \*restrict *s2*);44982 TSF char \*strtok\_r(char \*restrict *s*, const char \*restrict *sep*,44983 char \*\*restrict *lasts*);

44984

## 44985 DESCRIPTION

44986 CX For *strtok()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

44989 A sequence of calls to *strtok()* breaks the string pointed to by *s1* into a sequence of tokens, each of which is delimited by a byte from the string pointed to by *s2*. The first call in the sequence has *s1* as its first argument, and is followed by calls with a null pointer as their first argument. The separator string pointed to by *s2* may be different from call to call.

44993 The first call in the sequence searches the string pointed to by *s1* for the first byte that is *not* contained in the current separator string pointed to by *s2*. If no such byte is found, then there are no tokens in the string pointed to by *s1* and *strtok()* returns a null pointer. If such a byte is found, it is the start of the first token.

44997 The *strtok()* function then searches from there for a byte that *is* contained in the current separator string. If no such byte is found, the current token extends to the end of the string pointed to by *s1*, and subsequent searches for a token shall return a null pointer. If such a byte is found, it is overwritten by a null byte, which terminates the current token. The *strtok()* function saves a pointer to the following byte, from which the next search for a token shall start.

45002 Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

45004 The implementation shall behave as if no function defined in this volume of IEEE Std. 1003.1-200x calls *strtok()*.

45006 CX The *strtok()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

45008 TSF The *strtok\_r()* function considers the null-terminated string *s* as a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *sep*. The argument *lasts* points to a user-provided pointer which points to stored information necessary for *strtok\_r()* to continue scanning the same string.

45012 In the first call to *strtok\_r()*, *s* points to a null-terminated string, *sep* to a null-terminated string of separator characters, and the value pointed to by *lasts* is ignored. The *strtok\_r()* function returns a pointer to the first character of the first token, writes a null character into *s* immediately following the returned token, and updates the pointer to which *lasts* points.

45016 In subsequent calls, *s* is a NULL pointer and *lasts* shall be unchanged from the previous call so that subsequent calls shall move through the string *s*, returning successive tokens until no tokens remain. The separator string *sep* may be different from call to call. When no token remains in *s*, a NULL pointer is returned.

## 45020 RETURN VALUE

45021 Upon successful completion, *strtok()* shall return a pointer to the first byte of a token. Otherwise,  
45022 if there is no token, *strtok()* shall return a null pointer.

45023 TSF The *strtok\_r()* function shall return a pointer to the token found, or a NULL pointer when no  
45024 token is found.

## 45025 ERRORS

45026 No errors are defined.

## 45027 EXAMPLES

## 45028 Searching for Word Separators

45029 The following example searches for tokens separated by space characters.

```
45030 #include <string.h>
45031 ...
45032 char *token;
45033 char *line = "LINE TO BE SEPARATED";
45034 char *search = " ";

45035 /* Token will point to "LINE". */
45036 token = strtok(line, search);

45037 /* Token will point to "TO". */
45038 token = strtok(NULL, search);
```

## 45039 Breaking a Line

45040 The following example uses *strtok()* to break a line into two character strings separated by any  
45041 combination of <space>s, <tab>s, or <newline>s.

```
45042 #include <string.h>
45043 ...
45044 struct element {
45045 char *key;
45046 char *data;
45047 };
45048 ...
45049 char line[LINE_MAX];
45050 char *key, *data;
45051 ...
45052 key = strtok(line, " \n");
45053 data = strtok(NULL, " \n");
45054 ...
```

## 45055 APPLICATION USAGE

45056 The *strtok\_r()* function is thread-safe and stores its state in a user-supplied buffer instead of  
45057 possibly using a static data area that may be overwritten by an unrelated call from another  
45058 thread.

## 45059 RATIONALE

45060 The *strtok()* function searches for a separator string within a larger string. It returns a pointer to  
45061 the last substring between separator strings. This function uses static storage to keep track of  
45062 the current string position between calls. The new function, *strtok\_r()*, takes an additional  
45063 argument, *lasts*, to keep track of the current position in the string.

45064 **FUTURE DIRECTIONS**

45065 None.

45066 **SEE ALSO**

45067 The Base Definitions volume of IEEE Std. 1003.1-200x, &lt;string.h&gt;

45068 **CHANGE HISTORY**

45069 First released in Issue 1. Derived from Issue 1 of the SVID.

45070 **Issue 4**45071 The DESCRIPTION is changed to make it clear that the function manipulates bytes rather than  
45072 (possibly multi-byte) characters.

45073 The following changes are incorporated for alignment with the ISO C standard:

- 45074
- The function is no longer marked as an extension.
  - The type of argument *s2* is changed from **char\*** to **const char\***.
- 45075

45076 **Issue 5**45077 The *strtok\_r()* function is included for alignment with the POSIX Threads Extension.45078 A note indicating that the *strtok()* function need not be reentrant is added to the DESCRIPTION.45079 **Issue 6**

45080 Extensions beyond the ISO C standard are now marked.

45081 The *strtok\_r()* function is marked as part of the Thread-Safe Functions option.

45082 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

45083 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
45084 its avoidance of possibly using a static data area.45085 The **restrict** keyword is added to the *strtok()* and *strtok\_r()* prototypes for alignment with the  
45086 ISO/IEC 9899:1999 standard.

## 45087 NAME

45088 strtol, strtoll — convert string to a long integer

## 45089 SYNOPSIS

45090 #include &lt;stdlib.h&gt;

45091 long strtol(const char \*restrict *str*, char \*\*restrict *endptr*, int *base*);45092 long long strtoll(const char \*restrict *str*, char \*\*restrict *endptr*,45093 int *base*)

## 45094 DESCRIPTION

45095 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 45096 conflict between the requirements described here and the ISO C standard is unintentional. This  
 45097 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

45098 These functions shall convert the initial portion of the string pointed to by *str* to a type **long** and  
 45099 **long long** representation, respectively. First, they decompose the input string into three parts:

- 45100 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 45101 2. A subject sequence interpreted as an integer represented in some radix determined by the  
 45102 value of *base*
- 45103 3. A final string of one or more unrecognized characters, including the terminating null byte  
 45104 of the input string.

45105 Then it attempts to convert the subject sequence to an integer, and returns the result.

45106 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,  
 45107 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A  
 45108 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An  
 45109 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to  
 45110 '7' only. A hexadecimal constant consists of the prefix "0x" or "0X" followed by a sequence of  
 45111 the decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

45112 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
 45113 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
 45114 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the  
 45115 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the  
 45116 value of *base* is 16, the characters "0x" or "0X" may optionally precede the sequence of letters  
 45117 and digits, following the sign if present.

45118 The subject sequence is defined as the longest initial subsequence of the input string, starting  
 45119 with the first non-white-space character that is of the expected form. The subject sequence  
 45120 contains no characters if the input string is empty or consists entirely of white-space characters,  
 45121 or if the first non-white-space character is other than a sign or a permissible letter or digit.

45122 If the subject sequence has the expected form and the value of *base* is 0, the sequence of  
 45123 characters starting with the first digit is interpreted as an integer constant. If the subject  
 45124 sequence has the expected form and the value of *base* is between 2 and 36, it is used as the base  
 45125 for conversion, ascribing to each letter its value as given above. If the subject sequence begins  
 45126 with a minus sign, the value resulting from the conversion is negated. A pointer to the final  
 45127 string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

45128 cx In other than the C or POSIX locales, other implementation-defined subject sequences may be  
 45129 accepted.

45130 If the subject sequence is empty or does not have the expected form, no conversion is performed;  
 45131 the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null



- 45132 pointer.
- 45133 The *strtol()* function shall not change the setting of *errno* if successful.
- 45134 Because 0, {LONG\_MIN} or {LLONG\_MIN}, and {LONG\_MAX} or {LLONG\_MAX} are returned  
45135 on error and are also valid returns on success, an application wishing to check for error  
45136 situations should set *errno* to 0, then call *strtol()*, then check *errno*.
- 45137 **RETURN VALUE**
- 45138 Upon successful completion, these functions shall return the converted value, if any. If no  
45139 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].
- 45140 If the correct value is outside the range of representable values, {LONG\_MIN}, {LONG\_MAX},  
45141 {LLONG\_MIN} or {LLONG\_MAX} shall be returned (according to the sign of the value), and  
45142 *errno* set to [ERANGE].
- 45143 **ERRORS**
- 45144 The *strtol()* function shall fail if:
- 45145 [ERANGE] The value to be returned is not representable.
- 45146 The *strtol()* function may fail if:
- 45147 CX [EINVAL] The value of *base* is not supported.
- 45148 **EXAMPLES**
- 45149 None.
- 45150 **APPLICATION USAGE**
- 45151 None.
- 45152 **RATIONALE**
- 45153 None.
- 45154 **FUTURE DIRECTIONS**
- 45155 None.
- 45156 **SEE ALSO**
- 45157 *isalpha()*, *scanf()*, *strtod()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>
- 45158 **CHANGE HISTORY**
- 45159 First released in Issue 1. Derived from Issue 1 of the SVID.
- 45160 **Issue 4**
- 45161 The DESCRIPTION is changed to make it clear when the function manipulates bytes and when  
45162 it manipulates characters.
- 45163 In the RETURN VALUE section, text indicating that *errno* is set when 0 is returned is marked as  
45164 an extension.
- 45165 The ERRORS section is updated in line with the RETURN VALUE section.
- 45166 The following changes are incorporated for alignment with the ISO C standard:
- 45167 • The function is no longer marked as an extension.
  - 45168 • The type of argument *str* is changed from **char\*** to **const char\***.
  - 45169 • The name of the second argument is changed from *ptr* to *endptr*.
  - 45170 • The DESCRIPTION is changed to indicate permitted forms of the subject sequence when *base*  
45171 is 0.

- 45172           • The RETURN VALUE section is changed to indicate that {LONG\_MAX} or {LONG\_MIN} is
- 45173            returned if the converted value is too large or too small.
  
- 45174 **Issue 5**
- 45175           The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
  
- 45176 **Issue 6**
- 45177           Extensions beyond the ISO C standard are now marked.
  
- 45178           The following new requirements on POSIX implementations derive from alignment with the
- 45179           Single UNIX Specification:
  
- 45180           • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
- 45181            added if no conversion could be performed.
  
- 45182           The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
  
- 45183           • The *strtol()* prototype is updated.
  
- 45184           • The *strtoll()* function is added.

## 45185 NAME

45186 strtoul, strtoull — convert string to an unsigned long

## 45187 SYNOPSIS

45188 #include &lt;stdlib.h&gt;

45189 long strtoul(const char \*restrict *str*, char \*\*restrict *endptr*, int *base*);45190 long long strtoull(const char\*restrict *str*, char \*\*restrict *endptr*,45191 int *base*);

## 45192 DESCRIPTION

45193 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 45194 conflict between the requirements described here and the ISO C standard is unintentional. This  
 45195 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

45196 These functions shall convert the initial portion of the string pointed to by *str* to a type **long** and  
 45197 **long long** representation, respectively. First, they decompose the input string into three parts:

- 45198 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 45199 2. A subject sequence interpreted as an integer represented in some radix determined by the  
 45200 value of *base*
- 45201 3. A final string of one or more unrecognized characters, including the terminating null byte  
 45202 of the input string

45203 Then it attempts to convert the subject sequence to an unsigned integer, and returns the result.

45204 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,  
 45205 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A  
 45206 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An  
 45207 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to  
 45208 '7' only. A hexadecimal constant consists of the prefix "0x" or "0X" followed by a sequence of  
 45209 the decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

45210 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
 45211 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
 45212 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the  
 45213 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the  
 45214 value of *base* is 16, the characters "0x" or "0X" may optionally precede the sequence of letters  
 45215 and digits, following the sign if present.

45216 The subject sequence is defined as the longest initial subsequence of the input string, starting  
 45217 with the first non-white-space character that is of the expected form. The subject sequence  
 45218 contains no characters if the input string is empty or consists entirely of white-space characters,  
 45219 or if the first non-white-space character is other than a sign or a permissible letter or digit.

45220 If the subject sequence has the expected form and the value of *base* is 0, the sequence of  
 45221 characters starting with the first digit is interpreted as an integer constant. If the subject  
 45222 sequence has the expected form and the value of *base* is between 2 and 36, it is used as the base  
 45223 for conversion, ascribing to each letter its value as given above. If the subject sequence begins  
 45224 with a minus sign, the value resulting from the conversion is negated. A pointer to the final  
 45225 string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

45226 cx In other than the C or POSIX locales, other implementation-defined subject sequences may be  
 45227 accepted.

45228 If the subject sequence is empty or does not have the expected form, no conversion is performed;  
 45229 the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null

45230 pointer.

45231 The *strtol()* function shall not change the setting of *errno* if successful.

45232 Because 0, {ULONG\_MAX}, and {ULLONG\_MAX} are returned on error and are also valid  
 45233 returns on success, an application wishing to check for error situations should set *errno* to 0, then  
 45234 call *strtol()*, then check *errno*.

#### 45235 RETURN VALUE

45236 Upon successful completion, these functions shall return the converted value, if any. If no  
 45237 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL]. If the  
 45238 correct value is outside the range of representable values, {ULONG\_MAX} or {ULLONG\_MAX}  
 45239 shall be returned and *errno* set to [ERANGE].

#### 45240 ERRORS

45241 The *strtol()* function shall fail if:

45242 CX [EINVAL] The value of *base* is not supported.

45243 [ERANGE] The value to be returned is not representable.

45244 The *strtol()* function may fail if:

45245 CX [EINVAL] No conversion could be performed.

#### 45246 EXAMPLES

45247 None.

#### 45248 APPLICATION USAGE

45249 None.

#### 45250 RATIONALE

45251 None.

#### 45252 FUTURE DIRECTIONS

45253 None.

#### 45254 SEE ALSO

45255 *isalpha()*, *scanf()*, *strtod()*, *strtol()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
 45256 <stdlib.h>

#### 45257 CHANGE HISTORY

45258 First released in Issue 4. Derived from the ANSI C standard.

#### 45259 Issue 5

45260 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

#### 45261 Issue 6

45262 Extensions beyond the ISO C standard are now marked.

45263 The following new requirements on POSIX implementations derive from alignment with the  
 45264 Single UNIX Specification:

- 45265 • The [EINVAL] error condition is added for when the value of *base* is not supported.

45266 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
 45267 added if no conversion could be performed.

45268 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 45269 • The *strtol()* prototype is updated.

45270

- The *strtoull()* function is added.

45271 **NAME**

45272 strxfrm — string transformation

45273 **SYNOPSIS**

45274 #include &lt;string.h&gt;

45275 size\_t strxfrm(char \*restrict *s1*, const char \*restrict *s2*, size\_t *n*);45276 **DESCRIPTION**

45277 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
45278 conflict between the requirements described here and the ISO C standard is unintentional. This  
45279 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

45280 The *strxfrm()* function shall transform the string pointed to by *s2* and place the resulting string  
45281 into the array pointed to by *s1*. The transformation is such that if *strcmp()* is applied to two  
45282 transformed strings, it returns a value greater than, equal to, or less than 0, corresponding to the  
45283 result of *strcoll()* applied to the same two original strings. No more than *n* bytes are placed into  
45284 the resulting array pointed to by *s1*, including the terminating null byte. If *n* is 0, *s1* is permitted  
45285 to be a null pointer. If copying takes place between objects that overlap, the behavior is  
45286 undefined.

45287 CX The *strxfrm()* function shall not change the setting of *errno* if successful.

45288 Because no return value is reserved to indicate an error, an application wishing to check for error  
45289 situations should set *errno* to 0, then call *strcoll()*, then check *errno*.

45290 **RETURN VALUE**

45291 Upon successful completion, *strxfrm()* shall return the length of the transformed string (not  
45292 including the terminating null byte). If the value returned is *n* or more, the contents of the array  
45293 pointed to by *s1* are indeterminate.

45294 CX On error, *strxfrm()* may set *errno* but no return value is reserved to indicate an error.

45295 **ERRORS**

45296 The *strxfrm()* function may fail if:

45297 CX [EINVAL] The string pointed to by the *s2* argument contains characters outside the  
45298 domain of the collating sequence.

45299 **EXAMPLES**

45300 None.

45301 **APPLICATION USAGE**

45302 The transformation function is such that two transformed strings can be ordered by *strcmp()* as  
45303 appropriate to collating sequence information in the program's locale (category *LC\_COLLATE*).

45304 The fact that when *n* is 0 *s1* is permitted to be a null pointer is useful to determine the size of the  
45305 *s1* array prior to making the transformation.

45306 **RATIONALE**

45307 None.

45308 **FUTURE DIRECTIONS**

45309 None.

45310 **SEE ALSO**

45311 *strcmp()*, *strcoll()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <string.h>

45312 **CHANGE HISTORY**

45313 First released in Issue 3.

45314 Entry included for alignment with the ISO C standard.

45315 **Issue 4**45316 The DESCRIPTION is changed to make it clear when the function manipulates byte values and  
45317 when it manipulates characters.45318 The sentence describing error returns in the RETURN VALUE section is marked as an extension,  
45319 as is the [EINVAL] error.

45320 The APPLICATION USAGE section is expanded.

45321 The following changes are incorporated for alignment with the ISO C standard:

- 45322
- The function is no longer marked as an extension.
- 45323
- The type of argument *s2* is changed from **char\*** to **const char\***.

45324 **Issue 5**45325 The DESCRIPTION is updated to indicate that *errno* does not change if the function is  
45326 successful.45327 **Issue 6**

45328 Extensions beyond the ISO C standard are now marked.

45329 The following new requirements on POSIX implementations derive from alignment with the  
45330 Single UNIX Specification:

- 45331
- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
45332 added if no conversion could be performed.

45333 The *strxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard. |

45334 **NAME**

45335 swab — swap bytes

45336 **SYNOPSIS**45337 XSI `#include <unistd.h>`45338 `void swab(const void *restrict src, void *restrict dest,`  
45339 `ssize_t nbytes);`

45340

45341 **DESCRIPTION**

45342 The *swab()* function shall copy *nbytes* bytes, which are pointed to by *src*, to the object pointed to  
45343 by *dest*, exchanging adjacent bytes. The *nbytes* argument should be even. If *nbytes* is odd, *swab()*  
45344 copies and exchanges *nbytes*–1 bytes and the disposition of the last byte is unspecified. If  
45345 copying takes place between objects that overlap, the behavior is undefined. If *nbytes* is  
45346 negative, *swab()* does nothing.

45347 **RETURN VALUE**

45348 None.

45349 **ERRORS**

45350 No errors are defined.

45351 **EXAMPLES**

45352 None.

45353 **APPLICATION USAGE**

45354 None.

45355 **RATIONALE**

45356 None.

45357 **FUTURE DIRECTIONS**

45358 None.

45359 **SEE ALSO**45360 The Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>45361 **CHANGE HISTORY**

45362 First released in Issue 1. Derived from Issue 1 of the SVID.

45363 **Issue 4**45364 The <**unistd.h**> header is added to the SYNOPSIS section.

45365 The type of argument *src* is changed from **char\*** to **const void\***, *dest* is changed from **char\*** to  
45366 **void\***, and *nbytes* is changed from **int** to **ssize\_t**.

45367 The DESCRIPTION is changed as follows:

- 45368 • States explicitly that copying between overlapping objects results in undefined behavior.
- 45369 • Takes account of the type change to *nbyte*; that is, previously it was defined as **int** and could  
45370 be positive or negative, whereas now it is defined as an **unsigned** type.
- 45371 • A statement about overlapping objects is added.

45372 The APPLICATION USAGE section is removed.



45373 **Issue 6**

45374 The **restrict** keyword is added to the *swab()* prototype for alignment with the  
45375 ISO/IEC 9899:1999 standard.

45376 **NAME**

45377 swapcontext — swap user context

45378 **SYNOPSIS**

45379 XSI `#include <ucontext.h>`

45380 `int swapcontext(ucontext_t *restrict oucp,`  
45381 `const ucontext_t *restrict ucp);`

45382

45383 **DESCRIPTION**

45384 Refer to *makecontext()*.

45385 **NAME**

45386 swprintf — print formatted wide-character output

45387 **SYNOPSIS**

45388 #include &lt;stdio.h&gt;

45389 #include &lt;wchar.h&gt;

45390 int swprintf(wchar\_t \*ws, size\_t n, const wchar\_t \*format, ...);

45391 **DESCRIPTION**45392 Refer to *fwprintf()*.

45393 **NAME**45394 `swscanf` — convert formatted wide-character input45395 **SYNOPSIS**45396 `#include <stdio.h>`45397 `#include <wchar.h>`45398 `int swscanf(const wchar_t *ws, const wchar_t *format, ... );` |45399 **DESCRIPTION** |45400 Refer to *fwscanf()*.

45401 **NAME**

45402            symlink — make symbolic link to a file

45403 **SYNOPSIS**

45404            #include &lt;unistd.h&gt;

45405            int symlink(const char \*path1, const char \*path2);

45406 **DESCRIPTION**

45407            The *symlink()* function shall create a symbolic link called *path2* that contains the string pointed  
 45408            to by *path1* (*path2* is the name of the symbolic link created, *path1* is the string contained in the  
 45409            symbolic link).

45410            The string pointed to by *path1* shall be treated only as a character string and shall not be  
 45411            validated as a path name.

45412            If the *symlink()* function fails for any reason other than [EIO], any file named by *path2* shall be  
 45413            unaffected.

45414 **RETURN VALUE**

45415            Upon successful completion, *symlink()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 45416            indicate the error.

45417 **ERRORS**45418            The *symlink()* function shall fail if:

45419            [EACCES]            Write permission is denied in the directory where the symbolic link is being  
 45420            created, or search permission is denied for a component of the path prefix of  
 45421            *path2*.

45422            [EEXIST]            The *path2* argument names an existing file or symbolic link.

45423            [EIO]                An I/O error occurs while reading from or writing to the file system.

45424            [ELOOP]            A loop exists in symbolic links encountered during resolution of the *path2*  
 45425            argument.

45426            [ENAMETOOLONG]

45427            The length of the *path2* argument exceeds {PATH\_MAX} or a path name  
 45428            component is longer than {NAME\_MAX} or the length of the *path1* argument  
 45429            is longer than {SYMLINK\_MAX}.

45430            [ENOENT]            A component of *path2* does not name an existing file or *path2* is an empty  
 45431            string.

45432            [ENOSPC]            The directory in which the entry for the new symbolic link is being placed  
 45433            cannot be extended because no space is left on the file system containing the  
 45434            directory, or the new symbolic link cannot be created because no space is left  
 45435            on the file system which shall contain the link, or the file system is out of file-  
 45436            allocation resources.

45437            [ENOTDIR]           A component of the path prefix of *path2* is not a directory.

45438            [EROFS]             The new symbolic link would reside on a read-only file system.

45439            The *symlink()* function may fail if:

45440            [ELOOP]            More than {SYMLOOP\_MAX} symbolic links were encountered during  
 45441            resolution of the *path2* argument.

45442            [ENAMETOOLONG]

45443            As a result of encountering a symbolic link in resolution of the *path2*

45444 argument, the length of the substituted path name string exceeded  
45445 {PATH\_MAX} bytes (including the terminating null byte), or the length of the  
45446 string pointed to by *path1* exceeded {SYMLINK\_MAX}.

**45447 EXAMPLES**

45448 None.

**45449 APPLICATION USAGE**

45450 Like a hard link, a symbolic link allows a file to have multiple logical names. The presence of a  
45451 hard link guarantees the existence of a file, even after the original name has been removed. A  
45452 symbolic link provides no such assurance; in fact, the file named by the *path1* argument need not  
45453 exist when the link is created. A symbolic link can cross file system boundaries.

45454 Normal permission checks are made on each component of the symbolic link path name during  
45455 its resolution.

**45456 RATIONALE**

45457 Since IEEE Std. 1003.1-200x does not require any association of file times with symbolic links,  
45458 there is no requirement that file times be updated by *symlink()*.

**45459 FUTURE DIRECTIONS**

45460 None.

**45461 SEE ALSO**

45462 *lchown()*, *link()*, *lstat()*, *open()*, *readlink()*, *unlink()*, the Base Definitions volume of  
45463 IEEE Std. 1003.1-200x, <**unistd.h**>

**45464 CHANGE HISTORY**

45465 First released in Issue 4, Version 2.

**45466 Issue 5**

45467 Moved from X/OPEN UNIX extension to BASE.

**45468 Issue 6**

45469 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 45470 • The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.  
45471 This is since behavior may vary from one file system to another.

45472 The following changes were made to align with the IEEE P1003.1a draft standard:

- 45473 • The DESCRIPTION text is updated.
- 45474 • The [ELOOP] optional error condition is added.

45475 **NAME**

45476 sync — schedule file system updates

45477 **SYNOPSIS**

45478 XSI #include &lt;unistd.h&gt;

45479 void sync(void);

45480

45481 **DESCRIPTION**45482 The *sync()* function shall cause all information in memory that updates file systems to be  
45483 scheduled for writing out to all file systems.45484 The writing, although scheduled, is not necessarily complete upon return from *sync()*.45485 **RETURN VALUE**45486 The *sync()* function shall return no value.45487 **ERRORS**

45488 No errors are defined.

45489 **EXAMPLES**

45490 None.

45491 **APPLICATION USAGE**

45492 None.

45493 **RATIONALE**

45494 None.

45495 **FUTURE DIRECTIONS**

45496 None.

45497 **SEE ALSO**45498 *fsync()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <unistd.h>45499 **CHANGE HISTORY**

45500 First released in Issue 4, Version 2.

45501 **Issue 5**

45502 Moved from X/OPEN UNIX extension to BASE.

45503 **NAME**

45504 sysconf — get configurable system variables

45505 **SYNOPSIS**

45506 #include <unistd.h>

45507 long sysconf(int name);

45508 **DESCRIPTION**

45509 The *sysconf()* function provides a method for the application to determine the current value of a  
 45510 configurable system limit or option (*variable*). Support for some system variables is dependent  
 45511 on implementation options (as indicated by the margin codes in the following table). Where an  
 45512 implementation option is not supported, the variable need not be supported.

45513 The *name* argument represents the system variable to be queried. The following table lists the  
 45514 minimal set of system variables from <limits.h> or <unistd.h> that can be returned by *sysconf()*,  
 45515 and the symbolic constants, defined in <unistd.h> that are the corresponding values used for  
 45516 *name*. Support for some configuration variables is dependent on implementation options (see  
 45517 shading and margin codes in the table below). Where an implementation option is not  
 45518 supported, the variable need not be supported.

45519

45520

45521 AIO

45522

45523

45524

45525 XSI

45526

45527

45528

45529

45530

45531

45532

45533 XSI

45534

45535 XSI

45536

45537

45538

45539 TSF

45540

45541

45542

45543 MSG

45544

45545

45546 ADV

45547 BAR

45548 AIO

|  | Variable                                                                    | Value of Name          |
|--|-----------------------------------------------------------------------------|------------------------|
|  | {AIO_LISTIO_MAX}                                                            | _SC_AIO_LISTIO_MAX     |
|  | {AIO_MAX}                                                                   | _SC_AIO_MAX            |
|  | {AIO_PRIO_DELTA_MAX}                                                        | _SC_AIO_PRIO_DELTA_MAX |
|  | {ARG_MAX}                                                                   | _SC_ARG_MAX            |
|  | {ATEXIT_MAX}                                                                | _SC_ATEXIT_MAX         |
|  | {BC_BASE_MAX}                                                               | _SC_BC_BASE_MAX        |
|  | {BC_DIM_MAX}                                                                | _SC_BC_DIM_MAX         |
|  | {BC_SCALE_MAX}                                                              | _SC_BC_SCALE_MAX       |
|  | {BC_STRING_MAX}                                                             | _SC_BC_STRING_MAX      |
|  | {CHILD_MAX}                                                                 | _SC_CHILD_MAX          |
|  | Clock ticks/second                                                          | _SC_CLK_TCK            |
|  | {COLL_WEIGHTS_MAX}                                                          | _SC_COLL_WEIGHTS_MAX   |
|  | {DELAYTIMER_MAX}                                                            | _SC_DELAYTIMER_MAX     |
|  | {EXPR_NEST_MAX}                                                             | _SC_EXPR_NEST_MAX      |
|  | {IOV_MAX}                                                                   | _SC_IOV_MAX            |
|  | {LINE_MAX}                                                                  | _SC_LINE_MAX           |
|  | {LOGIN_NAME_MAX}                                                            | _SC_LOGIN_NAME_MAX     |
|  | {NGROUPS_MAX}                                                               | _SC_NGROUPS_MAX        |
|  | Maximum size of <i>getgrgid_r()</i> and<br><i>getgrnam_r()</i> data buffers | _SC_GETGR_R_SIZE_MAX   |
|  | Maximum size of <i>getpwuid_r()</i> and<br><i>getpwnam_r()</i> data buffers | _SC_GETPW_R_SIZE_MAX   |
|  | {MQ_OPEN_MAX}                                                               | _SC_MQ_OPEN_MAX        |
|  | {MQ_PRIO_MAX}                                                               | _SC_MQ_PRIO_MAX        |
|  | {OPEN_MAX}                                                                  | _SC_OPEN_MAX           |
|  | _POSIX_ADVISORY_INFO                                                        | _SC_ADVISORY_INFO      |
|  | _POSIX_BARRIERS                                                             | _SC_BARRIERS           |
|  | _POSIX_ASYNCHRONOUS_IO                                                      | _SC_ASYNCHRONOUS_IO    |



|           | Variable                     | Value of Name             |
|-----------|------------------------------|---------------------------|
| 45549     |                              |                           |
| 45550     |                              |                           |
| 45551     | _POSIX_BASE                  | _SC_BASE                  |
| 45552     | _POSIX_C_LANG_SUPPORT        | _SC_C_LANG_SUPPORT        |
| 45553     | _POSIX_C_LANG_SUPPORT_R      | _SC_C_LANG_SUPPORT_R      |
| 45554 CS  | _POSIX_CLOCK_SELECTION       | _SC_CLOCK_SELECTION       |
| 45555 CPT | _POSIX_CPUTIME               | _SC_CPUTIME               |
| 45556     | _POSIX_DEVICE_IO             | _SC_DEVICE_IO             |
| 45557     | _POSIX_DEVICE_SPECIFIC       | _SC_DEVICE_SPECIFIC       |
| 45558     | _POSIX_DEVICE_SPECIFIC_R     | _SC_DEVICE_SPECIFIC_R     |
| 45559     | _POSIX_FD_MGMT               | _SC_FD_MGMT               |
| 45560     | _POSIX_FIFO                  | _SC_FIFO                  |
| 45561     | _POSIX_FILE_ATTRIBUTES       | _SC_FILE_ATTRIBUTES       |
| 45562     | _POSIX_FILE_LOCKING          | _SC_FILE_LOCKING          |
| 45563     | _POSIX_FILE_SYSTEM           | _SC_FILE_SYSTEM           |
| 45564 FSC | _POSIX_FSYNC                 | _SC_FSYNC                 |
| 45565     | _POSIX_JOB_CONTROL           | _SC_JOB_CONTROL           |
| 45566 MF  | _POSIX_MAPPED_FILES          | _SC_MAPPED_FILES          |
| 45567 ML  | _POSIX_MEMLOCK               | _SC_MEMLOCK               |
| 45568 MLR | _POSIX_MEMLOCK_RANGE         | _SC_MEMLOCK_RANGE         |
| 45569 MPR | _POSIX_MEMORY_PROTECTION     | _SC_MEMORY_PROTECTION     |
| 45570 MSG | _POSIX_MESSAGE_PASSING       | _SC_MESSAGE_PASSING       |
| 45571 MON | _POSIX_MONOTONIC_CLOCK       | _SC_MONOTONIC_CLOCK       |
| 45572     | _POSIX_MULTIPLE_PROCESS      | _SC_MULTIPLE_PROCESS      |
| 45573     | _POSIX_NETWORKING            | _SC_NETWORKING            |
| 45574     | _POSIX_PIPE                  | _SC_PIPE                  |
| 45575 PIO | _POSIX_PRIORITIZED_IO        | _SC_PRIORITIZED_IO        |
| 45576 PS  | _POSIX_PRIORITY_SCHEDULING   | _SC_PRIORITY_SCHEDULING   |
| 45577 THR | _POSIX_READER_WRITER_LOCKS   | _SC_READER_WRITER_LOCKS   |
| 45578 RTS | _POSIX_REALTIME_SIGNALS      | _SC_REALTIME_SIGNALS      |
| 45579     | _POSIX_REGEX                 | _SC_REGEX                 |
| 45580     | _POSIX_SAVED_IDS             | _SC_SAVED_IDS             |
| 45581 SEM | _POSIX_SEMAPHORES            | _SC_SEMAPHORES            |
| 45582 SHM | _POSIX_SHARED_MEMORY_OBJECTS | _SC_SHARED_MEMORY_OBJECTS |
| 45583     | _POSIX_SHELL                 | _SC_SHELL                 |
| 45584     | _POSIX_SIGNALS               | _SC_SIGNALS               |
| 45585     | _POSIX_SINGLE_PROCESS        | _SC_SINGLE_PROCESS        |
| 45586 SPN | _POSIX_SPAWN                 | _SC_SPAWN                 |
| 45587 SPI | _POSIX_SPIN_LOCKS            | _SC_SPIN_LOCKS            |
| 45588 SS  | _POSIX_SPORADIC_SERVER       | _SC_SPORADIC_SERVER       |
| 45589 SIO | _POSIX_SYNCHRONIZED_IO       | _SC_SYNCHRONIZED_IO       |
| 45590     | _POSIX_SYSTEM_DATABASE       | _SC_SYSTEM_DATABASE       |
| 45591     | _POSIX_SYSTEM_DATABASE_R     | _SC_SYSTEM_DATABASE_R     |

45592

45593

45594 TSA

45595 TSS

45596 TCT

45597 TPI

45598 TPP

45599 TPS

45600 TSH

45601 TSF

45602 TSP

45603 THR

45604 TMO

45605 TMR

45606 TRC

45607 TEF

45608 TRI

45609 TRL

45610 TYM

45611

45612

45613

45614

45615

45616

45617

45618

45619

45620

45621

45622

45623

45624

45625 BE

45626

45627

45628

45629

45630

45631

45632

45633

45634 XSI

45635

|  | Variable                          | Value of Name                  |
|--|-----------------------------------|--------------------------------|
|  | _POSIX_THREAD_ATTR_STACKADDR      | _SC_THREAD_ATTR_STACKADDR      |
|  | _POSIX_THREAD_ATTR_STACKSIZE      | _SC_THREAD_ATTR_STACKSIZE      |
|  | _POSIX_THREAD_CPUTIME             | _SC_THREAD_CPUTIME             |
|  | _POSIX_THREAD_PRIO_INHERIT        | _SC_THREAD_PRIO_INHERIT        |
|  | _POSIX_THREAD_PRIO_PROTECT        | _SC_THREAD_PRIO_PROTECT        |
|  | _POSIX_THREAD_PRIORITY_SCHEDULING | _SC_THREAD_PRIORITY_SCHEDULING |
|  | _POSIX_THREAD_PROCESS_SHARED      | _SC_THREAD_PROCESS_SHARED      |
|  | _POSIX_THREAD_SAFE_FUNCTIONS      | _SC_THREAD_SAFE_FUNCTIONS      |
|  | _POSIX_THREAD_SPORADIC_SERVER     | _SC_THREAD_SPORADIC_SERVER     |
|  | _POSIX_THREADS                    | _SC_THREADS                    |
|  | _POSIX_TIMEOUTS                   | _SC_TIMEOUTS                   |
|  | _POSIX_TIMERS                     | _SC_TIMERS                     |
|  | _POSIX_TRACE                      | _SC_TRACE                      |
|  | _POSIX_TRACE_EVENT_FILTER         | _SC_TRACE_EVENT_FILTER         |
|  | _POSIX_TRACE_INHERIT              | _SC_TRACE_INHERIT              |
|  | _POSIX_TRACE_LOG                  | _SC_TRACE_LOG                  |
|  | _POSIX_TYPED_MEMORY_OBJECTS       | _SC_TYPED_MEMORY_OBJECTS       |
|  | _POSIX_USER_GROUPS                | _SC_USER_GROUPS                |
|  | _POSIX_USER_GROUPS_R              | _SC_USER_GROUPS_R              |
|  | _POSIX_VERSION                    | _SC_VERSION                    |
|  | _POSIX_V6_ILP32_OFF32             | _SC_V6_ILP32_OFF32             |
|  | _POSIX_V6_ILP32_OFFBIG            | _SC_V6_ILP32_OFFBIG            |
|  | _POSIX_V6_LP64_OFF64              | _SC_V6_LP64_OFF64              |
|  | _POSIX_V6_LPBIG_OFFBIG            | _SC_V6_LPBIG_OFFBIG            |
|  | _POSIX2_C_BIND                    | _SC_2_C_BIND                   |
|  | _POSIX2_C_DEV                     | _SC_2_C_DEV                    |
|  | _POSIX2_C_VERSION                 | _SC_2_C_VERSION                |
|  | _POSIX2_CHAR_TERM                 | _SC_2_CHAR_TERM                |
|  | _POSIX2_FORT_DEV                  | _SC_2_FORT_DEV                 |
|  | _POSIX2_FORT_RUN                  | _SC_2_FORT_RUN                 |
|  | _POSIX2_LOCALEDEF                 | _SC_2_LOCALEDEF                |
|  | _POSIX2_PBS                       | _SC_2_PBS                      |
|  | _POSIX2_PBS_ACCOUNTING            | _SC_2_PBS_ACCOUNTING           |
|  | _POSIX2_PBS_LOCATE                | _SC_2_PBS_LOCATE               |
|  | _POSIX2_PBS_MESSAGE               | _SC_2_PBS_MESSAGE              |
|  | _POSIX2_PBS_TRACK                 | _SC_2_PBS_TRACK                |
|  | _POSIX2_SW_DEV                    | _SC_2_SW_DEV                   |
|  | _POSIX2_UPE                       | _SC_2_UPE                      |
|  | _POSIX2_VERSION                   | _SC_2_VERSION                  |
|  | _REGEX_VERSION                    | _SC_REGEX_VERSION              |
|  | {PAGE_SIZE}                       | _SC_PAGE_SIZE                  |
|  | {PAGESIZE}                        | _SC_PAGESIZE                   |

45636

45637

45638 THR

45639

45640

45641

45642

45643 RTS

45644 SEM

45645

45646 RTS

45647

45648

45649 TMR

45650

45651

45652 XSI

45653

45654

45655

45656

45657

45658

45659

45660

45661

45662

45663

45664

| Variable                        | Value of Name                    |
|---------------------------------|----------------------------------|
| {PTHREAD_DESTRUCTOR_ITERATIONS} | _SC_THREAD_DESTRUCTOR_ITERATIONS |
| {PTHREAD_KEYS_MAX}              | _SC_THREAD_KEYS_MAX              |
| {PTHREAD_STACK_MIN}             | _SC_THREAD_STACK_MIN             |
| {PTHREAD_THREADS_MAX}           | _SC_THREAD_THREADS_MAX           |
| {RE_DUP_MAX}                    | _SC_RE_DUP_MAX                   |
| {RTSIG_MAX}                     | _SC_RTSIG_MAX                    |
| {SEM_NSEMS_MAX}                 | _SC_SEM_NSEMS_MAX                |
| {SEM_VALUE_MAX}                 | _SC_SEM_VALUE_MAX                |
| {SIGQUEUE_MAX}                  | _SC_SIGQUEUE_MAX                 |
| {STREAM_MAX}                    | _SC_STREAM_MAX                   |
| {SYMLOOP_MAX}                   | _SC_SYMLOOP_MAX                  |
| {TIMER_MAX}                     | _SC_TIMER_MAX                    |
| {TTY_NAME_MAX}                  | _SC_TTY_NAME_MAX                 |
| {TZNAME_MAX}                    | _SC_TZNAME_MAX                   |
| _XBS5_ILP32_OFF32 (LEGACY)      | _SC_XBS5_ILP32_OFF32 (LEGACY)    |
| _XBS5_ILP32_OFFBIG (LEGACY)     | _SC_XBS5_ILP32_OFFBIG (LEGACY)   |
| _XBS5_LP64_OFF64 (LEGACY)       | _SC_XBS5_LP64_OFF64 (LEGACY)     |
| _XBS5_LPBIG_OFFBIG (LEGACY)     | _SC_XBS5_LPBIG_OFFBIG (LEGACY)   |
| _XOPEN_CRYPT                    | _SC_XOPEN_CRYPT                  |
| _XOPEN_ENH_I18N                 | _SC_XOPEN_ENH_I18N               |
| _XOPEN_LEGACY                   | _SC_XOPEN_LEGACY                 |
| _XOPEN_REALTIME                 | _SC_XOPEN_REALTIME               |
| _XOPEN_REALTIME_THREADS         | _SC_XOPEN_REALTIME_THREADS       |
| _XOPEN_SHM                      | _SC_XOPEN_SHM                    |
| _XOPEN_UNIX                     | _SC_XOPEN_UNIX                   |
| _XOPEN_VERSION                  | _SC_XOPEN_VERSION                |
| _XOPEN_XCU_VERSION              | _SC_XOPEN_XCU_VERSION            |

45665 **RETURN VALUE**

45666

45667

45668

If *name* is an invalid value, *sysconf()* shall return  $-1$  and set *errno* to indicate the error. If the variable corresponding to *name* has no limit, *sysconf()* shall return  $-1$  without changing the value of *errno*. Note that indefinite limits do not imply infinite limits; see `<limits.h>`.

45669

45670

45671

45672

Otherwise, *sysconf()* shall return the current variable value on the system. The value returned shall not be more restrictive than the corresponding value described to the application when it was compiled with the implementation's `<limits.h>` or `<unistd.h>`. The value does not change during the lifetime of the calling process.

45673 **ERRORS**

45674

45675

The *sysconf()* function shall fail if:

[EINVAL] The value of the *name* argument is invalid.

45676 **EXAMPLES**

45677

None.

45678 **APPLICATION USAGE**

45679

45680

45681

As  $-1$  is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *sysconf()*, and, if it returns  $-1$ , check to see if *errno* is non-zero.

45682

45683

45684

If the value of *sysconf(\_SC\_2\_VERSION)* is not equal to the value of the `_POSIX2_VERSION` symbolic constant, the utilities available via *system()* or *popen()* might not behave as described in the Shell and Utilities volume of IEEE Std. 1003.1-200x. This would mean that the application is

45685 not running in an environment that conforms to the Shell and Utilities volume of  
45686 IEEE Std. 1003.1-200x. Some applications might be able to deal with this, others might not.  
45687 However, the functions defined in this volume of IEEE Std. 1003.1-200x continue to operate as  
45688 specified, even if: `sysconf(_SC_2_VERSION)` reports that the utilities no longer perform as  
45689 specified.

#### 45690 RATIONALE

45691 This functionality was added in response to requirements of application developers and of  
45692 system vendors who deal with many international system configurations. It is closely related to  
45693 `pathconf()` and `fpathconf()`.

45694 Although a portable application can run on all systems by never demanding more resources  
45695 than the minimum values published in this volume of IEEE Std. 1003.1-200x, it is useful for that  
45696 application to be able to use the actual value for the quantity of a resource available on any  
45697 given system. To do this, the application makes use of the value of a symbolic constant in  
45698 `<limits.h>` or `<unistd.h>`.

45699 However, once compiled, the application must still be able to cope if the amount of resource  
45700 available is increased. To that end, an application may need a means of determining the quantity  
45701 of a resource, or the presence of an option, at execution time.

45702 Two examples are offered:

- 45703 1. Applications may wish to act differently on systems with or without job control.  
45704 Applications vendors who wish to distribute only a single binary package to all instances  
45705 of a computer architecture would be forced to assume job control is never available if it  
45706 were to rely solely on the `<unistd.h>` value published in this volume of  
45707 IEEE Std. 1003.1-200x.
- 45708 2. International applications vendors occasionally require knowledge of the number of clock  
45709 ticks per second. Without these facilities, they would be required to either distribute their  
45710 applications partially in source form or to have 50Hz and 60Hz versions for the various  
45711 countries in which they operate.

45712 It is the knowledge that many applications are actually distributed widely in executable form  
45713 that leads to this facility. If limited to the most restrictive values in the headers, such  
45714 applications would have to be prepared to accept the most limited environments offered by the  
45715 smallest microcomputers. Although this is entirely portable, there was a consensus that they  
45716 should be able to take advantage of the facilities offered by large systems, without the  
45717 restrictions associated with source and object distributions.

45718 During the discussions of this feature, it was pointed out that it is almost always possible for an  
45719 application to discern what a value might be at runtime by suitably testing the various functions  
45720 themselves. And, in any event, it could always be written to adequately deal with error returns  
45721 from the various functions. In the end, it was felt that this imposed an unreasonable level of  
45722 complication and sophistication on the application writer.

45723 This runtime facility is not meant to provide ever-changing values that applications have to  
45724 check multiple times. The values are seen as changing no more frequently than once per system  
45725 initialization, such as by a system administrator or operator with an automatic configuration  
45726 program. This volume of IEEE Std. 1003.1-200x specifies that they shall not change within the  
45727 lifetime of the process.

45728 Some values apply to the system overall and others vary at the file system or directory level. The  
45729 latter are described in `pathconf()`.

45730 Note that all values returned must be expressible as integers. String values were considered, but  
45731 the additional flexibility of this approach was rejected due to its added complexity of

45732 implementation and use.

45733 Some values, such as {PATH\_MAX}, are sometimes so large that they must not be used to, say,  
45734 allocate arrays. The *sysconf()* function returns a negative value to show that this symbolic  
45735 constant is not even defined in this case.

45736 Similar to *pathconf()*, this permits the implementation not to have a limit. When one resource is  
45737 infinite, returning an error indicating that some other resource limit has been reached is  
45738 conforming behavior.

#### 45739 FUTURE DIRECTIONS

45740 None.

#### 45741 SEE ALSO

45742 *confstr()*, *pathconf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <limits.h>,  
45743 <unistd.h>, the Shell and Utilities volume of IEEE Std. 1003.1-200x, *getconf*

#### 45744 CHANGE HISTORY

45745 First released in Issue 3.

45746 Entry included for alignment with the POSIX.1-1988 standard.

#### 45747 Issue 4

45748 The type of the function return value is expanded to **long**.

45749 `_XOPEN_VERSION` is added to the table of configurable system limits; this should have been  
45750 included in Issue 3.

45751 The following variables are added to the table of configurable system limits in the  
45752 DESCRIPTION and marked as extensions:

45753 `_XOPEN_CRYPT`  
45754 `_XOPEN_ENH_I18N`  
45755 `_XOPEN_SHM`  
45756 `_XOPEN_UNIX`

45757 In the RETURN VALUE section the header <time.h> is given as an alternative to <limits.h> and  
45758 <unistd.h>.

45759 The second paragraph is added to the APPLICATION USAGE section.

45760 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 45761 • The variables {STREAM\_MAX} and {TZNAME\_MAX} are added to the table of variables in  
45762 the DESCRIPTION.

45763 The following change is incorporated for alignment with the ISO POSIX-2 standard:

- 45764 • The following variables are added to the table of configurable system limits in the  
45765 DESCRIPTION:

|       |                    |                   |                 |
|-------|--------------------|-------------------|-----------------|
| 45766 | {BC_BASE_MAX}      | _POSIX2_C_BIND    | _POSIX2_SW_DEV  |
| 45767 | {BC_DIM_MAX}       | _POSIX2_C_DEV     | _POSIX2_VERSION |
| 45768 | {BC_SCALE_MAX}     | _POSIX2_C_VERSION | {RE_DUP_MAX}    |
| 45769 | {BC_STRING_MAX}    | _POSIX2_CHAR_TERM |                 |
| 45770 | {COLL_WEIGHTS_MAX} | _POSIX2_FORT_DEV  |                 |
| 45771 | {EXPR_NEST_MAX}    | _POSIX2_FORT_RUN  |                 |
| 45772 | {LINE_MAX}         | _POSIX2_LOCALEDEF |                 |

**45773 Issue 4, Version 2**

45774 For X/OPEN UNIX conformance, the {ATEXIT\_MAX}, {IOV\_MAX}, {PAGESIZE}, {PAGE\_SIZE},  
 45775 and \_XOPEN\_UNIX variables are added to the list of configurable system values that can be  
 45776 determined by calling *sysconf()*.

**45777 Issue 5**

45778 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 45779 Threads Extension.

45780 The \_XBS\_ variables and name values are added to the table of system variables in the  
 45781 DESCRIPTION. These are all marked EX.

**45782 Issue 6**

45783 The symbol CLK\_TCK is obsolescent and removed. It is replaced with the phrase “clock ticks  
 45784 per second”.

45785 The symbol {PASS\_MAX} is removed.

45786 The following changes were made to align with the IEEE P1003.1a draft standard:

- 45787 • Table entries added for the following variables: \_SC\_REGEX, \_SC\_SHELL,  
 45788 \_SC\_REGEX\_VERSION, \_SC\_SYMLINK\_MAX.

45789 The following *sysconf()* variables and their associated names are added for alignment with  
 45790 IEEE Std. 1003.1d-1999:

45791 \_POSIX\_ADVISORY\_INFO  
 45792 \_POSIX\_CPUTIME  
 45793 \_POSIX\_SPAWN  
 45794 \_POSIX\_SPARADIC\_SERVER  
 45795 \_POSIX\_THREAD\_CPUTIME  
 45796 \_POSIX\_THREAD\_SPARADIC\_SERVER  
 45797 \_POSIX\_TIMEOUTS

45798 The following changes are made to the DESCRIPTION for alignment with  
 45799 IEEE Std. 1003.1j-2000:

- 45800 • A statement expressing the dependency of support for some system variables on  
 45801 implementation options is added.
- 45802 • The following system variables are added:

45803 \_POSIX\_BARRIERS  
 45804 \_POSIX\_CLOCK\_SELECTION  
 45805 \_POSIX\_MONOTONIC\_CLOCK  
 45806 \_POSIX\_READER\_WRITER\_LOCKS  
 45807 \_POSIX\_SPIN\_LOCKS  
 45808 \_POSIX\_TYPED\_MEMORY\_OBJECTS

45809 The following system variables are added for alignment with IEEE Std. 1003.2d-1994:

45810 \_POSIX2\_PBS  
 45811 \_POSIX2\_PBS\_ACCOUNTING  
 45812 \_POSIX2\_PBS\_LOCATE  
 45813 \_POSIX2\_PBS\_MESSAGE  
 45814 \_POSIX2\_PBS\_TRACK

45815 The following *sysconf()* variables and their associated names are added for alignment with  
 45816 IEEE Std. 1003.1q-2000:

45817        \_POSIX\_TRACE  
45818        \_POSIX\_TRACE\_EVENT\_FILTER  
45819        \_POSIX\_TRACE\_INHERIT  
45820        \_POXIC\_TRACE\_LOG

45821        The macros associated with the *c89* programming models are marked LEGACY, and new  
45822        equivalent macros associated with *c99* are introduced.

45823 **NAME**

45824            syslog — log a message

45825 **SYNOPSIS**

45826 XSI        #include <syslog.h>

45827            void syslog(int *priority*, const char \**message*, ... /\* *argument* \*/);

45828

45829 **DESCRIPTION**

45830            Refer to *closelog*().



45831 **NAME**

45832 system — issue a command

45833 **SYNOPSIS**

45834 #include &lt;stdlib.h&gt;

45835 int system(const char \**command*);45836 **DESCRIPTION**

45837 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 45838 conflict between the requirements described here and the ISO C standard is unintentional. This  
 45839 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

45840 The *system()* function passes the string pointed to by *command* to the host environment to be  
 45841 executed by a command processor in an implementation-defined manner. The environment of  
 45842 the executed command shall be as if a child process were created using the *fork()* function, and  
 45843 the child process invoked a command interpreter using the *execl()* function.

45844 **CX** If the implementation supports the Shell and Utilities volume of IEEE Std. 1003.1-200x  
 45845 commands, the environment of the executed command shall be as if a child process were created  
 45846 using *fork()*, and the child process invoked the *sh* utility using *execl()* as follows:

```
45847 execl(<shell path>, "sh", "-c", command, (char *)0);
```

45848 where <*shell path*> is an unspecified path name for the *sh* utility.

45849 The *system()* function ignores the SIGINT and SIGQUIT signals, and blocks the SIGCHLD  
 45850 signal, while waiting for the command to terminate. If this might cause the application to miss a  
 45851 signal that would have killed it, then the application should examine the return value from  
 45852 *system()* and take whatever action is appropriate to the application if the command terminated  
 45853 due to receipt of a signal.

45854 The *system()* function shall not affect the termination status of any child of the calling processes  
 45855 other than the process or processes it itself creates.

45856 The *system()* function shall not return until the child process has terminated.

45857 **RETURN VALUE**

45858 If *command* is a null pointer, *system()* shall return non-zero to indicate that a command processor  
 45859 **CX** is available, or zero if none is available. If the implementation supports the utilities defined in  
 45860 the Shell and Utilities volume of IEEE Std. 1003.1-200x, *system()* shall always return non-zero  
 45861 when *command* is NULL.

45862 **CX** If *command* is not a null pointer, *system()* shall return the termination status of the command  
 45863 language interpreter in the format specified by *waitpid()*. If the implementation supports the  
 45864 utilities defined in the Shell and Utilities volume of IEEE Std. 1003.1-200x, the termination status  
 45865 shall be as defined for the *sh* utility; otherwise, the termination status is unspecified. If some  
 45866 error prevents the command language interpreter from executing after the child process is  
 45867 created, the return value from *system()* shall be as if the command language interpreter had  
 45868 terminated using *exit(127)* or *\_exit(127)*. If a child process cannot be created, or if the  
 45869 termination status for the command language interpreter cannot be obtained, *system()* shall  
 45870 return -1 and set *errno* to indicate the error.

45871 **ERRORS**

45872 **CX** The *system()* function may set *errno* values as described by *fork()*.

45873 In addition, *system()* may fail if:

45874 **CX** [ECHILD] The status of the child process created by *system()* is no longer available.

45875 **EXAMPLES**

45876 None.

45877 **APPLICATION USAGE**

45878 If the return value of *system()* is not `-1`, its value can be decoded through the use of the macros  
45879 described in `<sys/wait.h>`. For convenience, these macros are also provided in `<stdlib.h>`.

45880 To determine whether or not the environment specified in the Shell and Utilities volume of  
45881 IEEE Std. 1003.1-200x is present, use *sysconf(SC\_2\_VERSION)*.

45882 Note that, while *system()* must ignore SIGINT and SIGQUIT and block SIGCHLD while waiting  
45883 for the child to terminate, the handling of signals in the executed command is as specified by  
45884 *fork()* and *exec*. For example, if SIGINT is being caught or is set to SIG\_DFL when *system()* is  
45885 called, then the child is started with SIGINT handling set to SIG\_DFL.

45886 Ignoring SIGINT and SIGQUIT in the parent process prevents coordination problems (two  
45887 processes reading from the same terminal, for example) when the executed command ignores or  
45888 catches one of the signals. It is also usually the correct action when the user has given a  
45889 command to the application to be executed synchronously (as in the `'!'` command in many  
45890 interactive applications). In either case, the signal should be delivered only to the child process,  
45891 not to the application itself. There is one situation where ignoring the signals might have less  
45892 than the desired effect. This is when the application uses *system()* to perform some task invisible  
45893 to the user. If the user typed the interrupt character ("`^C`", for example) while *system()* is being  
45894 used in this way, one would expect the application to be killed, but only the executed command  
45895 is killed. Applications that use *system()* in this way should carefully check the return status from  
45896 *system()* to see if the executed command was successful, and should take appropriate action  
45897 when the command fails.

45898 Blocking SIGCHLD while waiting for the child to terminate prevents the application from  
45899 catching the signal and obtaining status from *system()*'s child process before *system()* can get the  
45900 status itself.

45901 The context in which the utility is ultimately executed may differ from that in which *system()*  
45902 was called. For example, file descriptors that have the FD\_CLOEXEC flag set are closed, and the  
45903 process ID and parent process ID are different. Also, if the executed utility changes its  
45904 environment variables or its current working directory, that change is not reflected in the caller's  
45905 context.

45906 There is no defined way for an application to find the specific path for the shell. However,  
45907 *confstr()* can provide a value for *PATH* that is guaranteed to find the *sh* utility.

45908 **RATIONALE**

45909 The *system()* function should not be used by programs that have set user (or group) ID  
45910 privileges. The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used  
45911 instead. This prevents any unforeseen manipulation of the environment of the user that could  
45912 cause execution of commands not anticipated by the calling program.

45913 There are three levels of specification for the *system()* function. The ISO C standard gives the  
45914 most basic. It requires that the function exists, and defines a way for an application to query  
45915 whether a command language interpreter exists. It says nothing about the command language or  
45916 the environment in which the command is interpreted.

45917 IEEE Std. 1003.1-200x places additional restrictions on *system()*. It requires that if there is a  
45918 command language interpreter, the environment must be as specified by *fork()* and *exec*. This  
45919 ensures, for example, that close-on-exec works, that file locks are not inherited, and that the  
45920 process ID is different. It also specifies the return value from *system()* when the command line  
45921 can be run, thus giving the application some information about the command's completion

45922 status IEEE Std. 1003.1-200x in its base definition still says nothing about the interpretation of  
45923 the command.

45924 Finally, IEEE Std. 1003.1-200x requires the command to be interpreted as in the shell command  
45925 language defined in the Shell and Utilities volume of IEEE Std. 1003.1-200x.

45926 Note that, *system(NULL)* is required to return non-zero, indicating that there is a command  
45927 language interpreter. At first glance, this would seem to conflict with the ISO C standard which  
45928 allows *system(NULL)* to return zero. There is no conflict, however. A system must have a  
45929 command language interpreter, and is non-conforming if none is present. It is therefore  
45930 permissible for the *system()* function on such a system to implement the behavior specified by  
45931 the ISO C standard as long as it is understood that the implementation does not conform to  
45932 IEEE Std. 1003.1-200x if *system(NULL)* returns zero.

45933 It was explicitly decided that when *command* is NULL, *system()* should not be required to check  
45934 to make sure that the command language interpreter actually exists with the correct mode, that  
45935 there are enough processes to execute it, and so on. The call *system(NULL)* could, theoretically,  
45936 check for such problems as too many existing child processes, and return zero. However, it  
45937 would be inappropriate to return zero due to such a (presumably) transient condition. If some  
45938 condition exists that is not under the control of this application and that would cause any  
45939 *system()* call to fail, that system has been rendered non-conforming.

45940 Early drafts required, or allowed, *system()* to return with *errno* set to [EINTR] if it was  
45941 interrupted with a signal. This error return was removed, and a requirement that *system()* not  
45942 return until the child has terminated was added. This means that if a *waitpid()* call in *system()*  
45943 exits with *errno* set to [EINTR], *system()* must re-issue the *waitpid()*. This change was made for  
45944 two reasons:

- 45945 1. There is no way for an application to clean up if *system()* returns [EINTR], short of calling  
45946 *wait()*, and that could have the undesirable effect of returning the status of children other  
45947 than the one started by *system()*.
- 45948 2. While it might require a change in some historical implementations, those  
45949 implementations already have to be changed because they use *wait()* instead of *waitpid()*.

45950 Note that if the application is catching SIGCHLD signals, it will receive such a signal before a  
45951 successful *system()* call returns.

45952 To conform to IEEE Std. 1003.1-200x, *system()* must use *waitpid()*, or some similar function,  
45953 instead of *wait()*.

45954 The following code sample illustrates how *system()* might be implemented on an  
45955 implementation conforming to IEEE Std. 1003.1-200x.

```
45956 #include <signal.h>
45957 int system(const char *cmd)
45958 {
45959 int stat;
45960 pid_t pid;
45961 struct sigaction sa, savintr, savequit;
45962 sigset_t saveblock;
45963 if (cmd == NULL)
45964 return(1);
45965 sa.sa_handler = SIG_IGN;
45966 sigemptyset(&sa.sa_mask);
45967 sa.sa_flags = 0;
45968 sigemptyset(&savintr.sa_mask);
```

```

45969 sigemptyset(&savequit.sa_mask);
45970 sigaction(SIGINT, &sa, &saveintr);
45971 sigaction(SIGQUIT, &sa, &savequit);
45972 sigaddset(&sa.sa_mask, SIGCHLD);
45973 sigprocmask(SIG_BLOCK, &sa.sa_mask, &saveblock);
45974 if ((pid = fork()) == 0) {
45975 sigaction(SIGINT, &saveintr, (struct sigaction *)0);
45976 sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
45977 sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
45978 execl("/bin/sh", "sh", "-c", cmd, (char *)0);
45979 _exit(127);
45980 }
45981 if (pid == -1) {
45982 stat = -1; /* errno comes from fork() */
45983 } else {
45984 while (waitpid(pid, &stat, 0) == -1) {
45985 if (errno != EINTR){
45986 stat = -1;
45987 break;
45988 }
45989 }
45990 }
45991 sigaction(SIGINT, &saveintr, (struct sigaction *)0);
45992 sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
45993 sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
45994 return(stat);
45995 }

```

45996 Note that, while a particular implementation of *system()* (such as the one above) can assume a  
45997 particular path for the shell, such a path is not necessarily valid on another system. The above  
45998 example is not portable, and is not intended to be.

45999 One reviewer suggested that an implementation of *system()* might want to use an environment  
46000 variable such as *SHELL* to determine which command interpreter to use. The supposed  
46001 implementation would use the default command interpreter if the one specified by the  
46002 environment variable was not available. This would allow a user, when using an application  
46003 that prompts for command lines to be processed using *system()*, to specify a different command  
46004 interpreter. Such an implementation is discouraged. If the alternate command interpreter did not  
46005 follow the command line syntax specified in the Shell and Utilities volume of  
46006 IEEE Std. 1003.1-200x, then changing *SHELL* would render *system()* non-conforming. This would  
46007 affect applications that expected the specified behavior from *system()*, and since the Shell and  
46008 Utilities volume of IEEE Std. 1003.1-200x does not mention that *SHELL* affects *system()*, the  
46009 application would not know that it needed to unset *SHELL*.

#### 46010 FUTURE DIRECTIONS

46011 None.

#### 46012 SEE ALSO

46013 *exec*, *pipe()*, *waitpid()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<limits.h>**,  
46014 **<signal.h>**, **<stdlib.h>**, **<sys/wait.h>**, the Shell and Utilities volume of IEEE Std. 1003.1-200x

46015 **CHANGE HISTORY**

46016 First released in Issue 1. Derived from Issue 1 of the SVID.

46017 **Issue 4**

46018 Extensions beyond the ISO C standard are now marked.

46019 The following changes are incorporated for alignment with the ISO POSIX-2 standard:

- 46020
  - The function is no longer marked as an extension.
- 46021
  - The name of the argument is changed from *string* to *command*, and its type is changed from
- 46022
  - char\*** to **const char\***.
- 46023
  - The DESCRIPTION and RETURN VALUE sections are completely replaced to bring them in
- 46024
  - line with the ISO POSIX-2 standard. They still describe essentially the same functionality,
- 46025
  - albeit that the definition is more complete.
- 46026
  - The ERRORS section is changed to indicate that *system()* may return error values described
- 46027
  - for *fork()*.
- 46028
  - The APPLICATION USAGE section is added.

46029 The following changes were made to align with the IEEE P1003.1a draft standard:

- 46030
  - The DESCRIPTION is adjusted to reflect the behavior on systems that do not support the
- 46031
  - Shell option.

46032 **NAME**

46033 tan, tanf, tanl — tangent function

46034 **SYNOPSIS**

46035 #include &lt;math.h&gt;

46036 double tan(double x);

46037 float tanf(float x);

46038 long double tanl(long double x);

46039 **DESCRIPTION**

46040 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 46041 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46042 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

46043 These functions shall compute the tangent of its argument *x*, measured in radians.

46044 An application wishing to check for error situations should set *errno* to 0 before calling *tan()*. If  
 46045 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

46046 The *tan()* function may lose accuracy when its argument is far from 0.0.46047 **RETURN VALUE**46048 Upon successful completion, these functions shall return the tangent of *x*.46049 XSI If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

46050 If *x* is  $\pm\text{Inf}$ , either 0.0 shall be returned and *errno* set to [EDOM], or NaN shall be returned and  
 46051 *errno* may be set to [EDOM].

46052 If the correct value would cause overflow,  $\pm\text{HUGE\_VAL}$  shall be returned and *errno* shall be set  
 46053 to [ERANGE].

46054 If the correct value would cause underflow, 0.0 shall be returned and *errno* may be set to  
 46055 [ERANGE].

46056 **ERRORS**

46057 These functions shall fail if:

46058 [ERANGE] The value to be returned would cause overflow.

46059 These functions may fail if:

46060 XSI [EDOM] The value *x* is NaN or  $\pm\text{Inf}$ .

46061 [ERANGE] The value to be returned would cause underflow.

46062 XSI No other errors shall occur.

46063 **EXAMPLES**46064 **Taking the Tangent of a 45-Degree Angle**

46065 #include &lt;math.h&gt;

46066 ...

46067 double radians = 45.0 \* M\_PI / 180;

46068 double result;

46069 ...

46070 result = tan (radians);

46071 **APPLICATION USAGE**

46072 None.

46073 **RATIONALE**

46074 None.

46075 **FUTURE DIRECTIONS**

46076 None.

46077 **SEE ALSO**46078 *atan()*, *isnan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>46079 **CHANGE HISTORY**

46080 First released in Issue 1. Derived from Issue 1 of the SVID.

46081 **Issue 4**46082 References to *matherr()* are removed.46083 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the  
46084 ISO C standard and to rationalize error handling in the mathematics functions.

46085 The return value specified for [EDOM] is marked as an extension.

46086 **Issue 5**46087 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
46088 in previous issues.46089 **Issue 6**46090 The *tanf()* and *tanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

46091 **NAME**

46092 tanh, tanhf, tanhl — hyperbolic tangent function

46093 **SYNOPSIS**

46094 #include &lt;math.h&gt;

46095 double tanh(double x);

46096 float tanhf(float x);

46097 long double tanhl(long double x);

46098 **DESCRIPTION**

46099 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 46100 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46101 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

46102 These functions shall compute the hyperbolic tangent of  $x$ .

46103 An application wishing to check for error situations should set *errno* to 0 before calling *tanh()*. If  
 46104 *errno* is non-zero on return, or the return value is NaN, an error has occurred.

46105 **RETURN VALUE**46106 Upon successful completion, these functions shall return the hyperbolic tangent of  $x$ .46107 **XSI** If  $x$  is NaN, NaN shall be returned and *errno* may be set to [EDOM].

46108 If the correct value would cause underflow, 0.0 shall be returned and *errno* may be set to  
 46109 [ERANGE].

46110 **ERRORS**

46111 These functions may fail if:

46112 **XSI** [EDOM] The value of  $x$  is NaN.

46113 [ERANGE] The correct result would cause underflow.

46114 **XSI** No other errors shall occur.46115 **EXAMPLES**

46116 None.

46117 **APPLICATION USAGE**

46118 None.

46119 **RATIONALE**

46120 None.

46121 **FUTURE DIRECTIONS**

46122 None.

46123 **SEE ALSO**46124 *atanh()*, *isnan()*, *tan()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>46125 **CHANGE HISTORY**

46126 First released in Issue 1. Derived from Issue 1 of the SVID.

46127 **Issue 4**46128 References to *matherr()* are removed.

46129 The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the  
 46130 ISO C standard and to rationalize error handling in the mathematics functions.

46131 The return value specified for [EDOM] is marked as an extension.



46132 **Issue 5**

46133 The DESCRIPTION is updated to indicate how an application should check for an error. This  
46134 text was previously published in the APPLICATION USAGE section.

46135 **Issue 6**

46136 The *tanhf()* and *tanhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

46137 **NAME**

46138 tcdrain — wait for transmission of output

46139 **SYNOPSIS**

46140 #include &lt;termios.h&gt;

46141 int tcdrain(int *fildev*);46142 **DESCRIPTION**46143 The *tcdrain()* function shall wait until all output written to the object referred to by *fildev* is  
46144 transmitted. The *fildev* argument is an open file descriptor associated with a terminal.46145 Any attempts to use *tcdrain()* from a process which is a member of a background process group  
46146 on a *fildev* associated with its controlling terminal, shall cause the process group to be sent a  
46147 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is  
46148 allowed to perform the operation, and no signal is sent.46149 **RETURN VALUE**46150 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46151 indicate the error.46152 **ERRORS**46153 The *tcdrain()* function shall fail if:46154 [EBADF] The *fildev* argument is not a valid file descriptor. |46155 [EINTR] A signal interrupted *tcdrain()*. |46156 [ENOTTY] The file associated with *fildev* is not a terminal. |46157 The *tcdrain()* function may fail if:46158 [EIO] The process group of the writing process is orphaned, and the writing process |  
46159 is not ignoring or blocking SIGTTOU. |46160 **EXAMPLES**

46161 None.

46162 **APPLICATION USAGE**

46163 None.

46164 **RATIONALE**

46165 None.

46166 **FUTURE DIRECTIONS**

46167 None.

46168 **SEE ALSO**46169 *tcfldsh()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <termios.h>, <unistd.h>, the |  
46170 Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface |46171 **CHANGE HISTORY**

46172 First released in Issue 3.

46173 Entry included for alignment with the POSIX.1-1988 standard.

46174 **Issue 4**

46175 The [EIO] error is added to the ERRORS section.

46176 The FUTURE DIRECTIONS section is added.

46177 The following change is incorporated for alignment with the FIPS requirements:

46178           • The words “If \_POSIX\_JOB\_CONTROL is defined” are removed from the start of the second  
46179           paragraph in the DESCRIPTION. This is because job control is defined as mandatory for  
46180           Issue 4 conforming implementations.

46181 **Issue 6**

46182           The following new requirements on POSIX implementations derive from alignment with the  
46183           Single UNIX Specification:

- 46184           • In the DESCRIPTION, the final paragraph is no longer conditional on  
46185           \_POSIX\_JOB\_CONTROL. This is a FIPS requirement.
- 46186           • The [EIO] error is added.

46187 **NAME**

46188 tcflow — suspend or restart the transmission or reception of data

46189 **SYNOPSIS**

46190 #include &lt;termios.h&gt;

46191 int tcflow(int *fildev*, int *action*);46192 **DESCRIPTION**

46193 The *tcflow()* function shall suspend or restart transmission or reception of data on the object  
 46194 referred to by *fildev*, depending on the value of *action*. The *fildev* argument is an open file  
 46195 descriptor associated with a terminal.

- 46196 • If *action* is TCOFF, output shall be suspended.
- 46197 • If *action* is TCOON, suspended output shall be restarted.
- 46198 • If *action* is TCIOFF, the system shall transmit a STOP character, which is intended to cause  
 46199 the terminal device to stop transmitting data to the system.
- 46200 • If *action* is TCION, the system shall transmit a START character, which is intended to cause  
 46201 the terminal device to start transmitting data to the system.

46202 The default on the opening of a terminal file is that neither its input nor its output are  
 46203 suspended.

46204 Attempts to use *tcflow()* from a process which is a member of a background process group on a  
 46205 *fildev* associated with its controlling terminal, shall cause the process group to be sent a  
 46206 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is  
 46207 allowed to perform the operation, and no signal is sent.

46208 **RETURN VALUE**

46209 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 46210 indicate the error.

46211 **ERRORS**46212 The *tcflow()* function shall fail if:

- 46213 [EBADF] The *fildev* argument is not a valid file descriptor.
- 46214 [EINVAL] The *action* argument is not a supported value.
- 46215 [ENOTTY] The file associated with *fildev* is not a terminal.

46216 The *tcflow()* function may fail if:

- 46217 [EIO] The process group of the writing process is orphaned, and the writing process  
 46218 is not ignoring or blocking SIGTTOU.

46219 **EXAMPLES**

46220 None.

46221 **APPLICATION USAGE**

46222 None.

46223 **RATIONALE**

46224 None.

46225 **FUTURE DIRECTIONS**

46226 None.

**46227 SEE ALSO**

46228 *tcsendbreak()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <termios.h>, <unistd.h>, the  
46229 Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface

**46230 CHANGE HISTORY**

46231 First released in Issue 3.

46232 Entry included for alignment with the POSIX.1-1988 standard.

**46233 Issue 4**

46234 The descriptions of TCIOFF and TCION are reworded, indicating the intended consequences of  
46235 transmitting stop and start characters. Issue 3 implied that these consequences were guaranteed.

46236 The [EIO] error is added to the ERRORS section.

46237 The FUTURE DIRECTIONS section is added.

46238 The following change is incorporated for alignment with the FIPS requirements:

- 46239 • The words “If \_POSIX\_JOB\_CONTROL is defined” are removed from the start of the second  
46240 paragraph in the DESCRIPTION. This is because job control is defined as mandatory for  
46241 Issue 4 conforming implementations.

**46242 Issue 6**

46243 The following new requirements on POSIX implementations derive from alignment with the  
46244 Single UNIX Specification:

- 46245 • The [EIO] error is added.

46246 **NAME**

46247 tcflush — flush non-transmitted output data, non-read input data, or both

46248 **SYNOPSIS**

46249 #include &lt;termios.h&gt;

46250 int tcflush(int *fildev*, int *queue\_selector*);46251 **DESCRIPTION**46252 Upon successful completion, *tcflush()* shall discard data written to the object referred to by *fildev*  
46253 (an open file descriptor associated with a terminal) but not transmitted, or data received but not  
46254 read, depending on the value of *queue\_selector*:

- 46255 • If *queue\_selector* is TCIFLUSH, it shall flush data received but not read.
- 46256 • If *queue\_selector* is TCOFLUSH, it shall flush data written but not transmitted.
- 46257 • If *queue\_selector* is TCIOFLUSH, it shall flush both data received but not read and data  
46258 written but not transmitted.

46259 Attempts to use *tcflush()* from a process which is a member of a background process group on a  
46260 *fildev* associated with its controlling terminal, shall cause the process group to be sent a  
46261 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is  
46262 allowed to perform the operation, and no signal is sent.

46263 **RETURN VALUE**

46264 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46265 indicate the error.

46266 **ERRORS**46267 The *tcflush()* function shall fail if:46268 [EBADF] The *fildev* argument is not a valid file descriptor. |46269 [EINVAL] The *queue\_selector* argument is not a supported value. |46270 [ENOTTY] The file associated with *fildev* is not a terminal. |46271 The *tcflush()* function may fail if:

46272 [EIO] The process group of the writing process is orphaned, and the writing process |  
46273 is not ignoring or blocking SIGTTOU. |

46274 **EXAMPLES**

46275 None.

46276 **APPLICATION USAGE**

46277 None.

46278 **RATIONALE**

46279 None.

46280 **FUTURE DIRECTIONS**

46281 None.

46282 **SEE ALSO**

46283 *tcdrain()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <termios.h>, <unistd.h>, the |  
46284 Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface |

46285 **CHANGE HISTORY**

46286 First released in Issue 3.

46287 Entry included for alignment with the POSIX.1-1988 standard.

46288 **Issue 4**46289 The DESCRIPTION is modified to indicate that the flush operation only results if the call to  
46290 *tcflush()* is successful.

46291 The [EIO] error is added to the ERRORS section.

46292 The FUTURE DIRECTIONS section is added.

46293 The following change is incorporated for alignment with the FIPS requirements:

- 46294
- The words “If \_POSIX\_JOB\_CONTROL is defined” are removed from the start of the second  
46295 paragraph in the DESCRIPTION. This is because job control is defined as mandatory for  
46296 Issue 4 conforming implementations.

46297 **Issue 6**46298 The Open Group corrigenda item U035/1 has been applied. In the ERRORS and APPLICATION  
46299 USAGE sections, references to *tcflow()* are replaced with *tcflush()*.46300 The following new requirements on POSIX implementations derive from alignment with the  
46301 Single UNIX Specification:

- 46302
- In the DESCRIPTION, the final paragraph is no longer conditional on  
46303 \_POSIX\_JOB\_CONTROL. This is a FIPS requirement.

- 46304
- The [EIO] error is added.

## 46305 NAME

46306 tcgetattr — get the parameters associated with the terminal

## 46307 SYNOPSIS

46308 #include <termios.h>

46309 int tcgetattr(int *fildev*, struct termios \**termios\_p*);

## 46310 DESCRIPTION

46311 The *tcgetattr()* function shall get the parameters associated with the terminal referred to by *fildev*  
46312 and store them in the **termios** structure referenced by *termios\_p*. The *fildev* argument is an open  
46313 file descriptor associated with a terminal.

46314 The *termios\_p* argument is a pointer to a **termios** structure.

46315 The *tcgetattr()* operation is allowed from any process.

46316 If the terminal device supports different input and output baud rates, the baud rates stored in  
46317 the **termios** structure returned by *tcgetattr()* shall reflect the actual baud rates, even if they are  
46318 equal. If differing baud rates are not supported, the rate returned as the output baud rate shall be  
46319 the actual baud rate. If the terminal device does not support split baud rates, the input baud rate  
46320 stored in the **termios** structure shall be the output rate (as one of the symbolic values).

## 46321 RETURN VALUE

46322 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46323 indicate the error.

## 46324 ERRORS

46325 The *tcgetattr()* function shall fail if:

46326 [EBADF] The *fildev* argument is not a valid file descriptor.

46327 [ENOTTY] The file associated with *fildev* is not a terminal.

## 46328 EXAMPLES

46329 None.

## 46330 APPLICATION USAGE

46331 None.

## 46332 RATIONALE

46333 Care must be taken when changing the terminal attributes. Applications should always do a  
46334 *tcgetattr()*, save the **termios** structure values returned, and then do a *tcsetattr()* changing only  
46335 the necessary fields. The application should use the values saved from the *tcgetattr()* to reset the  
46336 terminal state whenever it is done with the terminal. This is necessary because terminal  
46337 attributes apply to the underlying port and not to each individual open instance; that is, all  
46338 processes that have used the terminal see the latest attribute changes.

46339 A program that uses these functions should be written to catch all signals and take other  
46340 appropriate actions to ensure that when the program terminates, whether planned or not, the  
46341 terminal device's state is restored to its original state.

46342 Existing practice dealing with error returns when only part of a request can be honored is based  
46343 on calls to the *ioctl()* function. In historical BSD and System V implementations, the  
46344 corresponding *ioctl()* returns zero if the requested actions were semantically correct, even if  
46345 some of the requested changes could not be made. Many existing applications assume this  
46346 behavior and would no longer work correctly if the return value were changed from zero to -1  
46347 in this case.



46348 Note that either specification has a problem. When zero is returned, it implies everything  
46349 succeeded even if some of the changes were not made. When -1 is returned, it implies  
46350 everything failed even though some of the changes were made.

46351 Applications that need all of the requested changes made to work properly should follow  
46352 *tcsetattr()* with a call to *tcgetattr()* and compare the appropriate field values.

46353 **FUTURE DIRECTIONS**

46354 None.

46355 **SEE ALSO**

46356 *tcsetattr()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <termios.h>, the Base  
46357 Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface

46358 **CHANGE HISTORY**

46359 First released in Issue 3.

46360 Entry included for alignment with the POSIX.1-1988 standard.

46361 **Issue 4**

46362 The FUTURE DIRECTIONS section is added to allow for alignment with the ISO POSIX-1  
46363 standard.

46364 **Issue 6**

46365 In the DESCRIPTION, the rate returned as the input baud rate shall be the output rate.  
46366 Previously, the number zero was also allowed but was obsolescent.

46367 **NAME**

46368 tcgetpgrp — get the foreground process group ID

46369 **SYNOPSIS**

46370 #include &lt;unistd.h&gt;

46371 pid\_t tcgetpgrp(int *fildev*);46372 **DESCRIPTION**46373 The *tcgetpgrp()* function shall return the value of the process group ID of the foreground process  
46374 group associated with the terminal.46375 If there is no foreground process group, *tcgetpgrp()* returns a value greater than 1 that does not  
46376 match the process group ID of any existing process group.46377 The *tcgetpgrp()* function is allowed from a process that is a member of a background process  
46378 group; however, the information may be subsequently changed by a process that is a member of  
46379 a foreground process group.46380 **RETURN VALUE**46381 Upon successful completion, *tcgetpgrp()* shall return the value of the process group ID of the  
46382 foreground process associated with the terminal. Otherwise, -1 shall be returned and *errno* set to  
46383 indicate the error.46384 **ERRORS**46385 The *tcgetpgrp()* function shall fail if:46386 [EBADF] The *fildev* argument is not a valid file descriptor. |46387 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the |  
46388 controlling terminal.46389 **EXAMPLES**

46390 None.

46391 **APPLICATION USAGE**

46392 None.

46393 **RATIONALE**

46394 None.

46395 **FUTURE DIRECTIONS**

46396 None.

46397 **SEE ALSO**46398 *setsid()*, *setpgid()*, *tcsetpgrp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |  
46399 <sys/types.h>, <unistd.h>46400 **CHANGE HISTORY**

46401 First released in Issue 3.

46402 Entry included for alignment with the POSIX.1-1988 standard.

46403 **Issue 4**46404 The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
46405 XSI-conformant systems.

46406 The &lt;unistd.h&gt; header is added to the SYNOPSIS section.

46407 The following change is incorporated for alignment with the FIPS requirements:

- 46408           • The DESCRIPTION is clarified and the phrase “If \_POSIX\_JOB\_CONTROL is defined” is  
46409           removed because job control is now mandatory on all XSI-conformant systems.

46410 **Issue 6**

46411           In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

46412           The following new requirements on POSIX implementations derive from alignment with the  
46413           Single UNIX Specification:

- 46414           • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
46415           required for conforming implementations of previous POSIX specifications, it was not  
46416           required for UNIX applications.
- 46417           • In the DESCRIPTION, text previously conditional on support for \_POSIX\_JOB\_CONTROL is  
46418           now mandatory. This is a FIPS requirement.

46419 **NAME**

46420 tcgetsid — get process group ID for session leader for controlling terminal

46421 **SYNOPSIS**

46422 XSI #include <termios.h>

46423 pid\_t tcgetsid(int *fildes*);

46424

46425 **DESCRIPTION**

46426 The *tcgetsid()* function shall obtain the process group ID of the session for which the terminal  
46427 specified by *fildes* is the controlling terminal.

46428 **RETURN VALUE**

46429 Upon successful completion, *tcgetsid()* shall return the process group ID associated with the  
46430 terminal. Otherwise, a value of (**pid\_t**)-1 shall be returned and *errno* set to indicate the error.

46431 **ERRORS**

46432 The *tcgetsid()* function shall fail if:

46433 [EBADF] The *fildes* argument is not a valid file descriptor. |

46434 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the |  
46435 controlling terminal.

46436 **EXAMPLES**

46437 None.

46438 **APPLICATION USAGE**

46439 None.

46440 **RATIONALE**

46441 None.

46442 **FUTURE DIRECTIONS**

46443 None.

46444 **SEE ALSO**

46445 The Base Definitions volume of IEEE Std. 1003.1-200x, <**termios.h**> |

46446 **CHANGE HISTORY**

46447 First released in Issue 4, Version 2.

46448 **Issue 5**

46449 Moved from X/OPEN UNIX extension to BASE.

46450 The [EACCES] error has been removed from the list of mandatory errors, and the description of  
46451 [ENOTTY] has been reworded.

46452 **NAME**

46453 tcsendbreak — send a “break” for a specific duration

46454 **SYNOPSIS**

46455 #include &lt;termios.h&gt;

46456 int tcsendbreak(int *fildev*, int *duration*);46457 **DESCRIPTION**46458 The *fildev* argument is an open file descriptor associated with a terminal.

46459 If the terminal is using asynchronous serial data transmission, *tcsendbreak()* shall cause  
46460 transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is 0, it  
46461 shall cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5  
46462 seconds. If *duration* is not 0, it shall send zero-valued bits for an implementation-defined period  
46463 of time.

46464 If the terminal is not using asynchronous serial data transmission, it is implementation-defined  
46465 whether *tcsendbreak()* sends data to generate a break condition or returns without taking any  
46466 action.

46467 Attempts to use *tcsendbreak()* from a process which is a member of a background process group  
46468 on a *fildev* associated with its controlling terminal, shall cause the process group to be sent a  
46469 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is  
46470 allowed to perform the operation, and no signal is sent.

46471 **RETURN VALUE**

46472 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46473 indicate the error.

46474 **ERRORS**46475 The *tcsendbreak()* function shall fail if:46476 [EBADF] The *fildev* argument is not a valid file descriptor.46477 [ENOTTY] The file associated with *fildev* is not a terminal.46478 The *tcsendbreak()* function may fail if:

46479 [EIO] The process group of the writing process is orphaned, and the writing process  
46480 is not ignoring or blocking SIGTTOU.

46481 **EXAMPLES**

46482 None.

46483 **APPLICATION USAGE**

46484 None.

46485 **RATIONALE**

46486 None.

46487 **FUTURE DIRECTIONS**

46488 None.

46489 **SEE ALSO**

46490 The Base Definitions volume of IEEE Std. 1003.1-200x, <termios.h>, <unistd.h>, the Base  
46491 Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface

46492 **CHANGE HISTORY**

46493 First released in Issue 3.

46494 Entry included for alignment with the POSIX.1-1988 standard.

46495 **Issue 4**

46496 The [EIO] error is added to the ERRORS section.

46497 The following change is incorporated for alignment with the FIPS requirements:

- 46498
- In the DESCRIPTION the phrase “If \_POSIX\_JOB\_CONTROL is defined” is removed
- 46499 because job control is now mandatory on all XSI-conformant systems.

46500 **Issue 6**46501 The following new requirements on POSIX implementations derive from alignment with the  
46502 Single UNIX Specification:

- 46503
- In the DESCRIPTION, text previously conditional on \_POSIX\_JOB\_CONTROL is now
- 46504 mandated. This is a FIPS requirement.
- 
- 46505
- The [EIO] error is added.

## 46506 NAME

46507 tcsetattr — set the parameters associated with the terminal

46508 **Notes to Reviewers**46509 *This section with side shading will not appear in the final copy. - Ed.*

46510 See the FUTURE DIRECTIONS section.

46511 **SYNOPSIS**

46512 #include &lt;termios.h&gt;

46513 int tcsetattr(int *fildev*, int *optional\_actions*,46514 const struct termios \**termios\_p*);46515 **DESCRIPTION**46516 The *tcsetattr()* function shall set the parameters associated with the terminal referred to by the  
46517 open file descriptor *fildev* (an open file descriptor associated with a terminal) from the **termios**  
46518 structure referenced by *termios\_p* as follows:

- 46519 • If *optional\_actions* is TCSANOW, the change shall occur immediately.
- 46520 • If *optional\_actions* is TCSADRAIN, the change shall occur after all output written to *fildev* is  
46521 transmitted. This function should be used when changing parameters that affect output.
- 46522 • If *optional\_actions* is TCSAFLUSH, the change shall occur after all output written to *fildev* is  
46523 transmitted, and all input so far received but not read shall be discarded before the change is  
46524 made.

46525 If the output baud rate stored in the **termios** structure pointed to by *termios\_p* is the zero baud  
46526 rate, B0, the modem control lines shall no longer be asserted. Normally, this shall disconnect the  
46527 line.

46528 If the input baud rate stored in the **termios** structure pointed to by *termios\_p* is 0, the input baud  
46529 rate given to the hardware is the same as the output baud rate stored in the **termios** structure.

46530 The *tcsetattr()* function shall return successfully if it was able to perform any of the requested  
46531 actions, even if some of the requested actions could not be performed. It shall set all the  
46532 attributes that the implementation supports as requested and leaves all the attributes not  
46533 supported by the implementation unchanged. If no part of the request can be honored, it shall  
46534 return  $-1$  and set *errno* to [EINVAL]. If the input and output baud rates differ and are a  
46535 combination that is not supported, neither baud rate is changed. A subsequent call to *tcsetattr()*  
46536 shall return the actual state of the terminal device (reflecting both the changes made and not  
46537 made in the previous *tcsetattr()* call). The *tcsetattr()* function shall not change the values found  
46538 in the **termios** structure under any circumstances.

46539 The effect of *tcsetattr()* is undefined if the value of the **termios** structure pointed to by *termios\_p*  
46540 was not derived from the result of a call to *tcsetattr()* on *fildev*; an application should modify  
46541 only fields and flags defined by this volume of IEEE Std. 1003.1-200x between the call to  
46542 *tcsetattr()* and *tcsetattr()*, leaving all other fields and flags unmodified.

46543 No actions defined by this volume of IEEE Std. 1003.1-200x, other than a call to *tcsetattr()* or a  
46544 close of the last file descriptor in the system associated with this terminal device, shall cause any  
46545 of the terminal attributes defined by this volume of IEEE Std. 1003.1-200x to change.

46546 If *tcsetattr()* is called from a process which is a member of a background process group on a  
46547 *fildev* associated with its controlling terminal:

- 46548 • If the calling process is blocking or ignoring SIGTTOU signals, the operation completes  
46549 normally and no signal is sent.

46550           • Otherwise, a SIGTTOU signal shall be sent to the process group.

#### 46551 RETURN VALUE

46552           Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46553           indicate the error.

#### 46554 ERRORS

46555           The *tcsetattr()* function shall fail if:

46556           [EBADF]           The *fildev* argument is not a valid file descriptor.

46557           [EINTR]           A signal interrupted *tcsetattr()*.

46558           [EINVAL]          The *optional\_actions* argument is not a supported value, or an attempt was  
46559           made to change an attribute represented in the **termios** structure to an  
46560           unsupported value.

46561           [ENOTTY]          The file associated with *fildev* is not a terminal.

46562           The *tcsetattr()* function may fail if:

46563           [EIO]              The process group of the writing process is orphaned, and the writing process  
46564           is not ignoring or blocking SIGTTOU.

#### 46565 EXAMPLES

46566           None.

#### 46567 APPLICATION USAGE

46568           If trying to change baud rates, applications should call *tcsetattr()* then call *tcgetattr()* in order to  
46569           determine what baud rates were actually selected.

#### 46570 RATIONALE

46571           The *tcsetattr()* function can be interrupted in the following situations:

- 46572           • It is interrupted while waiting for output to drain.
- 46573           • It is called from a process in a background process group and SIGTTOU is caught.

46574           See also the RATIONALE section in *tcgetattr()*.

#### 46575 FUTURE DIRECTIONS

46576           Using an input baud rate of 0 to set the input rate equal to the output rate may not necessarily be  
46577           supported in a future version of this volume of IEEE Std. 1003.1-200x.

#### 46578 SEE ALSO

46579           *cfgetispeed()*, *tcgetattr()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<termios.h>**,  
46580           **<unistd.h>**, the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal  
46581           Interface

#### 46582 CHANGE HISTORY

46583           First released in Issue 3.

46584           Entry included for alignment with the POSIX.1-1988 standard.

#### 46585 Issue 4

46586           The words “and stores them in” are changed to “from” in the first paragraph of the  
46587           DESCRIPTION.

46588           The [EINTR] and [EIO] errors are added to the ERRORS section.

46589           The FUTURE DIRECTIONS section is added to allow for alignment with the ISO POSIX-1  
46590           standard.



- 46591 The following change is incorporated for alignment with the ISO POSIX-1 standard:
- 46592     • The argument *termios\_p* is changed from type **struct termios\*** to **const struct termios\***.
- 46593 The following change is incorporated for alignment with the FIPS requirements:
- 46594     • In the DESCRIPTION the phrase “If `_POSIX_JOB_CONTROL` is defined” is removed  
46595         because job control is now mandatory on all XSI-conformant systems.
- 46596 **Issue 6**
- 46597 The following new requirements on POSIX implementations derive from alignment with the  
46598 Single UNIX Specification:
- 46599     • In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now  
46600         mandated. This is a FIPS requirement.
- 46601     • The [EIO] error is added.
- 46602 In the DESCRIPTION, the text describing use of *tcsetattr()* from a process which is a member of  
46603 a background process group is clarified.

46604 **NAME**

46605 tcsetpgrp — set the foreground process group ID

46606 **SYNOPSIS**

46607 #include &lt;unistd.h&gt;

46608 int tcsetpgrp(int *fildev*, pid\_t *pgid\_id*);46609 **DESCRIPTION**

46610 If the process has a controlling terminal, *tcsetpgrp()* shall set the foreground process group ID  
46611 associated with the terminal to *pgid\_id*. The application shall ensure that the file associated with  
46612 *fildev* is the controlling terminal of the calling process and the controlling terminal is currently  
46613 associated with the session of the calling process. The application shall ensure that the value of  
46614 *pgid\_id* matches a process group ID of a process in the same session as the calling process.

46615 Attempts to use *tcsetpgrp()* from a process which is a member of a background process group on  
46616 a *fildev* associated with its controlling terminal will cause the process group to be sent a  
46617 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is  
46618 allowed to perform the operation, and no signal is sent.

46619 **RETURN VALUE**

46620 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46621 indicate the error.

46622 **ERRORS**46623 The *tcsetpgrp()* function shall fail if:

46624 [EBADF] The *fildev* argument is not a valid file descriptor.

46625 [EINVAL] This implementation does not support the value in the *pgid\_id* argument.

46626 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the  
46627 controlling terminal, or the controlling terminal is no longer associated with  
46628 the session of the calling process.

46629 [EPERM] The value of *pgid\_id* is a value supported by the implementation, but does not  
46630 match the process group ID of a process in the same session as the calling  
46631 process.

46632 **EXAMPLES**

46633 None.

46634 **APPLICATION USAGE**

46635 None.

46636 **RATIONALE**

46637 None.

46638 **FUTURE DIRECTIONS**

46639 None.

46640 **SEE ALSO**46641 *tcgetpgrp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>, <unistd.h>46642 **CHANGE HISTORY**

46643 First released in Issue 3.

46644 Entry included for alignment with the POSIX.1-1988 standard.

46645 **Issue 4**

46646 The `<sys/types.h>` header is now marked as optional (OH); this header need not be included on  
46647 XSI-conformant systems.

46648 The `<unistd.h>` header is added to the SYNOPSIS section.

46649 The [ENOSYS] error is removed from the ERRORS section.

46650 The following change is incorporated for alignment with the FIPS requirements:

- 46651 • In the DESCRIPTION the phrase “If `_POSIX_JOB_CONTROL` is defined” is removed  
46652 because job control is now mandatory on all XSI-conformant systems.

46653 **Issue 6**

46654 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

46655 The following new requirements on POSIX implementations derive from alignment with the  
46656 Single UNIX Specification:

- 46657 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
46658 required for conforming implementations of previous POSIX specifications, it was not  
46659 required for UNIX applications.
- 46660 • In the DESCRIPTION and ERRORS sections, text previously conditional on  
46661 `_POSIX_JOB_CONTROL` is now mandated. This is a FIPS requirement.

46662 The DESCRIPTION is updated to avoid use of the term “must” for application requirements. |

46663 The Open Group corrigenda item U047/4 has been applied. |

## 46664 NAME

46665 tdelete, tfind, tsearch, twalk — manage a binary search tree

## 46666 SYNOPSIS

```
46667 XSI #include <search.h>
46668 void *tdelete(const void *restrict key, void **restrict rootp,
46669 int(*compar)(const void *, const void *));
46670 void *tfind(const void *key, void *const *rootp,
46671 int(*compar)(const void *, const void *));
46672 void *tsearch(const void *key, void **rootp,
46673 int (*compar)(const void *, const void *));
46674 void twalk(const void *root,
46675 void (*action)(const void *, VISIT, int));
46676
```

## 46677 DESCRIPTION

46678 The *tdelete()*, *tfind()*, *tsearch()*, and *twalk()* functions manipulate binary search trees.  
 46679 Comparisons are made with a user-supplied routine, the address of which is passed as the  
 46680 *compar* argument. This routine is called with two arguments, the pointers to the elements being  
 46681 compared. The application shall ensure that the user-supplied routine returns an integer less  
 46682 than, equal to, or greater than 0, according to whether the first argument is to be considered less  
 46683 than, equal to, or greater than the second argument. The comparison function need not compare  
 46684 every byte, so arbitrary data may be contained in the elements in addition to the values being  
 46685 compared.

46686 The *tsearch()* function is used to build and access the tree. The *key* argument is a pointer to an  
 46687 element to be accessed or stored. If there is a node in the tree whose element is equal to the value  
 46688 pointed to by *key*, a pointer to this found node is returned. Otherwise, the value pointed to by  
 46689 *key* is inserted (that is, a new node is created and the value of *key* is copied to this node), and a  
 46690 pointer to this node returned. Only pointers are copied, so the application shall ensure that the  
 46691 calling routine stores the data. The *rootp* argument points to a variable that points to the root  
 46692 node of the tree. A null pointer value for the variable pointed to by *rootp* denotes an empty tree;  
 46693 in this case, the variable shall be set to point to the node which shall be at the root of the new  
 46694 tree.

46695 Like *tsearch()*, *tfind()* shall search for a node in the tree, returning a pointer to it if found.  
 46696 However, if it is not found, *tfind()* shall return a null pointer. The arguments for *tfind()* are the  
 46697 same as for *tsearch()*.

46698 The *tdelete()* function deletes a node from a binary search tree. The arguments are the same as  
 46699 for *tsearch()*. The variable pointed to by *rootp* shall be changed if the deleted node was the root  
 46700 of the tree. The *tdelete()* function returns a pointer to the parent of the deleted node, or a null  
 46701 pointer if the node is not found.

46702 The *twalk()* function traverses a binary search tree. The *root* argument is a pointer to the root  
 46703 node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below  
 46704 that node.) The argument *action* is the name of a routine to be invoked at each node. This routine  
 46705 is, in turn, called with three arguments. The first argument is the address of the node being  
 46706 visited. The structure pointed to by this argument is unspecified and shall not be modified by  
 46707 the application, but it is guaranteed that a pointer-to-node can be converted to pointer-to-  
 46708 pointer-to-element to access the element stored in the node. The second argument is a value  
 46709 from an enumeration data type:

```
46710 typedef enum { preorder, postorder, endorder, leaf } VISIT;
```

46711 (defined in `<search.h>`), depending on whether this is the first, second, or third time that the  
 46712 node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a  
 46713 leaf. The third argument is the level of the node in the tree, with the root being level 0.

46714 If the calling function alters the pointer to the root, the result is undefined.

#### 46715 RETURN VALUE

46716 If the node is found, both `tsearch()` and `tfind()` shall return a pointer to it. If not, `tfind()` shall  
 46717 return a null pointer, and `tsearch()` shall return a pointer to the inserted item.

46718 A null pointer shall be returned by `tsearch()` if there is not enough space available to create a new  
 46719 node.

46720 A null pointer shall be returned by `tdelete()`, `tfind()`, and `tsearch()` if `rootp` is a null pointer on  
 46721 entry.

46722 The `tdelete()` function shall return a pointer to the parent of the deleted node, or a null pointer if  
 46723 the node is not found.

46724 The `twalk()` function shall return no value.

#### 46725 ERRORS

46726 No errors are defined.

#### 46727 EXAMPLES

46728 The following code reads in strings and stores structures containing a pointer to each string and  
 46729 a count of its length. It then walks the tree, printing out the stored strings and their lengths in  
 46730 alphabetical order.

```

46731 #include <search.h>
46732 #include <string.h>
46733 #include <stdio.h>

46734 #define STRSZ 10000
46735 #define NODSZ 500

46736 struct node { /* Pointers to these are stored in the tree. */
46737 char *string;
46738 int length;
46739 };

46740 char string_space[STRSZ]; /* Space to store strings. */
46741 struct node nodes[NODSZ]; /* Nodes to store. */
46742 void *root = NULL; /* This points to the root. */

46743 int main(int argc, char *argv[])
46744 {
46745 char *strptr = string_space;
46746 struct node *nodeptr = nodes;
46747 void print_node(const void *, VISIT, int);
46748 int i = 0, node_compare(const void *, const void *);

46749 while (gets(strptr) != NULL && i++ < NODSZ) {
46750 /* Set node. */
46751 nodeptr->string = strptr;
46752 nodeptr->length = strlen(strptr);
46753 /* Put node into the tree. */
46754 (void) tsearch((void *)nodeptr, (void **)&root,
46755 node_compare);

```

```

46756 /* Adjust pointers, so we do not overwrite tree. */
46757 strptr += nodeptr->length + 1;
46758 nodeptr++;
46759 }
46760 twalk(root, print_node);
46761 return 0;
46762 }
46763 /*
46764 * This routine compares two nodes, based on an
46765 * alphabetical ordering of the string field.
46766 */
46767 int
46768 node_compare(const void *node1, const void *node2)
46769 {
46770 return strcmp(((const struct node *) node1)->string,
46771 ((const struct node *) node2)->string);
46772 }
46773 /*
46774 * This routine prints out a node, the second time
46775 * twalk encounters it or if it is a leaf.
46776 */
46777 void
46778 print_node(const void *ptr, VISIT order, int level)
46779 {
46780 const struct node *p = *(const struct node **) ptr;
46781 if (order == postorder || order == leaf) {
46782 (void) printf("string = %s, length = %d\n",
46783 p->string, p->length);
46784 }
46785 }

```

#### 46786 APPLICATION USAGE

46787 The *root* argument to *twalk()* is one level of indirection less than the *rootp* arguments to *tdelete()*  
 46788 and *tsearch()*.

46789 There are two nomenclatures used to refer to the order in which tree nodes are visited. The  
 46790 *tsearch()* function uses **preorder**, **postorder**, and **endorder** to refer respectively to visiting a node  
 46791 before any of its children, after its left child and before its right, and after both its children. The  
 46792 alternative nomenclature uses **preorder**, **inorder**, and **postorder** to refer to the same visits, which  
 46793 could result in some confusion over the meaning of **postorder**.

#### 46794 RATIONALE

46795 None.

#### 46796 FUTURE DIRECTIONS

46797 None.

#### 46798 SEE ALSO

46799 *hcreate()*, *tsearch()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <[search.h](#)>

46800 **CHANGE HISTORY**

- 46801 First released in Issue 1. Derived from Issue 1 of the SVID.
- 46802 **Issue 4**
- 46803 The type of argument *key* is changed from **char\*** to **const void\***.
- 46804 The function return value is changed from **char\*** to **void\***.
- 46805 Arguments to *compar* are formally defined.
- 46806 The type of argument *rootp* is changed from **char\*\*** to **void\*\*** for the *tsearch()* function.
- 46807 The type of argument *rootp* is changed from **char\*\*** to **void\*const\*** for the *tfind()* function.
- 46808 The type of argument *root* is changed from **char\*** to **const void\***, and the argument list to *action* is formally defined for the *twalk()* function.
- 46809
- 46810 Various minor wording changes are made in the DESCRIPTION to improve clarity and accuracy. In particular, additional notes are added about constraints on the first argument to *twalk()*.
- 46811
- 46812
- 46813 The sample code in the EXAMPLES section is updated to use ISO C standard syntax. Also the definition of the *root* and *argv* items is changed.
- 46814
- 46815 The paragraph in the APPLICATION USAGE section about casts is removed.
- 46816 **Issue 5**
- 46817 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in previous issues.
- 46818
- 46819 **Issue 6**
- 46820 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 46821 The **restrict** keyword is added to the *tdelete()* prototype for alignment with the ISO/IEC 9899:1999 standard.
- 46822

46823 **NAME**

46824 telfdir — current location of a named directory stream

46825 **SYNOPSIS**

46826 XSI #include &lt;dirent.h&gt;

46827 long telfdir(DIR \*dirp);

46828

46829 **DESCRIPTION**46830 The *telfdir()* function obtains the current location associated with the directory stream specified  
46831 by *dirp*.46832 If the most recent operation on the directory stream was a *seekdir()*, the directory position  
46833 returned from the *telfdir()* shall be the same as that supplied as a *loc* argument for *seekdir()*.46834 **RETURN VALUE**46835 Upon successful completion, *telfdir()* shall return the current location of the specified directory  
46836 stream.46837 **ERRORS**

46838 No errors are defined.

46839 **EXAMPLES**

46840 None.

46841 **APPLICATION USAGE**

46842 None.

46843 **RATIONALE**

46844 None.

46845 **FUTURE DIRECTIONS**

46846 None.

46847 **SEE ALSO**46848 *opendir()*, *readdir()*, *seekdir()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <dirent.h>46849 **CHANGE HISTORY**

46850 First released in Issue 2.

46851 **Issue 4**

46852 The &lt;sys/types.h&gt; header is removed from the SYNOPSIS section.

46853 The function return value is expanded to **long**.46854 **Issue 4, Version 2**46855 The DESCRIPTION is updated for X/OPEN UNIX conformance to indicate that a call to *telfdir()*  
46856 immediately following a call to *seekdir()*, returns the *loc* value passed to the *seekdir()* call.



46857 **NAME**

46858 tempnam — create a name for a temporary file

46859 **SYNOPSIS**46860 XSI 

```
#include <stdio.h>
```

46861 

```
char *tempnam(const char *dir, const char *pfx);
```

46862

46863 **DESCRIPTION**46864 The *tempnam()* function generates a path name that may be used for a temporary file.

46865 The *tempnam()* function allows the user to control the choice of a directory. The *dir* argument  
46866 points to the name of the directory in which the file is to be created. If *dir* is a null pointer or  
46867 points to a string which is not a name for an appropriate directory, the path prefix defined as  
46868 P\_tmpdir in the <stdio.h> header is used. If that directory is not accessible, an implementation-  
46869 defined directory may be used.

46870 Many applications prefer their temporary files to have certain initial letter sequences in their  
46871 names. The *pfx* argument should be used for this. This argument may be a null pointer or point  
46872 to a string of up to five bytes to be used as the beginning of the file name.

46873 Some implementations of *tempnam()* may use *tmpnam()* internally. On such implementations, if  
46874 called more than {TMP\_MAX} times in a single process, the behavior is implementation-defined.

46875 **RETURN VALUE**

46876 Upon successful completion, *tempnam()* shall allocate space for a string, put the generated path  
46877 name in that space, and return a pointer to it. The pointer shall be suitable for use in a  
46878 subsequent call to *free()*. Otherwise, it shall return a null pointer and set *errno* to indicate the  
46879 error.

46880 **ERRORS**46881 The *tempnam()* function shall fail if:

46882 [ENOMEM] Insufficient storage space is available.

46883 **EXAMPLES**46884 **Generating a Path Name**

46885 The following example generates a path name for a temporary file in directory */tmp*, with the  
46886 prefix *file*. After the file name has been created, the call to *free()* deallocates the space used to  
46887 store the file name.

```
46888 #include <stdio.h>
46889 #include <stdlib.h>
46890 ...
46891 char *directory = "/tmp";
46892 char *fileprefix = "file";
46893 char *file;

46894 file = tempnam(directory, fileprefix);
46895 free(file);
```

46896 **APPLICATION USAGE**

46897 This function only creates path names. It is the application's responsibility to create and remove  
46898 the files. Between the time a path name is created and the file is opened, it is possible for some  
46899 other process to create a file with the same name. Applications may find *tmpfile()* more useful.

46900 **RATIONALE**

46901 None.

46902 **FUTURE DIRECTIONS**

46903 None.

46904 **SEE ALSO**46905 *fopen()*, *free()*, *open()*, *tmpfile()*, *tmpnam()*, *unlink()*, the Base Definitions volume of  
46906 IEEE Std. 1003.1-200x, <stdio.h>46907 **CHANGE HISTORY**

46908 First released in Issue 1. Derived from Issue 1 of the SVID.

46909 **Issue 4**46910 The type of arguments *dir* and *px* is changed from **char\*** to **const char\***.46911 The DESCRIPTION is changed to indicate that *px* is treated as a string of bytes and not as a  
46912 string of (possibly multi-byte) characters.

46913 The second paragraph of the APPLICATION USAGE section is expanded.

46914 **Issue 5**46915 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
46916 previous issues.

46917 **NAME**

46918 tfind — search binary search tree

46919 **SYNOPSIS**

46920 XSI #include &lt;search.h&gt;

46921 void \*tfind(const void \*key, void \*const \*rootp,  
46922 int (\*compar)(const void \*, const void \*));

46923

46924 **DESCRIPTION**46925 Refer to *tdelete()*.

46926 **NAME**

46927 tgamma, tgammaf, tgammal — compute gamma() function

46928 **SYNOPSIS**

46929 #include &lt;math.h&gt;

46930 double tgamma(double x);

46931 float tgammaf(float x);

46932 long double tgammal(long double x);

46933 **DESCRIPTION**

46934 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
46935 conflict between the requirements described here and the ISO C standard is unintentional. This  
46936 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

46937 These functions shall compute the *gamma()* function of *x*.

46938 An application wishing to check for error situations should set *errno* to 0 before calling these  
46939 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

46940 **RETURN VALUE**46941 Upon successful completion, these functions shall return *Gamma(x)*.46942 If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

46943 If *x* is a negative integer, or if the result cannot be represented when *x* is 0, either HUGE\_VAL or  
46944 NaN shall be returned and *errno* shall be set to [EDOM].

46945 If the magnitude of *x* is too large or too small, ±HUGE\_VAL shall be returned and *errno* shall be  
46946 set to [ERANGE].

46947 **ERRORS**

46948 These functions shall fail if:

46949 [EDOM] The value of *x* is negative or the result cannot be represented when *x* is zero.46950 [ERANGE] The magnitude of *x* is too large or too small.

46951 These functions may fail if:

46952 [EDOM] The value of *x* is NaN.46953 **EXAMPLES**

46954 None.

46955 **APPLICATION USAGE**

46956 None.

46957 **RATIONALE**

46958 This function is named *tgamma()* in order to avoid conflicts with the historical *gamma()* and  
46959 *lgamma()* functions.

46960 **FUTURE DIRECTIONS**

46961 None.

46962 **SEE ALSO**46963 *lgamma()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>46964 **CHANGE HISTORY**

46965 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

46966 **NAME**

46967           time — get time

46968 **SYNOPSIS**

46969           #include &lt;time.h&gt;

46970           time\_t time(time\_t \*tloc);

46971 **DESCRIPTION**

46972 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 46973       conflict between the requirements described here and the ISO C standard is unintentional. This  
 46974       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

46975 CX       The *time()* function returns the value of time in seconds since the Epoch.

46976       The *tloc* argument points to an area where the return value is also stored. If *tloc* is a null pointer,  
 46977       no value is stored.

46978 **RETURN VALUE**

46979       Upon successful completion, *time()* shall return the value of time. Otherwise, **(time\_t)−1** shall be  
 46980       returned.

46981 **ERRORS**

46982       No errors are defined.

46983 **EXAMPLES**46984       **Getting the Current Time**

46985       The following example uses the *time()* function to calculate the time elapsed, in seconds, since  
 46986       January 1, 1970 0:00 UTC, *localtime()* to convert that value to a broken-down time, and *asctime()*  
 46987       to convert the broken-down time values into a printable string.

46988       #include &lt;stdio.h&gt;

46989       #include &lt;time.h&gt;

46990       main()

46991       {

46992       time\_t result;

46993           result = time(NULL);

46994           printf("%s%ld secs since the Epoch\n",

46995               asctime(localtime(&amp;result)),

46996               (long)result);

46997           return(0);

46998       }

46999       This example writes the current time to *stdout* in a form like this:

47000       Wed Jun 26 10:32:15 1996

47001       835810335 secs since the Epoch

47002 **Timing an Event**

47003 The following example gets the current time, prints it out in the user's format, and prints the  
47004 number of minutes to an event being timed.

```
47005 #include <time.h>
47006 #include <stdio.h>
47007 ...
47008 time_t now;
47009 int minutes_to_event;
47010 ...
47011 time(&now);
47012 printf("The time is ");
47013 puts(asctime(localtime(&now)));
47014 printf("There are %d minutes to the event.\n",
47015 minutes_to_event);
47016 ...
```

47017 **APPLICATION USAGE**

47018 None.

47019 **RATIONALE**

47020 The *time()* function returns a value in seconds (type **time\_t**) while *times()* returns a set of values  
47021 in clock ticks (type **clock\_t**). Some historical implementations, such as 4.3 BSD, have  
47022 mechanisms capable of returning more precise times (see below). A generalized timing scheme  
47023 to unify these various timing mechanisms has been proposed but not adopted.

47024 Implementations in which **time\_t** is a 32-bit signed integer (many historical implementations)  
47025 fail in the year 2038. This issue of this volume of IEEE Std. 1003.1-200x does not address this  
47026 problem. However, the use of the **time\_t** type is mandated in order to ease the eventual fix.

47027 The use of the **<time.h>**, header instead of **<sys/types.h>**, allows compatibility with the ISO C  
47028 standard.

47029 Many historical implementations (including Version 7) and the 1984 /usr/group standard use  
47030 **long** instead of **time\_t**. This volume of IEEE Std. 1003.1-200x uses the latter type in order to  
47031 agree with the ISO C standard.

47032 4.3 BSD includes *time()* only as an alternate function to the more flexible *gettimeofday()* function.

47033 **FUTURE DIRECTIONS**

47034 In a future version of this volume of IEEE Std. 1003.1-200x, **time\_t** is likely to be required to be  
47035 capable of representing times far in the future. Whether this will be mandated as a 64-bit type or  
47036 a requirement that a specific date in the future be representable (for example, 10000 AD) is not  
47037 yet determined. Systems purchased after the approval of this volume of IEEE Std. 1003.1-200x  
47038 should be evaluated to determine whether their lifetime will extend past 2038.

47039 **SEE ALSO**

47040 *asctime()*, *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *utime()*,  
47041 the Base Definitions volume of IEEE Std. 1003.1-200x, **<time.h>**

47042 **CHANGE HISTORY**

47043 First released in Issue 1. Derived from Issue 1 of the SVID.

47044 **Issue 4**

47045 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 47046 • The RETURN VALUE section is updated to indicate that (**time\_t**)−1 is returned on error.

47047 **Issue 6**

47048 Extensions beyond the ISO C standard are now marked.

## 47049 NAME

47050 timer\_create — create a per-process timer (**REALTIME**)

## 47051 SYNOPSIS

47052 TMR #include &lt;signal.h&gt;

47053 #include &lt;time.h&gt;

47054 int timer\_create(clockid\_t *clockid*, struct sigevent \*restrict *evp*,47055 timer\_t \*restrict *timerid*);

47056

## 47057 DESCRIPTION

47058 The *timer\_create()* function shall create a per-process timer using the specified clock, *clock\_id*, as  
 47059 the timing base. The *timer\_create()* function returns, in the location referenced by *timerid*, a timer  
 47060 ID of type **timer\_t** used to identify the timer in timer requests. This timer ID shall be unique  
 47061 within the calling process until the timer is deleted. The particular clock, *clock\_id*, is defined in  
 47062 <**time.h**>. The timer whose ID is returned shall be in a disarmed state upon return from  
 47063 *timer\_create()*.

47064 The *evp* argument, if non-NULL, points to a **sigevent** structure. This structure, allocated by the  
 47065 application, defines the asynchronous notification to occur as specified in Section 2.4.1 (on page  
 47066 528) when the timer expires. If the *evp* argument is NULL, the effect is as if the *evp* argument  
 47067 pointed to a **sigevent** structure with the *sigev\_notify* member having the value SIGEV\_SIGNAL,  
 47068 the *sigev\_signo* having a default signal number, and the *sigev\_value* member having the value of  
 47069 the timer ID.

47070 Each implementation shall define a set of clocks that can be used as timing bases for per-process  
 47071 MON timers. All implementations shall support a *clock\_id* of CLOCK\_REALTIME. If the Monotonic  
 47072 Clock option is supported, implementations shall support a *clock\_id* of CLOCK\_MONOTONIC.

47073 Per-process timers shall not be inherited by a child process across a *fork()* and shall be disarmed  
 47074 and deleted by an *exec*.

47075 CPT If \_POSIX\_CPUTIME is defined, implementations shall support *clock\_id* values representing the  
 47076 CPU-time clock of the calling process.

47077 TCT If \_POSIX\_THREAD\_CPUTIME is defined, implementations shall support *clock\_id* values  
 47078 representing the CPU-time clock of the calling thread.

47079 CPT|TCT It is implementation-defined whether a *timer\_create()* function will succeed if the value defined  
 47080 by *clock\_id* corresponds to the CPU-time clock of a process or thread different from the process  
 47081 or thread invoking the function.

## 47082 RETURN VALUE

47083 If the call succeeds, *timer\_create()* shall return zero and update the location referenced by *timerid*  
 47084 to a **timer\_t**, which can be passed to the per-process timer calls. If an error occurs, the function  
 47085 shall return a value of -1 and set *errno* to indicate the error. The value of *timerid* is undefined if  
 47086 an error occurs.

## 47087 ERRORS

47088 The *timer\_create()* function shall fail if:

47089 [EAGAIN] The system lacks sufficient signal queuing resources to honor the request.

47090 [EAGAIN] The calling process has already created all of the timers it is allowed by this  
47091 implementation.

47092 [EINVAL] The specified clock ID is not defined.



47093 CPT|TCT [ENOTSUP] The implementation does not support the creation of a timer attached to the  
 47094 CPU-time clock that is specified by *clock\_id* and associated with a process or  
 47095 thread different from the process or thread invoking *timer\_create()*.

#### 47096 EXAMPLES

47097 None.

#### 47098 APPLICATION USAGE

47099 None.

#### 47100 RATIONALE

##### 47101 **Periodic Timer Overrun and Resource Allocation**

47102 The specified timer facilities may deliver realtime signals (that is, queued signals) on  
 47103 implementations that support this option. Because realtime applications cannot afford to lose  
 47104 notifications of asynchronous events, like timer expirations or asynchronous I/O completions, it  
 47105 must be possible to ensure that sufficient resources exist to deliver the signal when the event  
 47106 occurs. In general, this is not a difficulty because there is a one-to-one correspondence between a  
 47107 request and a subsequent signal generation. If the request cannot allocate the signal delivery  
 47108 resources, it can fail the call with an [EAGAIN] error.

47109 Periodic timers are a special case. A single request can generate an indeterminate number of  
 47110 signals. This is not a problem if the requesting process can service the signals as fast as they are  
 47111 generated, thus making the signal delivery resources available for delivery of subsequent  
 47112 periodic timer expiration signals. But, in general, this cannot be assured—processing of periodic  
 47113 timer signals may “overrun”; that is, subsequent periodic timer expirations may occur before the  
 47114 currently pending signal has been delivered.

47115 Also, for signals, according to the POSIX.1-1990 standard, if subsequent occurrences of a  
 47116 pending signal are generated, it is implementation-defined whether a signal is delivered for each  
 47117 occurrence. This is not adequate for some realtime applications. So a mechanism is required to  
 47118 allow applications to detect how many timer expirations were delayed without requiring an  
 47119 indefinite amount of system resources to store the delayed expirations.

47120 The specified facilities provide for an overrun count. The overrun count is defined as the number  
 47121 of extra timer expirations that occurred between the time a timer expiration signal is generated  
 47122 and the time the signal is delivered. The signal-catching function, if it is concerned with  
 47123 overruns, can retrieve this count on entry. With this method, a periodic timer only needs one  
 47124 “signal queuing resource” that can be allocated at the time of the *timer\_create()* function call.

47125 A function is defined to retrieve the overrun count so that an application need not allocate static  
 47126 storage to contain the count, and an implementation need not update this storage  
 47127 asynchronously on timer expirations. But, for some high-frequency periodic applications, the  
 47128 overhead of an additional system call on each timer expiration may be prohibitive. The  
 47129 functions, as defined, permit an implementation to maintain the overrun count in user space,  
 47130 associated with the *timerid*. The *timer\_getoverrun()* function can then be implemented as a macro  
 47131 that uses the *timerid* argument (which may just be a pointer to a user space structure containing  
 47132 the counter) to locate the overrun count with no system call overhead. Other implementations,  
 47133 less concerned with this class of applications, can avoid the asynchronous update of user space  
 47134 by maintaining the count in a system structure at the cost of the extra system call to obtain it.

47135 **Timer Expiration Signal Parameters**

47136 The Realtime Signals Extension option supports an application-specific datum that is delivered  
47137 to the extended signal handler. This value is explicitly specified by the application, along with  
47138 the signal number to be delivered, in a **sigevent** structure. The type of the application-defined  
47139 value can be either an integer constant or a pointer. This explicit specification of the value, as  
47140 opposed to always sending the timer ID, was selected based on existing practice.

47141 It is common practice for realtime applications (on non-POSIX systems or realtime extended  
47142 POSIX systems) to use the parameters of event handlers as the case label of a switch statement  
47143 or as a pointer to an application-defined data structure. Because *timer\_ids* are dynamically  
47144 allocated by the *timer\_create()* function, they can be used for neither of these functions without  
47145 additional application overhead in the signal handler; for example, to search an array of saved  
47146 timer IDs to associate the ID with a constant or application data structure.

47147 **FUTURE DIRECTIONS**

47148 None.

47149 **SEE ALSO**

47150 *clock\_getres()*, *timer\_delete()*, *timer\_getoverrun()*, the Base Definitions volume of  
47151 IEEE Std. 1003.1-200x, <**time.h**>

47152 **CHANGE HISTORY**

47153 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47154 **Issue 6**

47155 The *timer\_create()* function is marked as part of the Timers option.

47156 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
47157 implementation does not support the Timers option.

47158 CPU-time clocks are added for alignment with IEEE Std. 1003.1d-1999.

47159 The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by adding the  
47160 requirement for the CLOCK\_MONOTONIC clock under the Monotonic Clock option.

47161 The **restrict** keyword is added to the *timer\_create()* prototype for alignment with the  
47162 ISO/IEC 9899:1999 standard.

47163 **NAME**

47164 timer\_delete — delete a per-process timer (**REALTIME**)

47165 **SYNOPSIS**

```
47166 TMR #include <time.h>
```

```
47167 int timer_delete(timer_t timerid);
```

47168

47169 **DESCRIPTION**

47170 The *timer\_delete()* function deletes the specified timer, *timerid*, previously created by the  
47171 *timer\_create()* function. If the timer is armed when *timer\_delete()* is called, the behavior shall be  
47172 as if the timer is automatically disarmed before removal. The disposition of pending signals for  
47173 the deleted timer is unspecified.

47174 **RETURN VALUE**

47175 If successful, the *timer\_delete()* function shall return a value of zero. Otherwise, the function shall  
47176 return a value of  $-1$  and set *errno* to indicate the error.

47177 **ERRORS**

47178 The *timer\_delete()* function shall fail if:

47179 [EINVAL] The timer ID specified by *timerid* is not a valid timer ID.

47180 **EXAMPLES**

47181 None.

47182 **APPLICATION USAGE**

47183 None.

47184 **RATIONALE**

47185 None.

47186 **FUTURE DIRECTIONS**

47187 None.

47188 **SEE ALSO**

47189 *timer\_create()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**time.h**>

47190 **CHANGE HISTORY**

47191 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47192 **Issue 6**

47193 The *timer\_delete()* function is marked as part of the Timers option.

47194 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
47195 implementation does not support the Timers option.

## 47196 NAME

47197 timer\_getoverrun, timer\_gettime, timer\_settime — per-process timers (**REALTIME**)

## 47198 SYNOPSIS

47199 TMR #include &lt;time.h&gt;

```

47200 int timer_getoverrun(timer_t timerid);
47201 int timer_gettime(timer_t timerid, struct itimerspec *value);
47202 int timer_settime(timer_t timerid, int flags,
47203 const struct itimerspec *restrict value,
47204 struct itimerspec *restrict ovalue);
47205

```

## 47206 DESCRIPTION

47207 Only a single signal shall be queued to the process for a given timer at any point in time. When a  
 47208 timer for which a signal is still pending expires, no signal shall be queued, and a timer overrun  
 47209 RTS shall occur. When a timer expiration signal is delivered to or accepted by a process, if the  
 47210 implementation supports the Realtime Signals Extension, the *timer\_getoverrun()* function returns  
 47211 the timer expiration overrun count for the specified timer. The overrun count returned contains  
 47212 the number of extra timer expirations that occurred between the time the signal was generated  
 47213 (queued) and when it was delivered or accepted, up to but not including an implementation-  
 47214 defined maximum of {DELAYTIMER\_MAX}. If the number of such extra expirations is greater  
 47215 than or equal to {DELAYTIMER\_MAX}, then the overrun count is set to {DELAYTIMER\_MAX}.  
 47216 The value returned by *timer\_getoverrun()* applies to the most recent expiration signal delivery or  
 47217 acceptance for the timer. If no expiration signal has been delivered for the timer, or if the  
 47218 Realtime Signals Extension is not supported, the return value of *timer\_getoverrun()* is  
 47219 unspecified.

47220 The *timer\_gettime()* function shall store the amount of time until the specified timer, *timerid*,  
 47221 expires and the reload value of the timer into the space pointed to by the *value* argument. The  
 47222 *it\_value* member of this structure shall contain the amount of time before the timer expires, or  
 47223 zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if  
 47224 the timer was armed with absolute time. The *it\_interval* member of *value* shall contain the reload  
 47225 value last set by *timer\_settime()*.

47226 The *timer\_settime()* function shall set the time until the next expiration of the timer specified by  
 47227 *timerid* from the *it\_value* member of the *value* argument and arms the timer if the *it\_value*  
 47228 member of *value* is non-zero. If the specified timer was already armed when *timer\_settime()* is  
 47229 called, this call shall reset the time until next expiration to the *value* specified. If the *it\_value*  
 47230 member of *value* is zero, the timer shall be disarmed. The effect of disarming or resetting a timer  
 47231 on pending expiration notifications is unspecified.

47232 If the flag **TIMER\_ABSTIME** is not set in the argument *flags*, *timer\_settime()* behaves as if the  
 47233 time until next expiration is set to be equal to the interval specified by the *it\_value* member of  
 47234 *value*. That is, the timer shall expire in *it\_value* nanoseconds from when the call is made. If the  
 47235 flag **TIMER\_ABSTIME** is set in the argument *flags*, *timer\_settime()* behaves as if the time until  
 47236 next expiration is set to be equal to the difference between the absolute time specified by the  
 47237 *it\_value* member of *value* and the current value of the clock associated with *timerid*. That is, the  
 47238 timer shall expire when the clock reaches the value specified by the *it\_value* member of *value*. If  
 47239 the specified time has already passed, the function shall succeed and the expiration notification  
 47240 shall be made.

47241 The reload value of the timer is set to the value specified by the *it\_interval* member of *value*.  
 47242 When a timer is armed with a non-zero *it\_interval*, a periodic (or repetitive) timer is specified.

47243 Time values that are between two consecutive non-negative integer multiples of the resolution  
 47244 of the specified timer shall be rounded up to the larger multiple of the resolution. Quantization  
 47245 error shall not cause the timer to expire earlier than the rounded time value.

47246 If the argument *ovalue* is not NULL, the function *timer\_settime()* shall store, in the location  
 47247 referenced by *ovalue*, a value representing the previous amount of time before the timer would  
 47248 have expired, or zero if the timer was disarmed, together with the previous timer reload value.

#### 47249 **Notes to Reviewers**

47250 *This section with side shading will not appear in the final copy. - Ed.*

47251 D1, XSH, ERN 388 suggests rewording “are subject to the resolution of the timer” to “may have  
 47252 been rounded to the resolution of the timer”.

47253 The members of *ovalue* are subject to the resolution of the timer, and they are the same values  
 47254 that would be returned by a *timer\_gettime()* call at that point in time.

#### 47255 **RETURN VALUE**

47256 If the *timer\_getoverrun()* function succeeds, it shall return the timer expiration overrun count as  
 47257 explained above.

47258 If the *timer\_gettime()* or *timer\_settime()* functions succeed, a value of 0 shall be returned.

47259 If an error occurs for any of these functions, the value -1 shall be returned, and *errno* set to  
 47260 indicate the error.

#### 47261 **ERRORS**

47262 The *timer\_getoverrun()*, *timer\_gettime()*, and *timer\_settime()* functions shall if:

47263 [EINVAL] The *timerid* argument does not correspond to an ID returned by *timer\_create()*  
 47264 but not yet deleted by *timer\_delete()*.

47265 The *timer\_settime()* function shall fail if:

47266 [EINVAL] A *value* structure specified a nanosecond value less than zero or greater than  
 47267 or equal to 1,000 million, and the *it\_value* member of that structure did not  
 47268 specify zero seconds and nanoseconds.

#### 47269 **EXAMPLES**

47270 None.

#### 47271 **APPLICATION USAGE**

47272 None.

#### 47273 **RATIONALE**

47274 The *clock\_settime()*, *timer\_settime()*, and *nanosleep()* functions are defined to truncate specified  
 47275 time values down to the resolution supported by the implementation. Values are truncated  
 47276 when set because this appears to be existing practice, and it does not seem reasonable to require  
 47277 an error in this case. Note that this is symmetric with the truncation that occurs when reading  
 47278 the time via *clock\_gettime()* or *timer\_gettime()* at a time that is not an integral multiple of the  
 47279 clock or timer resolution.

47280 This volume of IEEE Std. 1003.1-200x defines functions that allow an application to determine  
 47281 the implementation-supported resolution for the clocks and requires an implementation to  
 47282 document the resolution supported for timers and *nanosleep()* if they differ from the supported  
 47283 clock resolution. This is more of a procurement issue than a runtime application issue.

47284 **FUTURE DIRECTIONS**

47285 None.

47286 **SEE ALSO**47287 *clock\_getres()*, *timer\_create()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**time.h**>47288 **CHANGE HISTORY**

47289 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47290 **Issue 6**47291 The *timer\_getoverrun()*, *timer\_gettime()*, and *timer\_settime()* functions are marked as part of the  
47292 Timers option.47293 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
47294 implementation does not support the Timers option.47295 The [EINVAL] error condition is updated to include the following: “and the *it\_value* member of  
47296 that structure did not specify zero seconds and nanoseconds.” This change is for IEEE PASC  
47297 Interpretation 1003.1 #89.47298 The DESCRIPTION for *timer\_getoverrun()* is updated to clarify that “If no expiration signal has  
47299 been delivered for the timer, or if the Realtime Signals Extension is not supported, the return  
47300 value of *timer\_getoverrun()* is unspecified”.47301 The **restrict** keyword is added to the *timer\_settime()* prototype for alignment with the  
47302 ISO/IEC 9899:1999 standard.

47303 **NAME**

47304 times — get process and waited-for child process times

47305 **SYNOPSIS**

47306 #include &lt;sys/times.h&gt;

47307 clock\_t times(struct tms \*buffer);

47308 **DESCRIPTION**47309 The *times()* function shall fill the **tms** structure pointed to by *buffer* with time-accounting  
47310 information. The structure **tms** is defined in <sys/times.h>.

47311 All times are measured in terms of the number of clock ticks used.

47312 The times of a terminated child process are included in the *tms\_cutime* and *tms\_cstime* elements  
47313 of the parent when *wait()* or *waitpid()* returns the process ID of this terminated child. If a child  
47314 process has not waited for its children, their times shall not be included in its times.47315 • The *tms\_utime* structure member is the CPU time charged for the execution of user  
47316 instructions of the calling process.47317 • The *tms\_stime* structure member is the CPU time charged for execution by the system on  
47318 behalf of the calling process.47319 • The *tms\_cutime* structure member is the sum of the *tms\_utime* and *tms\_cutime* times of the  
47320 child processes.47321 • The *tms\_cstime* structure member is the sum of the *tms\_stime* and *tms\_cstime* times of the child  
47322 processes.47323 **RETURN VALUE**47324 Upon successful completion, *times()* shall return the elapsed real time, in clock ticks, since an  
47325 arbitrary point in the past (for example, system start-up time). This point does not change from  
47326 one invocation of *times()* within the process to another. The return value may overflow the  
47327 possible range of type **clock\_t**. If *times()* fails, **(clock\_t)-1** shall be returned and *errno* set to  
47328 indicate the error.47329 **ERRORS**

47330 No errors are defined.

47331 **EXAMPLES**47332 **Timing a Database Lookup**47333 The following example defines two functions, *start\_clock()* and *end\_clock()*, that are used to time  
47334 a lookup. It also defines variables of type **clock\_t** and **tms** to measure the duration of  
47335 transactions. The *start\_clock()* function saves the beginning times given by the *times()* function.  
47336 The *end\_clock()* function gets the ending times and prints the difference between the two times.47337 #include <sys/times.h>  
47338 #include <stdio.h>  
47339 ...  
47340 void start\_clock(void);  
47341 void end\_clock(char \*msg);  
47342 ...  
47343 static clock\_t st\_time;  
47344 static clock\_t en\_time;  
47345 static struct tms st\_cpu;  
47346 static struct tms en\_cpu;

```

47347 ...
47348 void
47349 start_clock()
47350 {
47351 st_time = times(&st_cpu);
47352 }
47353 void
47354 end_clock(char *msg)
47355 {
47356 en_time = times(&en_cpu);
47357
47358 printf(msg);
47359 printf("Real Time: %ld, User Time %ld, System Time %ld\n",
47360 en_time - st_time,
47361 en_cpu.tms_utime - st_cpu.tms_utime,
47362 en_cpu.tms_stime - st_cpu.tms_stime);
47362 }

```

#### 47363 APPLICATION USAGE

47364 Applications should use `sysconf(_SC_CLK_TCK)` to determine the number of clock ticks per  
47365 second as it may vary from system to system.

#### 47366 RATIONALE

47367 The accuracy of the times reported is intentionally left unspecified to allow implementations  
47368 flexibility in design, from uniprocessor to multiprocessor networks.

47369 The inclusion of times of child processes is recursive, so that a parent process may collect the  
47370 total times of all of its descendants. But the times of a child are only added to those of its parent  
47371 when its parent successfully waits on the child. Thus, it is not guaranteed that a parent process  
47372 can always see the total times of all its descendants; see also the discussion of the term *realtime* in  
47373 *alarm()*.

47374 If the type `clock_t` is defined to be a signed 32-bit integer, it overflows in somewhat more than a  
47375 year if there are 60 clock ticks per second, or less than a year if there are 100. There are individual  
47376 systems that run continuously for longer than that. This volume of IEEE Std. 1003.1-200x permits  
47377 an implementation to make the reference point for the returned value be the start-up time of the  
47378 process, rather than system start-up time.

47379 The term *charge* in this context has nothing to do with billing for services. The operating system  
47380 accounts for time used in this way. That information must be correct, regardless of how that  
47381 information is used.

#### 47382 FUTURE DIRECTIONS

47383 None.

#### 47384 SEE ALSO

47385 `exec`, `fork()`, `sysconf()`, `time()`, `wait()`, the Base Definitions volume of IEEE Std. 1003.1-200x,  
47386 `<sys/times.h>`

#### 47387 CHANGE HISTORY

47388 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 47389 Issue 4

47390 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 47391 • All references to the constant `CLK_TCK` are removed.



47392

- The RETURN VALUE section is updated to indicate that **(clock\_t)-1** is returned on error.

47393 **NAME**

47394            timezone — difference from UTC and local standard time

47395 **SYNOPSIS**

47396            #include <time.h>

47397 XSI        extern long timezone;

47398

47399 **DESCRIPTION**

47400            Refer to *tzset()*.

47401 **NAME**

47402 tmpfile — create a temporary file

47403 **SYNOPSIS**

47404 #include &lt;stdio.h&gt;

47405 FILE \*tmpfile(void);

47406 **DESCRIPTION**

47407 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 47408 conflict between the requirements described here and the ISO C standard is unintentional. This  
 47409 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

47410 The *tmpfile()* function shall create a temporary file and open a corresponding stream. The file is  
 47411 automatically deleted when all references to the file are closed. The file is opened as in *fopen()*  
 47412 for update (w+).

47413 CX The largest value that can be represented correctly in an object of type **off\_t** is established as the  
 47414 offset maximum in the open file description.

47415 In some implementations, a permanent file may be left behind if the process calling *tmpfile()* is  
 47416 killed while it is processing a call to *tmpfile()*.

47417 An error message may be written to standard error if the stream cannot be opened.

47418 **RETURN VALUE**

47419 Upon successful completion, *tmpfile()* shall return a pointer to the stream of the file that is  
 47420 CX created. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

47421 **ERRORS**47422 The *tmpfile()* function shall fail if:

47423 CX [EINTR] A signal was caught during *tmpfile()*.

47424 CX [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

47425 CX [ENFILE] The maximum allowable number of files is currently open in the system.

47426 CX [ENOSPC] The directory or file system which would contain the new file cannot be  
 47427 expanded.

47428 CX [EOVERFLOW] The file is a regular file and the size of the file cannot be represented correctly  
 47429 in an object of type **off\_t**.

47430 The *tmpfile()* function may fail if:

47431 CX [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

47432 CX [ENOMEM] Insufficient storage space is available.

47433 **EXAMPLES**47434 **Creating a Temporary File**

47435 The following example creates a temporary file for update, and returns a pointer to a stream for  
 47436 the created file in the *fp* variable.

47437 #include &lt;stdio.h&gt;

47438 ...

47439 FILE \*fp;

47440 fp = tmpfile ();

47441 **APPLICATION USAGE**

47442 It should be possible to open at least {TMP\_MAX} temporary files during the lifetime of the  
47443 program (this limit may be shared with *tmpnam()*) and there should be no limit on the number  
47444 simultaneously open other than this limit and any limit on the number of open files  
47445 ({FOPEN\_MAX}).

47446 **RATIONALE**

47447 None.

47448 **FUTURE DIRECTIONS**

47449 None.

47450 **SEE ALSO**

47451 *open()*, *tmpnam()*, *unlink()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**stdio.h**>

47452 **CHANGE HISTORY**

47453 First released in Issue 1. Derived from Issue 1 of the SVID.

47454 **Issue 4**

47455 The argument list is explicitly defined as **void**.

47456 The [EINTR] error is moved to the “fails” part of the ERRORS section; [EMFILE], [ENFILE], and  
47457 [ENOSPC] are no longer marked as extensions; [EACCES], [ENOTDIR], and [EROFS] are  
47458 removed; and the [EMFILE] error in the “may fail” part is marked as an extension.

47459 **Issue 5**

47460 Large File Summit extensions are added.

47461 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
47462 in previous issues.

47463 **Issue 6**

47464 Extensions beyond the ISO C standard are now marked.

47465 The following new requirements on POSIX implementations derive from alignment with the  
47466 Single UNIX Specification:

- 47467 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file  
47468 description. This change is to support large files.
- 47469 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support  
47470 large files.
- 47471 • The [EMFILE] optional error condition is added.

47472 The APPLICATION USAGE section is added for alignment with the ISO/IEC 9899:1999  
47473 standard.

47474 **NAME**

47475 tmpnam — create a name for a temporary file

47476 **SYNOPSIS**

47477 #include <stdio.h>

47478 char \*tmpnam(char \*s);

47479 **DESCRIPTION**

47480 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
47481 conflict between the requirements described here and the ISO C standard is unintentional. This  
47482 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

47483 The *tmpnam()* function shall generate a string that is a valid file name and that is not the same as  
47484 the name of an existing file. The function is potentially capable of generating {TMP\_MAX}  
47485 different strings, but any or all of them may already be in use by existing files and thus not be  
47486 suitable return values.

47487 The *tmpnam()* function generates a different string each time it is called from the same process,  
47488 up to {TMP\_MAX} times. If it is called more than {TMP\_MAX} times, the behavior is  
47489 implementation-defined.

47490 The implementation shall behave as if no function defined in this volume of  
47491 IEEE Std. 1003.1-200x calls *tmpnam()*.

47492 CX If the application uses any of the functions guaranteed to be available if either  
47493 `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS` is defined, the application shall  
47494 ensure that the *tmpnam()* function is called with a non-NULL parameter.

47495 **RETURN VALUE**

47496 Upon successful completion, *tmpnam()* shall return a pointer to a string. If no suitable string can  
47497 be generated, the *tmpnam()* function shall return a null pointer.

47498 If the argument *s* is a null pointer, *tmpnam()* shall leave its result in an internal static object and  
47499 return a pointer to that object. Subsequent calls to *tmpnam()* may modify the same object. If the  
47500 argument *s* is not a null pointer, it is presumed to point to an array of at least {L\_tmpnam} chars;  
47501 *tmpnam()* writes its result in that array and returns the argument as its value.

47502 **ERRORS**

47503 No errors are defined.

47504 **EXAMPLES**47505 **Generating a File Name**

47506 The following example generates a unique file name and stores it in the array pointed to by *ptr*.

```
47507 #include <stdio.h>
47508 ...
47509 char filename[L_tmpnam+1];
47510 char *ptr;

47511 ptr = tmpnam(filename);
```

47512 **APPLICATION USAGE**

47513 This function only creates file names. It is the application's responsibility to create and remove  
47514 the files.

47515 Between the time a path name is created and the file is opened, it is possible for some other  
47516 process to create a file with the same name. Applications may find *tmpfile()* more useful.

47517 **RATIONALE**

47518 None.

47519 **FUTURE DIRECTIONS**

47520 None.

47521 **SEE ALSO**47522 *fopen()*, *open()*, *tmpnam()*, *tmpfile()*, *unlink()*, the Base Definitions volume of |

47523 IEEE Std. 1003.1-200x, &lt;stdio.h&gt; |

47524 **CHANGE HISTORY**

47525 First released in Issue 1. Derived from Issue 1 of the SVID. |

47526 **Issue 5**

47527 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

47528 **Issue 6**

47529 Extensions beyond the ISO C standard are now marked.

47530 The DESCRIPTION is updated to avoid use of the term “must” for application requirements. |

47531 The DESCRIPTION is expanded for alignment with the ISO/IEC 9899:1999 standard. |

47532 **NAME**

47533           toascii — translate integer to a 7-bit ASCII character

47534 **SYNOPSIS**

47535 xSI       #include &lt;ctype.h&gt;

47536           int toascii(int c);

47537

47538 **DESCRIPTION**47539           The *toascii()* function shall convert its argument into a 7-bit ASCII character.47540 **RETURN VALUE**47541           The *toascii()* function shall return the value (*c* &0x7f).47542 **ERRORS**

47543           No errors are returned.

47544 **EXAMPLES**

47545           None.

47546 **APPLICATION USAGE**

47547           None.

47548 **RATIONALE**

47549           None.

47550 **FUTURE DIRECTIONS**

47551           None.

47552 **SEE ALSO**47553           *isascii()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>47554 **CHANGE HISTORY**

47555           First released in Issue 1. Derived from Issue 1 of the SVID.

47556 **NAME**

47557           tolower — transliterate uppercase characters to lowercase

47558 **SYNOPSIS**

47559           #include &lt;ctype.h&gt;

47560           int tolower(int c);

47561 **DESCRIPTION**

47562 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
47563 conflict between the requirements described here and the ISO C standard is unintentional. This  
47564 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

47565       The *tolower()* function has as a domain a type **int**, the value of which is representable as an  
47566 **unsigned char** or the value of EOF. If the argument has any other value, the behavior is  
47567 undefined. If the argument of *tolower()* represents an uppercase letter, and there exists a  
47568 **CX**       corresponding lowercase letter (as defined by character type information in the program locale  
47569 category *LC\_CTYPE*), the result is the corresponding lowercase letter. All other arguments in the  
47570 domain are returned unchanged.

47571 **RETURN VALUE**

47572       Upon successful completion, *tolower()* shall return the lowercase letter corresponding to the  
47573 argument passed; otherwise, it shall return the argument unchanged.

47574 **ERRORS**

47575       No errors are defined.

47576 **EXAMPLES**

47577       None.

47578 **APPLICATION USAGE**

47579       None.

47580 **RATIONALE**

47581       None.

47582 **FUTURE DIRECTIONS**

47583       None.

47584 **SEE ALSO**

47585       *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>, the Base Definitions  
47586 volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

47587 **CHANGE HISTORY**

47588       First released in Issue 1. Derived from Issue 1 of the SVID.

47589 **Issue 4**

47590       Reference to “shift information” is replaced by “character type information”.

47591       The RETURN VALUE section is added.

47592 **Issue 6**

47593       Extensions beyond the ISO C standard are now marked.



47594 **NAME**

47595           toupper — transliterate lowercase characters to uppercase

47596 **SYNOPSIS**

47597           #include &lt;ctype.h&gt;

47598           int toupper(int c);

47599 **DESCRIPTION**47600 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
47601       conflict between the requirements described here and the ISO C standard is unintentional. This  
47602       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.47603       The *toupper()* function has as a domain a type **int**, the value of which is representable as an  
47604       **unsigned char** or the value of EOF. If the argument has any other value, the behavior is  
47605       undefined. If the argument of *toupper()* represents a lowercase letter, and there exists a  
47606 cx       corresponding uppercase letter (as defined by character type information in the program locale  
47607       category *LC\_CTYPE*), the result is the corresponding uppercase letter. All other arguments in the  
47608       domain are returned unchanged.47609 **RETURN VALUE**47610       Upon successful completion, *toupper()* shall return the uppercase letter corresponding to the  
47611       argument passed.47612 **ERRORS**

47613       No errors are defined.

47614 **EXAMPLES**

47615       None.

47616 **APPLICATION USAGE**

47617       None.

47618 **RATIONALE**

47619       None.

47620 **FUTURE DIRECTIONS**

47621       None.

47622 **SEE ALSO**47623       *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <ctype.h>, the Base Definitions  
47624       volume of IEEE Std. 1003.1-200x, Chapter 7, Locale47625 **CHANGE HISTORY**

47626       First released in Issue 1. Derived from Issue 1 of the SVID.

47627 **Issue 4**

47628       Reference to “shift information” is replaced by “character type information”.

47629       The RETURN VALUE section is added.

47630 **Issue 6**

47631       Extensions beyond the ISO C standard are now marked.

47632 **NAME**

47633 towctrans — character transliteration

47634 **SYNOPSIS**

47635 #include &lt;wctype.h&gt;

47636 wint\_t towctrans(wint\_t *wc*, wctrans\_t *desc*);47637 **DESCRIPTION**

47638 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 47639 conflict between the requirements described here and the ISO C standard is unintentional. This  
 47640 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

47641 The *towctrans()* function transliterates the wide-character code *wc* using the mapping described  
 47642 by *desc*. The current setting of the *LC\_CTYPE* category should be the same as during the call to  
 47643 CX *wctrans()* that returned the value *desc*. If the value of *desc* is invalid (that is, not obtained by a  
 47644 call to *wctrans()* or *desc* is invalidated by a subsequent call to *setlocale()* that has affected  
 47645 category *LC\_CTYPE*), the result is unspecified.

47646 An application wishing to check for error situations should set *errno* to 0 before calling  
 47647 *towctrans()*. If *errno* is non-zero on return, an error has occurred.

47648 **RETURN VALUE**

47649 If successful, the *towctrans()* function shall return the mapped value of *wc* using the mapping  
 47650 described by *desc*. Otherwise, it shall return *wc* unchanged.

47651 **ERRORS**47652 The *towctrans()* function may fail if:47653 CX [EINVAL] *desc* contains an invalid transliteration descriptor.47654 **EXAMPLES**

47655 None.

47656 **APPLICATION USAGE**

47657 The strings "tolower" and "toupper" are reserved for the standard mapping names. In the  
 47658 table below, the functions in the left column are equivalent to the functions in the right column.

47659 tolower(*wc*) towctrans(*wc*, wctrans("tolower"))47660 toupper(*wc*) towctrans(*wc*, wctrans("toupper"))47661 **RATIONALE**

47662 None.

47663 **FUTURE DIRECTIONS**

47664 None.

47665 **SEE ALSO**

47666 *tolower()*, *toupper()*, *wctrans()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
 47667 <wctype.h>

47668 **CHANGE HISTORY**

47669 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

47670 **Issue 6**

47671 Extensions beyond the ISO C standard are now marked.

47672 **NAME**

47673 tolower — transliterate uppercase wide-character code to lowercase

47674 **SYNOPSIS**

47675 #include <wctype.h>

47676 wint\_t tolower(wint\_t wc);

47677 **DESCRIPTION**

47678 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
47679 conflict between the requirements described here and the ISO C standard is unintentional. This  
47680 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

47681 The *tolower()* function has as a domain a type **wint\_t**, the value of which the application shall  
47682 ensure is a character representable as a **wchar\_t**, and a wide-character code corresponding to a  
47683 valid character in the current locale or the value of WEOF. If the argument has any other value,  
47684 the behavior is undefined. If the argument of *tolower()* represents an uppercase wide-character  
47685 code, and there exists a corresponding lowercase wide-character code (as defined by character  
47686 type information in the program locale category *LC\_CTYPE*), the result is the corresponding  
47687 lowercase wide-character code. All other arguments in the domain are returned unchanged.

47688 **RETURN VALUE**

47689 Upon successful completion, *tolower()* shall return the lowercase letter corresponding to the  
47690 argument passed; otherwise, it shall return the argument unchanged.

47691 **ERRORS**

47692 No errors are defined.

47693 **EXAMPLES**

47694 None.

47695 **APPLICATION USAGE**

47696 None.

47697 **RATIONALE**

47698 None.

47699 **FUTURE DIRECTIONS**

47700 None.

47701 **SEE ALSO**

47702 *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**wctype.h**>, <**wchar.h**>, the  
47703 Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

47704 **CHANGE HISTORY**

47705 First released in Issue 4.

47706 **Issue 5**

47707 The following change has been made in this issue for alignment with  
47708 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 47709 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
47710 now made visible by inclusion of the header <**wctype.h**> rather than <**wchar.h**>.

47711 **Issue 6**

47712 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

47713 **NAME**

47714 towupper — transliterate lowercase wide-character code to uppercase

47715 **SYNOPSIS**

47716 #include &lt;wctype.h&gt;

47717 wint\_t towupper(wint\_t wc);

47718 **DESCRIPTION**

47719 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
47720 conflict between the requirements described here and the ISO C standard is unintentional. This  
47721 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

47722 The *towupper()* function has as a domain a type **wint\_t**, the value of which the application shall  
47723 ensure is a character representable as a **wchar\_t**, and a wide-character code corresponding to a  
47724 valid character in the current locale or the value of WEOF. If the argument has any other value,  
47725 the behavior is undefined. If the argument of *towupper()* represents a lowercase wide-character  
47726 code, and there exists a corresponding uppercase wide-character code (as defined by character  
47727 type information in the program locale category *LC\_CTYPE*), the result is the corresponding  
47728 uppercase wide-character code. All other arguments in the domain are returned unchanged.

47729 **RETURN VALUE**

47730 Upon successful completion, *towupper()* shall return the uppercase letter corresponding to the  
47731 argument passed. Otherwise, it shall return the argument unchanged.

47732 **ERRORS**

47733 No errors are defined.

47734 **EXAMPLES**

47735 None.

47736 **APPLICATION USAGE**

47737 None.

47738 **RATIONALE**

47739 None.

47740 **FUTURE DIRECTIONS**

47741 None.

47742 **SEE ALSO**

47743 *setlocale()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**wctype.h**>, <**wchar.h**>, the  
47744 Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale

47745 **CHANGE HISTORY**

47746 First released in Issue 4.

47747 **Issue 5**

47748 The following change has been made in this issue for alignment with  
47749 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 47750 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
47751 now made visible by inclusion of the header <**wctype.h**> rather than <**wchar.h**>.

47752 **Issue 6**

47753 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

47754 **NAME**

47755 trunc, truncf, trunc1 — round to truncated integer value

47756 **SYNOPSIS**

47757 #include <math.h>

47758 double trunc(double x);

47759 float truncf(float x);

47760 long double trunc1(long double x);

47761 **DESCRIPTION**

47762 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
47763 conflict between the requirements described here and the ISO C standard is unintentional. This  
47764 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

47765 These functions shall round their argument to the integer value, in floating format, nearest to but  
47766 no larger in magnitude than the argument.

47767 An application wishing to check for error situations should set *errno* to 0 before calling these  
47768 functions. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

47769 **RETURN VALUE**

47770 Upon successful completion, these functions shall return the truncated integer value.

47771 If *x* is  $\pm\text{Inf}$ , these functions shall return *x*.

47772 If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

47773 **ERRORS**

47774 These functions may fail if:

47775 [EDOM] The value of *x* is NaN.

47776 **EXAMPLES**

47777 None.

47778 **APPLICATION USAGE**

47779 None.

47780 **RATIONALE**

47781 None.

47782 **FUTURE DIRECTIONS**

47783 None.

47784 **SEE ALSO**

47785 The Base Definitions volume of IEEE Std. 1003.1-200x, <math.h>

47786 **CHANGE HISTORY**

47787 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

47788 **NAME**

47789 truncate — truncate a file to a specified length

47790 **SYNOPSIS**

47791 xSI #include &lt;unistd.h&gt;

47792 int truncate(const char \*path, off\_t length);

47793

47794 **DESCRIPTION**47795 The *truncate()* function shall cause the regular file named by *path* to have a size which shall be  
47796 equal to *length* bytes.47797 If the file previously was larger than *length*, the extra data is discarded. If the file was previously  
47798 shorter than *length*, its size is increased, and the extended area appears as if it were zero-filled.

47799 The application shall ensure that the process has write permission for the file.

47800 If the request would cause the file size to exceed the soft file size limit for the process, the  
47801 request shall fail and the implementation shall generate the SIGXFSZ signal for the process.47802 This function shall not modify the file offset for any open file descriptions associated with the  
47803 file. Upon successful completion, if the file size is changed, this function shall mark for update  
47804 the *st\_ctime* and *st\_mtime* fields of the file, and the S\_ISUID and S\_ISGID bits of the file mode  
47805 may be cleared.47806 **RETURN VALUE**47807 Upon successful completion, *truncate()* shall return 0. Otherwise, -1 shall be returned, and *errno*  
47808 set to indicate the error.47809 **ERRORS**47810 The *truncate()* function shall fail if:

47811 [EINTR] A signal was caught during execution.

47812 [EINVAL] The *length* argument was less than 0.

47813 [EFBIG] or [EINVAL]

47814 The *length* argument was greater than the maximum file size.

47815 [EIO] An I/O error occurred while reading from or writing to a file system.

47816 [EACCES] A component of the path prefix denies search permission, or write permission  
47817 is denied on the file.

47818 [EISDIR] The named file is a directory.

47819 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
47820 argument.

47821 [ENAMETOOLONG]

47822 The length of the *path* argument exceeds {PATH\_MAX} or a path name  
47823 component is longer than {NAME\_MAX}.47824 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.47825 [ENOTDIR] A component of the path prefix of *path* is not a directory.

47826 [EROFS] The named file resides on a read-only file system.

47827 The *truncate()* function may fail if:

47828 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
47829 resolution of the *path* argument.

47830 [ENAMETOOLONG]  
47831 Path name resolution of a symbolic link produced an intermediate result  
47832 whose length exceeds {PATH\_MAX}.

47833 **EXAMPLES**  
47834 None.

47835 **APPLICATION USAGE**  
47836 None.

47837 **RATIONALE**  
47838 None.

47839 **FUTURE DIRECTIONS**  
47840 None.

47841 **SEE ALSO**  
47842 *open()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>

47843 **CHANGE HISTORY**  
47844 First released in Issue 4, Version 2.

47845 **Issue 5**  
47846 Moved from X/OPEN UNIX extension to BASE.  
47847 Large File Summit extensions are added.

47848 **Issue 6**  
47849 This reference page is split out from the *truncate()* reference page.  
47850 The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`. This  
47851 is since behavior may vary from one file system to another.  
47852 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.  
47853 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
47854 [ELOOP] error condition is added.

47855 **NAME**

47856           tsearch — search a binary search tree

47857 **SYNOPSIS**

```
47858 xSI #include <search.h>
```

```
47859 void *tsearch(const void *key, void **rootp,
47860 int (*compar)(const void *, const void *));
```

47861

47862 **DESCRIPTION**

47863           Refer to *tdelete()*.



47864 **NAME**

47865 ttyname, ttyname\_r — find path name of a terminal

47866 **SYNOPSIS**

47867 #include &lt;unistd.h&gt;

47868 char \*ttyname(int *fd*);47869 TSF int ttyname\_r(int *fd*, char \**name*, size\_t *namesize*);

47870

47871 **DESCRIPTION**

47872 The `ttyname()` function shall return a pointer to a string containing a null-terminated path name  
 47873 of the terminal associated with file descriptor *fd*. The return value may point to static data  
 47874 whose content is overwritten by each call.

47875 The `ttyname()` function need not be reentrant. A function that is not required to be reentrant is  
 47876 not required to be thread-safe.

47877 TSF The `ttyname_r()` function stores the null-terminated path name of the terminal associated with  
 47878 the file descriptor *fd* in the character array referenced by *name*. The array is *namesize*  
 47879 characters long and should have space for the name and the terminating null character. The  
 47880 maximum length of the terminal name is {TTY\_NAME\_MAX}.

47881 **RETURN VALUE**

47882 Upon successful completion, `ttyname()` shall return a pointer to a string. Otherwise, a null  
 47883 pointer shall be returned and *errno* set to indicate the error.

47884 TSF If successful, the `ttyname_r()` function shall return zero. Otherwise, an error number shall be  
 47885 returned to indicate the error.

47886 **ERRORS**47887 The `ttyname()` function may fail if:47888 [EBADF] The *fd* argument is not a valid file descriptor.47889 [ENOTTY] The *fd* argument does not refer to a terminal.47890 The `ttyname_r()` function may fail if:47891 TSF [EBADF] The *fd* argument is not a valid file descriptor.47892 TSF [ENOTTY] The *fd* argument does not refer to a terminal.47893 **Notes to Reviewers**47894 *This section with side shading will not appear in the final copy. - Ed.*

47895 D1, XSH, ERN 397 points out an inconsistency for the [ERANGE] error  
 47896 condition below and suggests this be changed to [E2BIG].

47897 TSF [ERANGE] The value of *namesize* is smaller than the length of the string to be returned  
 47898 including the terminating null character.

47899 **EXAMPLES**

47900 None.

47901 **APPLICATION USAGE**

47902 None.

47903 **RATIONALE**

47904 The term *terminal* is used instead of the historical term *terminal device* in order to avoid a  
 47905 reference to an undefined term.

47906 The thread-safe version places the terminal name in a user-supplied buffer and returns a non-  
 47907 zero value if it fails. The non-thread-safe version may return the name in a static data area that  
 47908 may be overwritten by each call.

47909 **FUTURE DIRECTIONS**

47910 None.

47911 **SEE ALSO**

47912 The Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>

47913 **CHANGE HISTORY**

47914 First released in Issue 1. Derived from Issue 1 of the SVID.

47915 **Issue 4**

47916 The <**unistd.h**> header is added to the SYNOPSIS.

47917 The statement indicating that *errno* is set on error in the RETURN VALUE section, and the errors  
 47918 [EBADF] and [ENOTTY], are marked as extensions.

47919 **Issue 5**

47920 The *ttyname\_r()* function is included for alignment with the POSIX Threads Extension.

47921 A note indicating that the *ttyname()* function need not be reentrant is added to the  
 47922 DESCRIPTION.

47923 **Issue 6**

47924 The *ttyname\_r()* function is marked as part of the Thread-Safe Functions option.

47925 The following new requirements on POSIX implementations derive from alignment with the  
 47926 Single UNIX Specification:

- 47927 • The statement that *errno* is set on error is added.
- 47928 • The [EBADF] and [ENOTTY] optional error conditions are added.

47929 **NAME**

47930 twalk — traverse a binary search tree

47931 **SYNOPSIS**

47932 XSI #include &lt;search.h&gt;

47933 void twalk(const void \*root,  
47934 void (\*action)(const void \*, VISIT, int ));  
4793547936 **DESCRIPTION**47937 Refer to *tdelete()*.

47938 **NAME**

47939           tzname — timezone strings

47940 **SYNOPSIS**

47941           #include &lt;time.h&gt;

47942           extern char \*tzname[2];

47943 **DESCRIPTION**47944           Refer to *tzset()*.

47945 **NAME**

47946 daylight, timezone, tzname, tzset — set timezone conversion information

47947 **SYNOPSIS**

47948 #include &lt;time.h&gt;

47949 XSI extern int daylight;

47950 extern long timezone;

47951 extern char \*tzname[2];

47952 void tzset(void);

47953 **DESCRIPTION**

47954 The *tzset()* function uses the value of the environment variable *TZ* to set time conversion information used by *ctime()*, *localtime()*, *mktime()*, and *strftime()*. If *TZ* is absent from the environment, implementation-defined default timezone information is used.

47957 The *tzset()* function shall set the external variable *tzname* as follows:

47958 tzname[0] = "std";

47959 tzname[1] = "dst";

47960 where *std* and *dst* are as described in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables.

47962 XSI The *tzset()* function also shall set the external variable *daylight* to 0 if Daylight Savings Time conversions should never be applied for the timezone in use; otherwise, non-zero. The external variable *timezone* shall be set to the difference, in seconds, between Coordinated Universal Time (UTC) and local standard time.

47966 **RETURN VALUE**47967 The *tzset()* function shall return no value.47968 **ERRORS**

47969 No errors are defined.

47970 **EXAMPLES**

47971 Example TZ variables and their timezone differences are given in the table below:

47972

|       | <b>TZ</b> | <i>timezone</i> |
|-------|-----------|-----------------|
| 47973 |           |                 |
| 47974 | EST       | 5*60*60         |
| 47975 | GMT       | 0*60*60         |
| 47976 | JST       | -9*60*60        |
| 47977 | MET       | -1*60*60        |
| 47978 | MST       | 7*60*60         |
| 47979 | PST       | 8*60*60         |

47980 **APPLICATION USAGE**

47981 None.

47982 **RATIONALE**

47983 None.

47984 **FUTURE DIRECTIONS**

47985 None.

47986 **SEE ALSO**

47987 *ctime()*, *localtime()*, *mktime()*, *strptime()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
47988 <**time.h**>

47989 **CHANGE HISTORY**

47990 First released in Issue 1. Derived from Issue 1 of the SVID.

47991 **Issue 4**

47992 The reference to *timezone* in the SYNOPSIS section is marked as an extension.

47993 The type of *timezone* is expanded to **extern long**.

47994 The <**time.h**> header is added to the SYNOPSIS section.

47995 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 47996
- The argument list is explicitly defined as **void**.

47997 **NAME**

47998            ualarm — set the interval timer

47999 **SYNOPSIS**

48000 XSI        #include &lt;unistd.h&gt;

48001            useconds\_t ualarm(useconds\_t *useconds*, useconds\_t *interval*);

48002

48003 **DESCRIPTION**

48004        The *ualarm()* function shall cause the SIGALRM signal to be generated for the calling process  
 48005        after the number of realtime microseconds specified by the *useconds* argument has elapsed.  
 48006        When the *interval* argument is non-zero, repeated timeout notification occurs with a period in  
 48007        microseconds specified by the *interval* argument. If the notification signal, SIGALRM, is not  
 48008        caught or ignored, the calling process is terminated.

48009        Implementations may place limitations on the granularity of timer values. For each interval  
 48010        timer, if the requested timer value requires a finer granularity than the implementation supports,  
 48011        the actual timer value shall be rounded up to the next supported value.

48012        Interactions between *ualarm()* and any of the following are unspecified:

48013            *alarm()*48014            *nanosleep()*48015            *setitimer()*48016            *timer\_create()*48017            *timer\_delete()*48018            *timer\_getoverrun()*48019            *timer\_gettime()*48020            *timer\_settime()*48021            *sleep()*48022 **RETURN VALUE**

48023        The *ualarm()* function shall return the number of microseconds remaining from the previous  
 48024        *ualarm()* call. If no timeouts are pending or if *ualarm()* has not previously been called, *ualarm()*  
 48025        shall return 0.

48026 **ERRORS**

48027        No errors are defined.

48028 **EXAMPLES**

48029        None.

48030 **APPLICATION USAGE**

48031        Applications are recommended to use *nanosleep()* if the Timers option is supported, or  
 48032        *setitimer()*, *timer\_create()*, *timer\_delete()*, *timer\_getoverrun()*, *timer\_gettime()*, or *timer\_settime()*  
 48033        instead of this function.

48034 **RATIONALE**

48035        None.

48036 **FUTURE DIRECTIONS**

48037        None.

48038 **SEE ALSO**

48039        *alarm()*, *nanosleep()*, *setitimer()*, *sleep()*, *timer\_create()*, *timer\_delete()*, *timer\_getoverrun()*, the Base  
 48040        Definitions volume of IEEE Std. 1003.1-200x, <unistd.h>

48041 **CHANGE HISTORY**

48042 First released in Issue 4, Version 2.

48043 **Issue 5**

48044 Moved from X/OPEN UNIX extension to BASE.



48045 **NAME**48046 `ulimit` — get and set process limits48047 **SYNOPSIS**48048 XSI `#include <ulimit.h>`48049 `long ulimit(int cmd, ...);`

48050

48051 **DESCRIPTION**

48052 The `ulimit()` function provides for control over process limits. The process limits that can be  
 48053 controlled by this function include the maximum size of a single file that can be written (this is  
 48054 equivalent to using `setrlimit()` with `RLIMIT_FSIZE`). The `cmd` values, defined in `<ulimit.h>`  
 48055 include:

48056 `UL_GETFSIZE` Return the file size limit (`RLIMIT_FSIZE`) of the process. The limit is in units  
 48057 of 512-byte blocks and is inherited by child processes. Files of any size can be  
 48058 read. The return value shall be the integer part of the soft file size limit  
 48059 divided by 512. If the result cannot be represented as a **long**, the result is  
 48060 unspecified.

48061 `UL_SETFSIZE` Set the file size limit for output operations of the process to the value of the  
 48062 second argument, taken as a **long**, multiplied by 512. If the result would  
 48063 overflow an `rlim_t`, the actual value set is unspecified. Any process may  
 48064 decrease its own limit, but only a process with appropriate privileges may  
 48065 increase the limit. The return value shall be the integer part of the new file size  
 48066 limit divided by 512.

48067 The `ulimit()` function shall not change the setting of `errno` if successful.

48068 As all return values are permissible in a successful situation, an application wishing to check for  
 48069 error situations should set `errno` to 0, then call `ulimit()`, and, if it returns `-1`, check to see if `errno` is  
 48070 non-zero.

48071 **RETURN VALUE**

48072 Upon successful completion, `ulimit()` shall return the value of the requested limit. Otherwise, `-1`  
 48073 shall be returned and `errno` set to indicate the error.

48074 **ERRORS**

48075 The `ulimit()` function shall fail and the limit shall be unchanged if:

48076 `[EINVAL]` The `cmd` argument is not valid.

48077 `[EPERM]` A process not having appropriate privileges attempts to increase its file size  
 48078 limit.

48079 **EXAMPLES**

48080 None.

48081 **APPLICATION USAGE**

48082 None.

48083 **RATIONALE**

48084 None.

48085 **FUTURE DIRECTIONS**

48086 None.

48087 **SEE ALSO**

48088 *getrlimit()*, *setrlimit()*, *write()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**ulimit.h**>

48089 **CHANGE HISTORY**

48090 First released in Issue 1. Derived from Issue 1 of the SVID.

48091 **Issue 4**

48092 The use of **long** is replaced by **long** in the SYNOPSIS and the DESCRIPTION sections.

48093 **Issue 4, Version 2**

48094 In the DESCRIPTION, the discussion of UL\_GETFSIZE and UL\_SETFSIZE is revised generally to distinguish between the soft and the hard file size limit of the process. For UL\_GETFSIZE, the return value is defined more precisely. For UL\_SETFSIZE, the effect on both file size limits is specified, as is the effect if the result would overflow an **rlim\_t**.

48098 **Issue 5**

48099 In the description of UL\_SETFSIZE, the text is corrected to refer to **rlim\_t** rather than the spurious **rlimit\_t**.

48101 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

48102 **NAME**

48103       umask — set and get file mode creation mask

48104 **SYNOPSIS**

48105       #include &lt;sys/stat.h&gt;

48106       mode\_t umask(mode\_t *cmask*);48107 **DESCRIPTION**

48108       The *umask()* function shall set the process' file mode creation mask to *cmask* and return the  
48109       previous value of the mask. Only the file permission bits of *cmask* (see <sys/stat.h>) are used; the  
48110       meaning of the other bits is implementation-defined.

48111       The process' file mode creation mask is used during *open()*, *creat()*, *mkdir()*, and *mkfifo()* to turn  
48112       off permission bits in the *mode* argument supplied. Bit positions that are set in *cmask* are cleared  
48113       in the mode of the created file.

48114 **RETURN VALUE**

48115       The file permission bits in the value returned by *umask()* shall be the previous value of the file  
48116       mode creation mask. The state of any other bits in that value is unspecified, except that a  
48117       subsequent call to *umask()* with the returned value as *cmask* shall leave the state of the mask the  
48118       same as its state before the first call, including any unspecified use of those bits.

48119 **ERRORS**

48120       No errors are defined.

48121 **EXAMPLES**

48122       None.

48123 **APPLICATION USAGE**

48124       None.

48125 **RATIONALE**

48126       Unsigned argument and return types for *umask()* were proposed. The return type and the  
48127       argument were both changed to **mode\_t**.

48128       Historical implementations have made use of additional bits in *cmask* for their implementation-  
48129       defined purposes. The addition of the text that the meaning of other bits of the field is  
48130       implementation-defined permits these implementations to conform to this volume of  
48131       IEEE Std. 1003.1-200x.

48132 **FUTURE DIRECTIONS**

48133       None.

48134 **SEE ALSO**

48135       *creat()*, *mkdir()*, *mkfifo()*, *open()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
48136       <sys/stat.h>, <sys/types.h>

48137 **CHANGE HISTORY**

48138       First released in Issue 1. Derived from Issue 1 of the SVID.

48139 **Issue 4**

48140       The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
48141       XSI-conformant systems.

48142       The RETURN VALUE section is expanded, in line with the ISO POSIX-1 standard, to describe  
48143       the situation with regard to additional bits in the file mode creation mask.

48144 **Issue 6**

48145 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

48146 The following new requirements on POSIX implementations derive from alignment with the  
48147 Single UNIX Specification:

- 48148 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
48149 required for conforming implementations of previous POSIX specifications, it was not  
48150 required for UNIX applications.

48151 **NAME**

48152            uname — get name of current system

48153 **SYNOPSIS**

48154            #include &lt;sys/utsname.h&gt;

48155            int uname(struct utsname \*name);

48156 **DESCRIPTION**48157            The *uname()* function shall store information identifying the current system in the structure  
48158            pointed to by *name*.48159            The *uname()* function uses the **utsname** structure defined in <sys/utsname.h>.48160            The *uname()* function returns a string naming the current system in the character array *sysname*.  
48161            Similarly, *nodename* contains the name that the system is known by on a communications  
48162            network. The arrays *release* and *version* further identify the operating system. The array *machine*  
48163            contains a name that identifies the hardware that the system is running on.

48164            The format of each member is implementation-defined.

48165 **RETURN VALUE**48166            Upon successful completion, a non-negative value shall be returned. Otherwise, -1 shall be  
48167            returned and *errno* set to indicate the error.48168 **ERRORS**

48169            No errors are defined.

48170 **EXAMPLES**

48171            None.

48172 **APPLICATION USAGE**48173            The inclusion of the *nodename* member in this structure does not imply that it is sufficient  
48174            information for interfacing to communications networks.48175 **RATIONALE**48176            The values of the structure members are not constrained to have any relation to the version of  
48177            this volume of IEEE Std. 1003.1-200x implemented in the operating system. An application  
48178            should instead depend on `_POSIX_VERSION` and related constants defined in <unistd.h>.48179            This volume of IEEE Std. 1003.1-200x does not define the sizes of the members of the structure  
48180            and permits them to be of different sizes, although most implementations define them all to be  
48181            the same size: eight bytes plus one byte for the string terminator. That size for *nodename* is not  
48182            enough for use with many networks.48183            The *uname()* function is specific to System III, System V, and related implementations, and it  
48184            does not exist in Version 7 or 4.3 BSD. The values it returns are set at system compile time in  
48185            those historical implementations.48186            4.3 BSD has *gethostname()* and *gethostid()*, which return a symbolic name and a numeric value,  
48187            respectively. There are related *sethostname()* and *sethostid()* functions that are used to set the  
48188            values the other two functions return. The length of the host name is limited to 31 characters in  
48189            most implementations and the host ID is a 32-bit integer.48190 **FUTURE DIRECTIONS**

48191            None.

48192 **SEE ALSO**

48193           The Base Definitions volume of IEEE Std. 1003.1-200x, &lt;sys/utsname.h&gt;

48194 **CHANGE HISTORY**

48195           First released in Issue 1. Derived from Issue 1 of the SVID.

48196 **Issue 4**

48197           The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 48198           • The DESCRIPTION is changed to indicate that the format of members in the **utsname**
- 48199            structure is implementation-defined.
  
- 48200           • The RETURN VALUE section is updated to indicate that `-1` is returned and *errno* set to
- 48201            indicate an error.

48202 **NAME**

48203 ungetc — push byte back into input stream

48204 **SYNOPSIS**

48205 #include &lt;stdio.h&gt;

48206 int ungetc(int *c*, FILE \**stream*);48207 **DESCRIPTION**

48208 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
48209 conflict between the requirements described here and the ISO C standard is unintentional. This  
48210 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

48211 The *ungetc()* function shall push the byte specified by *c* (converted to an **unsigned char**) back  
48212 onto the input stream pointed to by *stream*. The pushed-back bytes shall be returned by  
48213 subsequent reads on that stream in the reverse order of their pushing. A successful intervening  
48214 call (with the stream pointed to by *stream*) to a file-positioning function (*fseek()*, *fsetpos()*, or  
48215 *rewind()*) discards any pushed-back bytes for the stream. The external storage corresponding to  
48216 the stream is unchanged.

48217 One byte of push-back is guaranteed. If *ungetc()* is called too many times on the same stream  
48218 without an intervening read or file-positioning operation on that stream, the operation may fail.

48219 If the value of *c* equals that of the macro EOF, the operation fails and the input stream is  
48220 unchanged.

48221 A successful call to *ungetc()* shall clear the end-of-file indicator for the stream. The value of the  
48222 file-position indicator for the stream after reading or discarding all pushed-back bytes shall be  
48223 the same as it was before the bytes were pushed back. The file-position indicator is decremented  
48224 by each successful call to *ungetc()*; if its value was 0 before a call, its value is indeterminate after  
48225 the call.

48226 **RETURN VALUE**

48227 Upon successful completion, *ungetc()* shall return the byte pushed back after conversion.  
48228 Otherwise, it shall return EOF.

48229 **ERRORS**

48230 No errors are defined.

48231 **EXAMPLES**

48232 None.

48233 **APPLICATION USAGE**

48234 None.

48235 **RATIONALE**

48236 None.

48237 **FUTURE DIRECTIONS**

48238 None.

48239 **SEE ALSO**

48240 *fseek()*, *getc()*, *fsetpos()*, *read()*, *rewind()*, *setbuf()*, the Base Definitions volume of  
48241 IEEE Std. 1003.1-200x, <stdio.h>

48242 **CHANGE HISTORY**

48243 First released in Issue 1. Derived from Issue 1 of the SVID.

48244 **Issue 4**

48245 The DESCRIPTION is changed to make it clear that *ungetc()* manipulates bytes rather than  
48246 (possibly multi-byte) characters.

48247 The APPLICATION USAGE section is removed.

48248 The following changes are incorporated for alignment with the ISO C standard:

- 48249 • The *fsetpos()* function is added to the list of file-positioning functions in the DESCRIPTION.
- 48250 • Also, this issue states that the file-position indicator is decremented by each successful call to  
48251 *ungetc()*, although note that XSI-conformant systems do not distinguish between text and  
48252 binary streams. Previous issues state that the disposition of this indicator is unspecified.



48253 **NAME**

48254 ungetwc — push wide-character code back into input stream

48255 **SYNOPSIS**

48256 #include &lt;stdio.h&gt;

48257 #include &lt;wchar.h&gt;

48258 wint\_t ungetwc(wint\_t *wc*, FILE \**stream*);48259 **DESCRIPTION**

48260 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 48261 conflict between the requirements described here and the ISO C standard is unintentional. This  
 48262 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

48263 The *ungetwc()* function shall push the character corresponding to the wide-character code  
 48264 specified by *wc* back onto the input stream pointed to by *stream*. The pushed-back characters  
 48265 shall be returned by subsequent reads on that stream in the reverse order of their pushing. A  
 48266 successful intervening call (with the stream pointed to by *stream*) to a file-positioning function  
 48267 (*fseek()*, *fsetpos()*, or *rewind()*) discards any pushed-back characters for the stream. The external  
 48268 storage corresponding to the stream is unchanged.

48269 One character of push-back is guaranteed. If *ungetwc()* is called too many times on the same  
 48270 stream without an intervening read or file-positioning operation on that stream, the operation  
 48271 may fail.

48272 If the value of *wc* equals that of the macro WEOF, the operation fails and the input stream is  
 48273 unchanged.

48274 A successful call to *ungetwc()* shall clear the end-of-file indicator for the stream. The value of the  
 48275 file-position indicator for the stream after reading or discarding all pushed-back characters shall  
 48276 be the same as it was before the characters were pushed back. The file-position indicator is  
 48277 decremented (by one or more) by each successful call to *ungetwc()*; if its value was 0 before a  
 48278 call, its value is indeterminate after the call.

48279 **RETURN VALUE**

48280 Upon successful completion, *ungetwc()* shall return the wide-character code corresponding to  
 48281 the pushed-back character. Otherwise, it shall return WEOF.

48282 **ERRORS**48283 The *ungetwc()* function may fail if:

|       |          |                                                                              |  |
|-------|----------|------------------------------------------------------------------------------|--|
| 48284 | [EILSEQ] | An invalid character sequence is detected, or a wide-character code does not |  |
| 48285 |          | correspond to a valid character.                                             |  |

48286 **EXAMPLES**

48287 None.

48288 **APPLICATION USAGE**

48289 None.

48290 **RATIONALE**

48291 None.

48292 **FUTURE DIRECTIONS**

48293 None.

48294 **SEE ALSO**

48295 *fseek()*, *fsetpos()*, *read()*, *rewind()*, *setbuf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |  
48296 `<stdio.h>`, `<wchar.h>`

48297 **CHANGE HISTORY**

48298 First released in Issue 4. Derived from the MSE working draft. |

48299 **Issue 5**

48300 The Optional Header (OH) marking is removed from `<stdio.h>`.

## 48301 NAME

48302 unlink — remove a directory entry

## 48303 SYNOPSIS

48304 #include &lt;unistd.h&gt;

48305 int unlink(const char \*path);

## 48306 DESCRIPTION

48307 The *unlink()* function shall remove a link to a file. If *path* names a symbolic link, *unlink()* |  
 48308 removes the symbolic link named by *path* and does not affect any file or directory named by the |  
 48309 contents of the symbolic link. Otherwise, *unlink()* removes the link named by the path name |  
 48310 pointed to by *path* and decrements the link count of the file referenced by the link.

48311 When the file's link count becomes 0 and no process has the file open, the space occupied by the |  
 48312 file shall be freed and the file shall no longer be accessible. If one or more processes have the file |  
 48313 open when the last link is removed, the link shall be removed before *unlink()* returns, but the |  
 48314 removal of the file contents shall be postponed until all references to the file are closed.

48315 The application shall ensure that the *path* argument does not name a directory unless the process |  
 48316 has appropriate privileges and the implementation supports using *unlink()* on directories.

48317 Upon successful completion, *unlink()* shall mark for update the *st\_ctime* and *st\_mtime* fields of |  
 48318 the parent directory. Also, if the file's link count is not 0, the *st\_ctime* field of the file shall be |  
 48319 marked for update.

## 48320 RETURN VALUE

48321 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to |  
 48322 indicate the error. If -1 is returned, the named file shall not be changed.

## 48323 ERRORS

48324 The *unlink()* function shall fail and shall not unlink the file if:

48325 [EACCES] Search permission is denied for a component of the path prefix, or write |  
 48326 permission is denied on the directory containing the directory entry to be |  
 48327 removed.

48328 [EBUSY] The file named by the *path* argument cannot be unlinked because it is being |  
 48329 used by the system or another process and the implementation considers this |  
 48330 an error.

48331 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* |  
 48332 argument.

48333 [ENAMETOOLONG] |  
 48334 The length of the *path* argument exceeds {PATH\_MAX} or a path name |  
 48335 component is longer than {NAME\_MAX}.

48336 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string. |

48337 [ENOTDIR] A component of the path prefix is not a directory. |

48338 [EPERM] The file named by *path* is a directory, and either the calling process does not |  
 48339 have appropriate privileges, or the implementation prohibits using *unlink()* |  
 48340 on directories.

48341 XSI [EPERM] or [EACCES]

48342 The S\_ISVTX flag is set on the directory containing the file referred to by the |  
 48343 *path* argument and the caller is not the file owner, nor is the caller the |  
 48344 directory owner, nor does the caller have appropriate privileges.

48345 [EROFS] The directory entry to be unlinked is part of a read-only file system.

48346 The *unlink()* function may fail and not unlink the file if:

48347 XSI [EBUSY] The file named by *path* is a named STREAM.

48348 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
48349 resolution of the *path* argument.

48350 [ENAMETOOLONG]  
48351 As a result of encountering a symbolic link in resolution of the *path* argument,  
48352 the length of the substituted path name string exceeded {PATH\_MAX}.

48353 [ETXTBSY] The entry to be unlinked is the last directory entry to a pure procedure (shared  
48354 text) file that is being executed.

## 48355 EXAMPLES

### 48356 Removing a Link to a File

48357 The following example shows how to remove a link to a file named `/home/cnd/mod1` by  
48358 removing the entry named `/modules/pass1`.

```
48359 #include <unistd.h>
48360 char *path = "/modules/pass1";
48361 int status;
48362 ...
48363 status = unlink(path);
```

### 48364 Checking for an Error

48365 The following example fragment creates a temporary password lock file named **LOCKFILE**,  
48366 which is defined as `/etc/ptmp`, and gets a file descriptor for it. If the file cannot be opened for  
48367 writing, *unlink()* is used to remove the link between the file descriptor and **LOCKFILE**.

```
48368 #include <sys/types.h>
48369 #include <stdio.h>
48370 #include <fcntl.h>
48371 #include <errno.h>
48372 #include <unistd.h>
48373 #include <sys/stat.h>
48374 #define LOCKFILE "/etc/ptmp"
48375 int pfd; /* Integer for file descriptor returned by open call. */
48376 FILE *fpfd; /* File pointer for use in putpwent(). */
48377 ...
48378 /* Open password Lock file. If it exists, this is an error. */
48379 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL, S_IRUSR
48380 | S_IWUSR | S_IRGRP | S_IROTH)) == -1) {
48381 fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
48382 exit(1);
48383 }
48384 /* Lock file created, proceed with fdopen of lock file so that
48385 putpwent() can be used.
48386 */
48387 if ((fpfd = fdopen(pfd, "w")) == NULL) {
```

```

48388 close(pfd);
48389 unlink(LOCKFILE);
48390 exit(1);
48391 }

```

### 48392 **Replacing Files**

48393 The following example fragment uses *unlink()* to discard links to files, so that they can be  
48394 replaced with new versions of the files. The first call remove the link to **LOCKFILE** if an error  
48395 occurs. Successive calls remove the links to **SAVEFILE** and **PASSWDFILE** so that new links can  
48396 be created, then removes the link to **LOCKFILE** when it is no longer needed.

```

48397 #include <sys/types.h>
48398 #include <stdio.h>
48399 #include <fcntl.h>
48400 #include <errno.h>
48401 #include <unistd.h>
48402 #include <sys/stat.h>

48403 #define LOCKFILE "/etc/ptmp"
48404 #define PASSWDFILE "/etc/passwd"
48405 #define SAVEFILE "/etc/opasswd"
48406 ...
48407 /* If no change was made, assume error and leave passwd unchanged. */
48408 if (!valid_change) {
48409 fprintf(stderr, "Could not change password for user %s\n", user);
48410 unlink(LOCKFILE);
48411 exit(1);
48412 }

48413 /* Change permissions on new password file. */
48414 chmod(LOCKFILE, S_IRUSR | S_IRGRP | S_IROTH);

48415 /* Remove saved password file. */
48416 unlink(SAVEFILE);

48417 /* Save current password file. */
48418 link(PASSWDFILE, SAVEFILE);

48419 /* Remove current password file. */
48420 unlink(PASSWDFILE);

48421 /* Save new password file as current password file. */
48422 link(LOCKFILE, PASSWDFILE);

48423 /* Remove lock file. */
48424 unlink(LOCKFILE);

48425 exit(0);

```

### 48426 **APPLICATION USAGE**

48427 Applications should use *rmdir()* to remove a directory.

### 48428 **RATIONALE**

48429 Unlinking a directory is restricted to the superuser in many historical implementations for  
48430 reasons given in *link()* (see also *rename()*).

48431 The meaning of [EBUSY] in historical implementations is “mount point busy”. Since this volume  
 48432 of IEEE Std. 1003.1-200x does not cover the system administration concepts of mounting and  
 48433 unmounting, the description of the error was changed to “resource busy”. (This meaning is used  
 48434 by some device drivers when a second process tries to open an exclusive use device.) The  
 48435 wording is also intended to allow implementations to refuse to remove a directory if it is the  
 48436 root or current working directory of any process.

48437 **FUTURE DIRECTIONS**

48438 None.

48439 **SEE ALSO**

48440 *close()*, *link()*, *remove()*, *rmdir()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
 48441 <**unistd.h**>

48442 **CHANGE HISTORY**

48443 First released in Issue 1. Derived from Issue 1 of the SVID.

48444 **Issue 4**

48445 The <**unistd.h**> header is added to the SYNOPSIS section.

48446 The error [ETXTBSY] is marked as an extension.

48447 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 48448 • The type of argument *path* is changed from **char\*** to **const char\***.

48449 The following change is incorporated for alignment with the FIPS requirements:

- 48450 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path  
 48451 name component is larger than {NAME\_MAX} is now defined as mandatory and marked as  
 48452 an extension.

48453 **Issue 4, Version 2**

48454 The entry is updated for X/OPEN UNIX conformance as follows:

- 48455 • In the DESCRIPTION, the effect is specified if *path* specifies a symbolic link.
- 48456 • In the ERRORS section, [ELOOP] is added to indicate that too many symbolic links were  
 48457 encountered during path name resolution
- 48458 • In the ERRORS section, [EPERM] or [EACCES] are added to indicate a permission check  
 48459 failure when operating on directories with S\_ISVTX set.
- 48460 • In the ERRORS section, a second [ENAMETOOLONG] condition is defined that may report  
 48461 excessive length of an intermediate result of path name resolution of a symbolic link.

48462 **Issue 5**

48463 The [EBUSY] error is added to the “may fail” part of the ERRORS section.

48464 **Issue 6**

48465 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 48466 • The [ENAMETOOLONG] error is restored as an error dependent on \_POSIX\_NO\_TRUNC.  
 48467 This is since behavior may vary from one file system to another.

48468 The following new requirements on POSIX implementations derive from alignment with the  
 48469 Single UNIX Specification:

- 48470 • In the DESCRIPTION, the effect is specified if *path* specifies a symbolic link.
- 48471 • The [ELOOP] mandatory error condition is added.

48472           • A second [ENAMETOOLONG] is added as an optional error condition.

48473           • The [ETXTBSY] optional error condition is added.

48474           The following changes were made to align with the IEEE P1003.1a draft standard:

48475           • The [ELOOP] optional error condition is added.

48476           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48477 **NAME**

48478 unlockpt — unlock a pseudo-terminal master/slave pair

48479 **SYNOPSIS**48480 XSI `#include <stdlib.h>`48481 `int unlockpt(int fildev);`

48482

48483 **DESCRIPTION**48484 The `unlockpt()` function shall unlock the slave pseudo-terminal device associated with the  
48485 master to which *fildev* refers.48486 Portable applications shall ensure that they call `unlockpt()` before opening the slave side of a  
48487 pseudo-terminal device.48488 **RETURN VALUE**48489 Upon successful completion, `unlockpt()` shall return 0. Otherwise, it shall return `-1` and set *errno*  
48490 to indicate the error.48491 **ERRORS**48492 The `unlockpt()` function may fail if:48493 [EBADF] The *fildev* argument is not a file descriptor open for writing. |48494 [EINVAL] The *fildev* argument is not associated with a master pseudo-terminal device. |48495 **EXAMPLES**

48496 None.

48497 **APPLICATION USAGE**

48498 None.

48499 **RATIONALE**

48500 None.

48501 **FUTURE DIRECTIONS**

48502 None.

48503 **SEE ALSO**48504 `grantpt()`, `open()`, `ptsname()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<stdlib.h>` |48505 **CHANGE HISTORY**

48506 First released in Issue 4, Version 2.

48507 **Issue 5**

48508 Moved from X/OPEN UNIX extension to BASE.

48509 **Issue 6**

48510 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.



48511 **NAME**

48512           unsetenv — remove environment variable

48513 **SYNOPSIS**

48514           #include &lt;stdlib.h&gt;

48515           int unsetenv(const char \*name);

48516 **DESCRIPTION**

48517           The *unsetenv()* function removes an environment variable from the environment of the calling  
48518           process. The *name* argument points to a string, which is the name of the variable to be removed.  
48519           The named argument shall not contain an '=' character. If the named variable does not exist in  
48520           the current environment, the environment is unchanged and the function is considered to have  
48521           completed successfully.

48522           If the application modifies *environ* or the pointers to which it points, the behavior of *unsetenv()* is  
48523           undefined. The *unsetenv()* function shall update the list of pointers to which *environ* points.

48524           The *unsetenv()* function need not be reentrant. A function that is not required to be reentrant is  
48525           not required to be thread-safe.

48526 **RETURN VALUE**

48527           Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to  
48528           indicate the error, and the environment shall be unchanged.

48529 **ERRORS**48530           The *unsetenv()* function shall fail if:

48531           [EINVAL]           The *name* argument is a null pointer, points to an empty string, or points to a  
48532           string containing an '=' character.

48533 **EXAMPLES**

48534           None.

48535 **APPLICATION USAGE**

48536           None.

48537 **RATIONALE**48538           Refer to the RATIONALE section in *setenv()*.48539 **FUTURE DIRECTIONS**

48540           None.

48541 **SEE ALSO**

48542           *getenv()*, *setenv()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdlib.h>, |  
48543           <sys/types.h>, <unistd.h>

48544 **CHANGE HISTORY**

48545           First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

## 48546 NAME

48547            usleep — suspend execution for an interval

## 48548 SYNOPSIS

48549 XSI        #include &lt;unistd.h&gt;

48550            int usleep(useconds\_t useconds);

48551

## 48552 DESCRIPTION

48553        The *usleep()* function shall cause the calling thread to be suspended from execution until either  
 48554        the number of realtime microseconds specified by the argument *useconds* has elapsed or a signal  
 48555        is delivered to the calling thread and its action is to invoke a signal-catching function or to  
 48556        terminate the process. The suspension time may be longer than requested due to the scheduling  
 48557        of other activity by the system.

48558        The application shall ensure that the *useconds* argument is less than 1,000,000. If the value of  
 48559        *useconds* is 0, then the call has no effect.

48560        If a SIGALRM signal is generated for the calling process during execution of *usleep()* and if the  
 48561        SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *usleep()*  
 48562        returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also  
 48563        unspecified whether it remains pending after *usleep()* returns or it is discarded.

48564        If a SIGALRM signal is generated for the calling process during execution of *usleep()*, except as a  
 48565        result of a prior call to *alarm()*, and if the SIGALRM signal is not being ignored or blocked from  
 48566        delivery, it is unspecified whether that signal has any effect other than causing *usleep()* to return.

48567        If a signal-catching function interrupts *usleep()* and examines or changes either the time a  
 48568        SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or  
 48569        whether the SIGALRM signal is blocked from delivery, the results are unspecified.

48570        If a signal-catching function interrupts *usleep()* and calls *siglongjmp()* or *longjmp()* to restore an  
 48571        environment saved prior to the *usleep()* call, the action associated with the SIGALRM signal and  
 48572        the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also  
 48573        unspecified whether the SIGALRM signal is blocked, unless the process' signal mask is restored  
 48574        as part of the environment.

48575        Implementations may place limitations on the granularity of timer values. For each interval  
 48576        timer, if the requested timer value requires a finer granularity than the implementation supports,  
 48577        the actual timer value shall be rounded up to the next supported value.

48578        Interactions between *usleep()* and any of the following are unspecified:

48579            *nanosleep()*48580            *setitimer()*48581            *timer\_create()*48582            *timer\_delete()*48583            *timer\_getoverrun()*48584            *timer\_gettime()*48585            *timer\_settime()*48586            *ualarm()*48587            *sleep()*

**48588 RETURN VALUE**

48589       Upon successful completion, *usleep()* shall return 0; otherwise, it shall return -1 and set *errno* to  
48590       indicate the error.

**48591 ERRORS**

48592       The *usleep()* function may fail if:

48593       [EINVAL]       The time interval specified 1,000,000 or more microseconds. |

**48594 EXAMPLES**

48595       None.

**48596 APPLICATION USAGE**

48597       Applications are recommended to use *nanosleep()* if the Timers option is supported, or |  
48598       *setitimer()*, *timer\_create()*, *timer\_delete()*, *timer\_getoverrun()*, *timer\_gettime()*, or *timer\_settime()*  
48599       instead of this function.

**48600 RATIONALE**

48601       None.

**48602 FUTURE DIRECTIONS**

48603       None.

**48604 SEE ALSO**

48605       *alarm()*, *getitimer()*, *nanosleep()*, *sigaction()*, *sleep()*, *timer\_create()*, *timer\_delete()*,  
48606       *timer\_getoverrun()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**> |

**48607 CHANGE HISTORY**

48608       First released in Issue 4, Version 2.

**48609 Issue 5**

48610       Moved from X/OPEN UNIX extension to BASE.

48611       The DESCRIPTION is changed to indicate that timers are now thread-based rather than  
48612       process-based.

**48613 Issue 6**

48614       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48615 **NAME**

48616 utime — set file access and modification times

48617 **SYNOPSIS**

48618 #include &lt;utime.h&gt;

48619 int utime(const char \*path, const struct utimbuf \*times);

48620 **DESCRIPTION**48621 The *utime()* function shall set the access and modification times of the file named by the *path*  
48622 argument.48623 If *times* is a null pointer, the access and modification times of the file are set to the current time.  
48624 The application shall ensure that the effective user ID of the process matches the owner of the  
48625 file, or the process has write permission to the file or has appropriate privileges, to use *utime()* in  
48626 this manner.48627 If *times* is not a null pointer, *times* is interpreted as a pointer to a **utimbuf** structure and the  
48628 access and modification times are set to the values contained in the designated structure. Only a  
48629 process with effective user ID equal to the user ID of the file or a process with appropriate  
48630 privileges may use *utime()* this way.48631 The **utimbuf** structure is defined by the header <utime.h>. The times in the structure **utimbuf**  
48632 are measured in seconds since the Epoch.48633 Upon successful completion, *utime()* shall mark the time of the last file status change, *st\_ctime*,  
48634 to be updated; see <sys/stat.h>.48635 **RETURN VALUE**48636 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* shall  
48637 be set to indicate the error, and the file times shall not be affected.48638 **ERRORS**48639 The *utime()* function shall fail if:48640 [EACCES] Search permission is denied by a component of the path prefix; or the *times* |  
48641 argument is a null pointer and the effective user ID of the process does not |  
48642 match the owner of the file, the process does not have write permission for the |  
48643 file, and the process does not have appropriate privileges. |48644 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* |  
48645 argument. |48646 [ENAMETOOLONG] |  
48647 The length of the *path* argument exceeds {PATH\_MAX} or a path name |  
48648 component is longer than {NAME\_MAX}. |48649 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string. |

48650 [ENOTDIR] A component of the path prefix is not a directory. |

48651 [EPERM] The *times* argument is not a null pointer and the calling process' effective user |  
48652 ID does not match the owner of the file and the calling process does not have |  
48653 the appropriate privileges. |

48654 [EROFS] The file system containing the file is read-only. |

48655 The *utime()* function may fail if:48656 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during |  
48657 resolution of the *path* argument. |

48658 [ENAMETOOLONG]  
 48659 As a result of encountering a symbolic link in resolution of the *path* argument,  
 48660 the length of the substituted path name string exceeded {PATH\_MAX}.

#### 48661 EXAMPLES

48662 None.

#### 48663 APPLICATION USAGE

48664 None.

#### 48665 RATIONALE

48666 The *actime* structure member must be present so that an application may set it, even though an  
 48667 implementation may ignore it and not change the access time on the file. If an application  
 48668 intends to leave one of the times of a file unchanged while changing the other, it should use  
 48669 *stat()* to retrieve the file's *st\_atime* and *st\_mtime* parameters, set *actime* and *modtime* in the buffer,  
 48670 and change one of them before making the *utime()* call.

#### 48671 FUTURE DIRECTIONS

48672 None.

#### 48673 SEE ALSO

48674 The Base Definitions volume of IEEE Std. 1003.1-200x, <sys/types.h>, <utime.h>

#### 48675 CHANGE HISTORY

48676 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 48677 Issue 4

48678 The <sys/types.h> header is now marked as optional (OH); this header need not be included on  
 48679 XSI-conformant systems.

48680 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 48681 • The type of argument *path* is changed from **char\*** to **const char\***, and *times* is changed from  
 48682 **struct utimbuf\*** to **const struct utimbuf\***.

48683 The following change is incorporated for alignment with the FIPS requirements:

- 48684 • In the ERRORS section, the condition whereby [ENAMETOOLONG] is returned if a path  
 48685 name component is larger than {NAME\_MAX} is now defined as mandatory and marked as  
 48686 an extension.

#### 48687 Issue 4, Version 2

48688 The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- 48689 • It states that [ELOOP] is returned if too many symbolic links are encountered during path  
 48690 name resolution.
- 48691 • A second [ENAMETOOLONG] condition is defined that may report excessive length of an  
 48692 intermediate result of path name resolution of a symbolic link.

#### 48693 Issue 6

48694 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

48695 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 48696 • The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.  
 48697 This is since behavior may vary from one file system to another.

48698 The following new requirements on POSIX implementations derive from alignment with the  
 48699 Single UNIX Specification:

- 48700       • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
- 48701       required for conforming implementations of previous POSIX specifications, it was not
- 48702       required for UNIX applications.
- 48703       • The [ELOOP] mandatory error condition is added.
- 48704       • A second [ENAMETOOLONG] is added as an optional error condition.
- 48705       The following changes were made to align with the IEEE P1003.1a draft standard:
- 48706       • The [ELOOP] optional error condition is added.
- 48707       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48708 **NAME**48709 utimes — set file access and modification times (**LEGACY**)48710 **SYNOPSIS**48711 XSI `#include <sys/time.h>`48712 `int utimes(const char *path, const struct timeval times[2]);`

48713

48714 **DESCRIPTION**

48715 The *utimes()* function shall set the access and modification times of the file pointed to by the *path*  
 48716 argument to the value of the *times* argument. The *utimes()* function allows time specifications  
 48717 accurate to the microsecond.

48718 For *utimes()*, the *times* argument is an array of **timeval** structures. The first array member  
 48719 represents the date and time of last access, and the second member represents the date and time  
 48720 of last modification. The times in the **timeval** structure are measured in seconds and  
 48721 microseconds since the Epoch, although rounding toward the nearest second may occur.

48722 If the *times* argument is a null pointer, the access and modification times of the file shall be set to  
 48723 the current time. The application shall ensure that the effective user ID of the process is the same  
 48724 as the owner of the file, or has write access to the file or appropriate privileges to use this call in  
 48725 this manner. Upon completion, *utimes()* shall mark the time of the last file status change,  
 48726 *st\_ctime*, for update.

48727 **RETURN VALUE**

48728 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* shall  
 48729 be set to indicate the error, and the file times shall not be affected.

48730 **ERRORS**48731 The *utimes()* function shall fail if:

48732 [EACCES] Search permission is denied by a component of the path prefix; or the *times*  
 48733 argument is a null pointer and the effective user ID of the process does not  
 48734 match the owner of the file and write access is denied.

48735 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 48736 argument.

48737 [ENAMETOOLONG]  
 48738 The length of the *path* argument exceeds {PATH\_MAX} or a path name  
 48739 component is longer than {NAME\_MAX}.

48740 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

48741 [ENOTDIR] A component of the path prefix is not a directory.

48742 [EPERM] The *times* argument is not a null pointer and the calling process' effective user  
 48743 ID has write access to the file but does not match the owner of the file and the  
 48744 calling process does not have the appropriate privileges.

48745 [EROFS] The file system containing the file is read-only.

48746 The *utimes()* function may fail if:

48747 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 48748 resolution of the *path* argument.

48749 [ENAMETOOLONG]  
 48750 Path name resolution of a symbolic link produced an intermediate result  
 48751 whose length exceeds {PATH\_MAX}.

48752 **EXAMPLES**

48753 None.

48754 **APPLICATION USAGE**

48755 For applications portability, the *utime()* function should be used to set file access and  
48756 modification times instead of *utimes()*.

48757 **RATIONALE**

48758 None.

48759 **FUTURE DIRECTIONS**

48760 This function may be withdrawn in a future version.

48761 **SEE ALSO**

48762 The Base Definitions volume of IEEE Std. 1003.1-200x, &lt;sys/time.h&gt;

48763 **CHANGE HISTORY**

48764 First released in Issue 4, Version 2.

48765 **Issue 5**

48766 Moved from X/OPEN UNIX extension to BASE.

48767 **Issue 6**

48768 This function is marked LEGACY.

48769 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 48770 • The [ENAMETOOLONG] error is restored as an error dependent on `_POSIX_NO_TRUNC`.  
48771 This is since behavior may vary from one file system to another.

48772 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48773 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
48774 [ELOOP] error condition is added.



48775 **NAME**

48776 va\_arg, va\_copy, va\_end, va\_start — handle variable argument list |

48777 **SYNOPSIS**

48778 #include &lt;stdarg.h&gt;

48779 type va\_arg(va\_list ap, type);

48780 void va\_copy(va\_list dest, va\_list src); |

48781 void va\_end(va\_list ap); |

48782 void va\_start(va\_list ap, argN);

48783 **DESCRIPTION**

48784 Refer to the Base Definitions volume of IEEE Std. 1003.1-200x, &lt;stdarg.h&gt;. |

## 48785 NAME

48786 vfork — create new process; share virtual memory

## 48787 SYNOPSIS

48788 XSI #include &lt;unistd.h&gt;

48789 pid\_t vfork(void);

48790

## 48791 DESCRIPTION

48792 The *vfork()* function has the same effect as *fork()*, except that the behavior is undefined if the  
 48793 process created by *vfork()* either modifies any data other than a variable of type **pid\_t** used to  
 48794 store the return value from *vfork()*, or returns from the function in which *vfork()* was called, or  
 48795 calls any other function before successfully calling *\_exit()* or one of the *exec* family of functions.

## 48796 RETURN VALUE

48797 Upon successful completion, *vfork()* shall return 0 to the child process and return the process ID  
 48798 of the child process to the parent process. Otherwise, -1 shall be returned to the parent, no child  
 48799 process shall be created, and *errno* shall be set to indicate the error.

## 48800 ERRORS

48801 The *vfork()* function shall fail if:

48802 [EAGAIN] The system-wide limit on the total number of processes under execution  
 48803 would be exceeded, or the system-imposed limit on the total number of  
 48804 processes under execution by a single user would be exceeded.

48805 [ENOMEM] There is insufficient swap space for the new process.

## 48806 EXAMPLES

48807 None.

## 48808 APPLICATION USAGE

48809 On some systems, *vfork()* is the same as *fork()*.

48810 The *vfork()* function differs from *fork()* only in that the child process can share code and data  
 48811 with the calling process (parent process). This speeds cloning activity significantly at a risk to  
 48812 the integrity of the parent process if *vfork()* is misused.

48813 The use of *vfork()* for any purpose except as a prelude to an immediate call to a function from  
 48814 the *exec* family, or to *\_exit()*, is not advised.

48815 The *vfork()* function can be used to create new processes without fully copying the address  
 48816 space of the old process. If a forked process is simply going to call *exec*, the data space copied  
 48817 from the parent to the child by *fork()* is not used. This is particularly inefficient in a paged  
 48818 environment, making *vfork()* particularly useful. Depending upon the size of the parent's data  
 48819 space, *vfork()* can give a significant performance improvement over *fork()*.

48820 The *vfork()* function can normally be used just like *fork()*. It does not work, however, to return  
 48821 while running in the child's context from the caller of *vfork()* since the eventual return from  
 48822 *vfork()* would then return to a no longer existent stack frame. Care should be taken, also, to call  
 48823 *\_exit()* rather than *exit()* if *exec* cannot be used, since *exit()* flushes and closes standard I/O  
 48824 channels, thereby damaging the parent process' standard I/O data structures. (Even with *fork()*,  
 48825 it is wrong to call *exit()*, since buffered data would then be flushed twice.)

48826 If signal handlers are invoked in the child process after *vfork()*, they must follow the same rules  
 48827 as other code in the child process.

48828 **RATIONALE**

48829 None.

48830 **FUTURE DIRECTIONS**

48831 None.

48832 **SEE ALSO**48833 *exec*, *exit()*, *fork()*, *wait()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**unistd.h**>48834 **CHANGE HISTORY**

48835 First released in Issue 4, Version 2.

48836 **Issue 5**

48837 Moved from X/OPEN UNIX extension to BASE.

48838 **NAME**

48839 fprintf, vfprintf, vsnprintf, vsprintf — format output of a stdarg argument list

48840 **SYNOPSIS**

48841 #include <stdarg.h>

48842 #include <stdio.h>

48843 int vfprintf(FILE \*restrict stream, const char \*restrict format,  
48844 va\_list ap);

48845 int vprintf(const char \*restrict format, va\_list ap);

48846 XSI int vsnprintf(char \*restrict s, size\_t n, const char \*restrict format,  
48847 va\_list ap);

48848 int vsprintf(char \*restrict s, const char \*restrict format, va\_list ap);

48849 **DESCRIPTION**

48850 CX For *vfprintf()*, *vprintf()*, and *vsprintf()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

48854 XSI The *vprintf()*, *vfprintf()*, *vsnprintf()*, and *vsprintf()* functions shall be the same as *printf()*, *fprintf()*, *snprintf()*, and *sprintf()* respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by <stdarg.h>.

48857 These functions do not invoke the *va\_end* macro. As these functions invoke the *va\_arg* macro, the value of *ap* after the return is indeterminate.

48859 **RETURN VALUE**

48860 Refer to *fprintf()*.

48861 **ERRORS**

48862 Refer to *fprintf()*.

48863 **EXAMPLES**

48864 None.

48865 **APPLICATION USAGE**

48866 Applications using these functions should call *va\_end(ap)* afterwards to clean up.

48867 **RATIONALE**

48868 None.

48869 **FUTURE DIRECTIONS**

48870 None.

48871 **SEE ALSO**

48872 *fprintf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdarg.h>, <stdio.h>

48873 **CHANGE HISTORY**

48874 First released in Issue 1. Derived from Issue 1 of the SVID.

48875 **Issue 4**

48876 The APPLICATION USAGE section is added.

48877 The FUTURE DIRECTIONS section is removed.

48878 The following changes are incorporated for alignment with the ISO C standard:

- 48879 • These functions are no longer marked as extensions.

- 48880           • The type of argument *format* is changed from **char\*** to **const char\***.
- 48881           • Reference to the **<varargs.h>** header in the DESCRIPTION is replaced by **<stdarg.h>**. The  
48882           last paragraph has also been added to indicate interactions with the *va\_arg* and *va\_end*  
48883           macros.
- 48884 **Issue 5**
- 48885           The *vsnprintf()* function is added.
- 48886 **Issue 6**
- 48887           The *vfprintf()*, *vprintf()*, *vsnprintf()*, and *vsprintf()* functions are updated for alignment with the  
48888           ISO/IEC 9899:1999 standard.

48889 **NAME**

48890 vfprintf, fprintf, vscanf — format input of a stdarg list

48891 **SYNOPSIS**

48892 #include &lt;stdarg.h&gt;

48893 #include &lt;stdio.h&gt;

48894 int vfprintf(FILE \*restrict stream, const char \*restrict format,

48895 va\_list arg);

48896 int fprintf(const char \*restrict format, va\_list arg);

48897 int vscanf(const char \*restrict s, const char \*restrict format,

48898 va\_list arg);

48899 **DESCRIPTION**48900 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
48901 conflict between the requirements described here and the ISO C standard is unintentional. This  
48902 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.48903 These functions shall be equivalent to the *scanf()*, *fscanf()*, and *sscanf()* functions, respectively,  
48904 except that instead of being called with a variable number of arguments, they are called with an  
48905 argument list as defined by the <stdarg.h> header. These functions do not invoke the *va\_end*  
48906 macro. As these functions invoke the *va\_arg* macro, the value of *ap* after the return is  
48907 indeterminate.48908 **RETURN VALUE**48909 Refer to *fscanf()*.48910 **ERRORS**48911 Refer to *fscanf()*.48912 **EXAMPLES**

48913 None.

48914 **APPLICATION USAGE**48915 Applications using these functions should call *va\_end(ap)* afterwards to clean up.48916 **RATIONALE**

48917 None.

48918 **FUTURE DIRECTIONS**

48919 None.

48920 **SEE ALSO**48921 *fscanf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdarg.h>, <stdio.h>48922 **CHANGE HISTORY**

48923 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

48924 **NAME**

48925 vfwprintf, vswprintf, vwprintf — wide-character formatted output of a stdarg argument list

48926 **SYNOPSIS**

48927 #include &lt;stdarg.h&gt;

48928 #include &lt;stdio.h&gt;

48929 #include &lt;wchar.h&gt;

48930 int vfwprintf(FILE \*restrict *stream*, const wchar\_t \*restrict *format*,  
48931 va\_list *arg*);48932 int vswprintf(wchar\_t \*restrict *ws*, size\_t *n*, const wchar\_t \*restrict *format*,  
48933 va\_list *arg*);48934 int vwprintf(const wchar\_t \*restrict *format*, va\_list *arg*);48935 **DESCRIPTION**48936 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
48937 conflict between the requirements described here and the ISO C standard is unintentional. This  
48938 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.48939 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* functions shall be the same as *fwprintf()*, *swprintf()*,  
48940 and *wprintf()* respectively, except that instead of being called with a variable number of  
48941 arguments, they are called with an argument list as defined by <stdarg.h>.48942 These functions do not invoke the *va\_end* macro. However, as these functions do invoke the  
48943 *va\_arg* macro, the value of *ap* after the return is indeterminate.48944 **RETURN VALUE**48945 Refer to *fwprintf()*.48946 **ERRORS**48947 Refer to *fwprintf()*.48948 **EXAMPLES**

48949 None.

48950 **APPLICATION USAGE**48951 Applications using these functions should call *va\_end(ap)* afterwards to clean up.48952 **RATIONALE**

48953 None.

48954 **FUTURE DIRECTIONS**

48955 None.

48956 **SEE ALSO**48957 *fwprintf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdarg.h>, <stdio.h>,  
48958 <wchar.h>48959 **CHANGE HISTORY**48960 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
48961 (E).48962 **Issue 6**48963 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* prototypes are updated for alignment with the  
48964 ISO/IEC 9899:1999 standard. ()

48965 **NAME**

48966 vfwscanf, vswscanf, vwscanf — wide-character formatted input of a stdarg list

48967 **SYNOPSIS**

48968 #include <stdarg.h>

48969 #include <stdio.h>

48970 #include <wchar.h>

48971 int vfwscanf(FILE \*restrict *stream*, const wchar\_t \*restrict *format*,  
48972 va\_list *arg*);

48973 int vswscanf(const wchar\_t \*restrict *ws*, const wchar\_t \*restrict *format*,  
48974 va\_list *arg*);

48975 int vwscanf(const wchar\_t \*restrict *format*, va\_list *arg*);

48976 **DESCRIPTION**

48977 **cx** The functionality described on this reference page is aligned with the ISO C standard. Any  
48978 conflict between the requirements described here and the ISO C standard is unintentional. This  
48979 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

48980 These functions shall be equivalent to the *fwscanf()*, *swscanf()*, and *wscanf()* functions,  
48981 respectively, except that instead of being called with a variable number of arguments, they are  
48982 called with an argument list as defined by the <stdarg.h> header. These functions do not invoke  
48983 the *va\_end* macro. As these functions invoke the *va\_arg* macro, the value of *ap* after the return is  
48984 indeterminate.

48985 **RETURN VALUE**

48986 Refer to *fwscanf()*.

48987 **ERRORS**

48988 Refer to *fwscanf()*.

48989 **EXAMPLES**

48990 None.

48991 **APPLICATION USAGE**

48992 Applications using these functions should call *va\_end(ap)* afterwards to clean up.

48993 **RATIONALE**

48994 None.

48995 **FUTURE DIRECTIONS**

48996 None.

48997 **SEE ALSO**

48998 *fwscanf()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <stdarg.h>, <stdio.h>,  
48999 <wchar.h>

49000 **CHANGE HISTORY**

49001 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.



49002 **NAME**

49003 vprintf, vsnprintf, vsprintf — format output of a stdarg argument list

49004 **SYNOPSIS**

49005 #include &lt;stdarg.h&gt;

49006 #include &lt;stdio.h&gt;

49007 int vprintf(const char \**format*, va\_list *ap*);49008 XSI int vsnprintf(char \**s*, size\_t *n*, const char \**format*, va\_list *ap*);49009 int vsprintf(char \**s*, const char \**format*, va\_list *ap*);49010 **DESCRIPTION**49011 Refer to *fprintf()*.

49012 **NAME**

49013           vscanf, vsscanf — format input of a stdarg list

49014 **SYNOPSIS**

49015           #include &lt;stdarg.h&gt;

49016           #include &lt;stdio.h&gt;

49017           int vscanf(const char \*restrict *format*, va\_list *arg*);49018           int vsscanf(const char \*restrict *s*, const char \*restrict *format*,49019                 va\_list *arg*);49020 **DESCRIPTION**49021           Refer to *vfscanf()*.

49022 **NAME**

49023 vswprintf, vwprintf — wide-character formatted output of a stdarg argument list

49024 **SYNOPSIS**

49025 #include &lt;stdarg.h&gt;

49026 #include &lt;stdio.h&gt;

49027 #include &lt;wchar.h&gt;

49028 int vswprintf(wchar\_t \*ws, size\_t n, const wchar\_t \*format, |  
49029 va\_list arg); |

49030 int vwprintf(const wchar\_t \*format, va\_list arg); |

49031 **DESCRIPTION** |49032 Refer to *vwprintf()*. |

49033 **NAME**

49034           vswscanf, vwscanf — wide-character formatted input of a stdarg list

49035 **SYNOPSIS**

49036           #include &lt;stdarg.h&gt;

49037           #include &lt;stdio.h&gt;

49038           #include &lt;wchar.h&gt;

49039           int vswscanf(const wchar\_t \*restrict *ws*, const wchar\_t \*restrict *format*,  
49040                       va\_list *arg*);49041           int vwscanf(const wchar\_t \*restrict *format*, va\_list *arg*);49042 **DESCRIPTION**49043           Refer to *vfwscanf()*.

## 49044 NAME

49045 wait, waitpid — wait for a child process to stop or terminate

## 49046 SYNOPSIS

49047 #include &lt;sys/wait.h&gt;

49048 pid\_t wait(int \*stat\_loc);

49049 pid\_t waitpid(pid\_t pid, int \*stat\_loc, int options);

## 49050 DESCRIPTION

49051 The *wait()* and *waitpid()* functions allow the calling process to obtain status information  
 49052 pertaining to one of its child processes. Various options permit status information to be obtained  
 49053 for child processes that have terminated or stopped. If status information is available for two or  
 49054 more child processes, the order in which their status is reported is unspecified.

49055 The *wait()* function shall suspend execution of the calling thread until status information for one  
 49056 of the terminated child processes of the calling process is available, or until delivery of a signal  
 49057 whose action is either to execute a signal-catching function or to terminate the process. If more  
 49058 than one thread is suspended in *wait()* or *waitpid()* awaiting termination of the same process,  
 49059 exactly one thread shall return the process status at the time of the target process termination. If  
 49060 status information is available prior to the call to *wait()*, return shall be immediate.

49061 The *waitpid()* function shall behave identically to *wait()* if the *pid* argument is **(pid\_t)-1** and the  
 49062 *options* argument is 0. Otherwise, its behavior shall be modified by the values of the *pid* and  
 49063 *options* arguments.

49064 The *pid* argument specifies a set of child processes for which *status* is requested. The *waitpid()*  
 49065 function shall only return the status of a child process from this set:

- 49066 • If *pid* is equal to **(pid\_t)-1**, *status* is requested for any child process. In this respect, *waitpid()*  
 49067 is then equivalent to *wait()*.
- 49068 • If *pid* is greater than 0, it specifies the process ID of a single child process for which *status* is  
 49069 requested.
- 49070 • If *pid* is 0, *status* is requested for any child process whose process group ID is equal to that of  
 49071 the calling process.
- 49072 • If *pid* is less than **(pid\_t)-1**, *status* is requested for any child process whose process group ID  
 49073 is equal to the absolute value of *pid*.

49074 The *options* argument is constructed from the bitwise-inclusive OR of zero or more of the  
 49075 following flags, defined in the header <sys/wait.h>:

49076 XSI WCONTINUED The *waitpid()* function shall report the status of any continued child process  
 49077 specified by *pid* whose status has not been reported since it continued from a  
 49078 job control stop.

49079 WNOHANG The *waitpid()* function shall not suspend execution of the calling thread if  
 49080 *status* is not immediately available for one of the child processes specified by  
 49081 *pid*.

49082 WUNTRACED The status of any child processes specified by *pid* that are stopped, and whose  
 49083 status has not yet been reported since they stopped, shall also be reported to  
 49084 the requesting process.

49085 XSI If the calling process has SA\_NOCLDWAIT set or has SIGCHLD set to SIG\_IGN, and the  
 49086 process has no unwaited-for children that were transformed into zombie processes, the calling  
 49087 thread shall block until all of the children of the process containing the calling thread terminate,  
 49088 and *wait()* and *waitpid()* shall fail and set *errno* to [ECHILD].

49089 If *wait()* or *waitpid()* return because the status of a child process is available, these functions  
 49090 shall return a value equal to the process ID of the child process. In this case, if the value of the  
 49091 argument *stat\_loc* is not a null pointer, information shall be stored in the location pointed to by  
 49092 *stat\_loc*. The value stored at the location pointed to by *stat\_loc* shall be 0 if and only if the status  
 49093 returned is from a terminated child process that terminated by one of the following means:

- 49094 1. The process returned 0 from *main()*.
- 49095 2. The process called *\_exit()* or *exit()* with a *status* argument of 0.
- 49096 3. The process was terminated because the last thread in the process terminated.

49097 Regardless of its value, this information may be interpreted using the following macros, which  
 49098 are defined in `<sys/wait.h>` and evaluate to integral expressions; the *stat\_val* argument is the  
 49099 integer value pointed to by *stat\_loc*.

49100 **WIFEXITED(*stat\_val*)**

49101 Evaluates to a non-zero value if *status* was returned for a child process that terminated  
 49102 normally.

49103 **WEXITSTATUS(*stat\_val*)**

49104 If the value of **WIFEXITED(*stat\_val*)** is non-zero, this macro evaluates to the low-order 8 bits  
 49105 of the *status* argument that the child process passed to *\_exit()* or *exit()*, or the value the child  
 49106 process returned from *main()*.

49107 **WIFSIGNALED(*stat\_val*)**

49108 Evaluates to non-zero value if *status* was returned for a child process that terminated due to  
 49109 the receipt of a signal that was not caught (see `<signal.h>`).

49110 **WTERMSIG(*stat\_val*)**

49111 If the value of **WIFSIGNALED(*stat\_val*)** is non-zero, this macro evaluates to the number of  
 49112 the signal that caused the termination of the child process.

49113 **WIFSTOPPED(*stat\_val*)**

49114 Evaluates to a non-zero value if *status* was returned for a child process that is currently  
 49115 stopped.

49116 **WSTOPSIG(*stat\_val*)**

49117 If the value of **WIFSTOPPED(*stat\_val*)** is non-zero, this macro evaluates to the number of the  
 49118 signal that caused the child process to stop.

49119 XSI **WIFCONTINUED(*stat\_val*)**

49120 Evaluates to a non-zero value if *status* was returned for a child process that has continued  
 49121 from a job control stop.

49122 SPN It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes  
 49123 created by *posix\_spawn()* or *posix\_spawnnp()* may indicate a **WIFSTOPPED(*stat\_val*)** before  
 49124 subsequent calls to *wait()* or *waitpid()* indicate **WIFEXITED(*stat\_val*)** as the result of an error  
 49125 detected before the new process image starts executing.

49126 It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes  
 49127 created by *posix\_spawn()* or *posix\_spawnnp()* may indicate a **WIFSIGNALED(*stat\_val*)** if a signal is  
 49128 sent to the parent's process group after *posix\_spawn()* or *posix\_spawnnp()* is called.

49129 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that specified the  
 49130 XSI **WUNTRACED** flag and did not specify the **WCONTINUED** flag, exactly one of the macros  
 49131 **WIFEXITED(\**stat\_loc*)**, **WIFSIGNALED(\**stat\_loc*)**, and **WIFSTOPPED(\**stat\_loc*)** shall evaluate to  
 49132 a non-zero value.

49133 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that specified the  
 49134 XSI WUNTRACED and WCONTINUED flags, exactly one of the macros WIFEXITED(\**stat\_loc*),  
 49135 XSI WIFSIGNALED(\**stat\_loc*), WIFSTOPPED(\**stat\_loc*), and WIFCONTINUED(\**stat\_loc*) shall  
 49136 evaluate to a non-zero value.

49137 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that did not specify the  
 49138 XSI WUNTRACED or WCONTINUED flags, or by a call to the *wait()* function, exactly one of the  
 49139 macros WIFEXITED(\**stat\_loc*) and WIFSIGNALED(\**stat\_loc*) shall evaluate to a non-zero value.

49140 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that did not specify the  
 49141 XSI WUNTRACED flag and specified the WCONTINUED flag, or by a call to the *wait()* function,  
 49142 XSI exactly one of the macros WIFEXITED(\**stat\_loc*), WIFSIGNALED(\**stat\_loc*), and  
 49143 WIFCONTINUED(\**stat\_loc*) shall evaluate to a non-zero value.

49144 If `_POSIX_REALTIME_SIGNALS` is defined, and the implementation queues the SIGCHLD  
 49145 signal, then if *wait()* or *waitpid()* returns because the status of a child process is available, any  
 49146 pending SIGCHLD signal associated with the process ID of the child process shall be discarded.  
 49147 Any other pending SIGCHLD signals shall remain pending.

49148 Otherwise, if SIGCHLD is blocked, if *wait()* or *waitpid()* return because the status of a child  
 49149 process is available, any pending SIGCHLD signal shall be cleared unless the status of another  
 49150 child process is available.

49151 For all other conditions, it is unspecified whether child *status* will be available when a SIGCHLD  
 49152 signal is delivered.

49153 There may be additional implementation-defined circumstances under which *wait()* or *waitpid()*  
 49154 report *status*. This shall not occur unless the calling process or one of its child processes explicitly  
 49155 makes use of a non-standard extension. In these cases the interpretation of the reported *status* is  
 49156 implementation-defined.

49157 XSI If a parent process terminates without waiting for all of its child processes to terminate, the  
 49158 remaining child processes shall be assigned a new parent process ID corresponding to an  
 49159 implementation-defined system process.

#### 49160 RETURN VALUE

49161 If *wait()* or *waitpid()* returns because the status of a child process is available, these functions  
 49162 shall return a value equal to the process ID of the child process for which *status* is reported. If  
 49163 *wait()* or *waitpid()* returns due to the delivery of a signal to the calling process, `-1` shall be  
 49164 returned and *errno* set to `[EINTR]`. If *waitpid()* was invoked with `WNOHANG` set in *options*, it  
 49165 has at least one child process specified by *pid* for which *status* is not available, and *status* is not  
 49166 available for any process specified by *pid*, `0` is returned. Otherwise, `(pid_t)-1` shall be returned,  
 49167 and *errno* set to indicate the error.

#### 49168 ERRORS

49169 The *wait()* function shall fail if:

49170 `[ECHILD]` The calling process has no existing unwaited-for child processes.

49171 `[EINTR]` The function was interrupted by a signal. The value of the location pointed to  
 49172 by *stat\_loc* is undefined.

49173 The *waitpid()* function shall fail if:

49174 `[ECHILD]` The process specified by *pid* does not exist or is not a child of the calling  
 49175 process, or the process group specified by *pid* does not exist or does not have  
 49176 any member process that is a child of the calling process.

49177 [EINTR] The function was interrupted by a signal. The value of the location pointed to  
49178 by *stat\_loc* is undefined.

49179 [EINVAL] The *options* argument is not valid.

#### 49180 EXAMPLES

49181 None.

#### 49182 APPLICATION USAGE

49183 None.

#### 49184 RATIONALE

49185 A call to the *wait()* or *waitpid()* function only returns *status* on an immediate child process of the  
49186 calling process; that is, a child that was produced by a single *fork()* call (perhaps followed by an  
49187 *exec* or other function calls) from the parent. If a child produces grandchildren by further use of  
49188 *fork()*, none of those grandchildren nor any of their descendants affect the behavior of a *wait()*  
49189 from the original parent process. Nothing in this volume of IEEE Std. 1003.1-200x prevents an  
49190 implementation from providing extensions that permit a process to get *status* from a grandchild  
49191 or any other process, but a process that does not use such extensions must be guaranteed to see  
49192 *status* from only its direct children.

49193 The *waitpid()* function is provided for three reasons:

- 49194 1. To support job control
- 49195 2. To permit a non-blocking version of the *wait()* function
- 49196 3. To permit a library routine, such as *system()* or *pclose()*, to wait for its children without  
49197 interfering with other terminated children for which the process has not waited

49198 The first two of these facilities are based on the *wait3()* function provided by 4.3 BSD. The  
49199 function uses the *options* argument, which is identical to an argument to *wait3()*. The  
49200 WUNTRACED flag is used only in conjunction with job control on systems supporting job  
49201 control. Its name comes from 4.3 BSD and refers to the fact that there are two types of stopped  
49202 processes in that implementation: processes being traced via the *ptrace()* debugging facility and  
49203 (untraced) processes stopped by job control signals. Since *ptrace()* is not part of this volume of  
49204 IEEE Std. 1003.1-200x, only the second type is relevant. The name WUNTRACED was retained  
49205 because its usage is the same, even though the name is not intuitively meaningful in this context.

49206 The third reason for the *waitpid()* function is to permit independent sections of a process to  
49207 spawn and wait for children without interfering with each other. For example, the following  
49208 problem occurs in developing a portable shell, or command interpreter:

```
49209 stream = popen("/bin/true");
49210 (void) system("sleep 100");
49211 (void) pclose(stream);
```

49212 On all historical implementations, the final *pclose()* fails to reap the *wait()* *status* of the *popen()*.

49213 The status values are retrieved by macros, rather than given as specific bit encodings as they are  
49214 in most historical implementations (and thus expected by existing programs). This was  
49215 necessary to eliminate a limitation on the number of signals an implementation can support that  
49216 was inherent in the traditional encodings. This volume of IEEE Std. 1003.1-200x does require that  
49217 a *status* value of zero corresponds to a process calling *\_exit(0)*, as this is the most common  
49218 encoding expected by existing programs. Some of the macro names were adopted from 4.3 BSD.

49219 These macros syntactically operate on an arbitrary integer value. The behavior is undefined  
49220 unless that value is one stored by a successful call to *wait()* or *waitpid()* in the location pointed  
49221 to by the *stat\_loc* argument. An early proposal attempted to make this clearer by specifying each



49222 argument as *\*stat\_loc* rather than *stat\_val*. However, that did not follow the conventions of other  
 49223 specifications in this volume of IEEE Std. 1003.1-200x or traditional usage. It also could have  
 49224 implied that the argument to the macro must literally be *\*stat\_loc*; in fact, that value can be  
 49225 stored or passed as an argument to other functions before being interpreted by these macros.

49226 The extension that affects *wait()* and *waitpid()* and is common in historical implementations is  
 49227 the *ptrace()* function. It is called by a child process and causes that child to stop and return a  
 49228 *status* that appears identical to the *status* indicated by WIFSTOPPED. The *status* of *ptrace()*  
 49229 children is traditionally returned regardless of the WUNTRACED flag (or by the *wait()*  
 49230 function). Most applications do not need to concern themselves with such extensions because  
 49231 they have control over what extensions they or their children use. However, applications, such  
 49232 as command interpreters, that invoke arbitrary processes may see this behavior when those  
 49233 arbitrary processes misuse such extensions.

49234 Implementations that support *core* file creation or other implementation-defined actions on  
 49235 termination of some processes traditionally provide a bit in the *status* returned by *wait()* to  
 49236 indicate that such actions have occurred.

49237 Allowing the *wait()* family of functions to discard a pending SIGCHLD signal that is associated  
 49238 with a successfully waited-for child process puts them into the *sigwait()* and *sigwaitinfo()*  
 49239 category with respect to SIGCHLD.

49240 This definition allows implementations to treat a pending SIGCHLD signal as accepted by the  
 49241 process in *wait()*, with the same meaning of “accepted” as when that word is applied to the  
 49242 *sigwait()* family of functions.

49243 Allowing the *wait()* family of functions to behave this way permits an implementation to be able  
 49244 to deal precisely with SIGCHLD signals.

49245 In particular, an implementation that does accept (discard) the SIGCHLD signal can make the  
 49246 following guarantees regardless of the queuing depth of signals in general (the list of waitable  
 49247 children can hold the SIGCHLD queue):

- 49248 1. If a SIGCHLD signal handler is established via *sigaction()* without the SA\_RESETHAND  
 49249 flag, SIGCHLD signals can be accurately counted; that is, exactly one SIGCHLD signal will  
 49250 be delivered to or accepted by the process for every child process that terminates.
- 49251 2. A single *wait()* issued from a SIGCHLD signal handler can be guaranteed to return  
 49252 immediately with status information for a child process.
- 49253 3. When SA\_SIGINFO is requested, the SIGCHLD signal handler can be guaranteed to  
 49254 receive a non-NULL pointer to a **siginfo\_t** structure that describes a child process for  
 49255 which a wait via *waitpid()* or *waitid()* will not block or fail.
- 49256 4. The *system()* function will not cause a process's SIGCHLD handler to be called as a result of  
 49257 the *fork()/exec* executed within *system()* because *system()* will accept the SIGCHLD signal  
 49258 when it performs a *waitpid()* for its child process. This is a desirable behavior of *system()*  
 49259 so that it can be used in a library without causing side effects to the application linked with  
 49260 the library.

49261 An implementation that does not permit the *wait()* family of functions to accept (discard) a  
 49262 pending SIGCHLD signal associated with a successfully waited-for child, cannot make the  
 49263 guarantees described above for the following reasons:

#### 49264 Guarantee #1

49265 Although it might be assumed that reliable queuing of all SIGCHLD signals generated by  
 49266 the system can make this guarantee, the counter example is the case of a process that blocks  
 49267 SIGCHLD and performs an indefinite loop of *fork()/wait()* operations. If the

49268 implementation supports queued signals, then eventually the system will run out of  
 49269 memory for the queue. The guarantee cannot be made because there must be some limit to  
 49270 the depth of queuing.

49271 Guarantees #2 and #3

49272 These cannot be guaranteed unless the *wait()* family of functions accepts the SIGCHLD  
 49273 signal. Otherwise, a *fork()/wait()* executed while SIGCHLD is blocked (as in the *system()*  
 49274 function) will result in an invocation of the handler when SIGCHLD is unblocked, after the  
 49275 process has disappeared.

49276 Guarantee #4

49277 Although possible to make this guarantee, *system()* would have to set the SIGCHLD  
 49278 handler to SIG\_DFL so that the SIGCHLD signal generated by its *fork()* would be discarded  
 49279 (the SIGCHLD default action is to be ignored), then restore it to its previous setting. This  
 49280 would have the undesirable side effect of discarding all SIGCHLD signals pending to the  
 49281 process.

#### 49282 FUTURE DIRECTIONS

49283 None.

#### 49284 SEE ALSO

49285 *exec*, *exit()*, *fork()*, *waitid()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <*sys/types.h*>,  
 49286 <*sys/wait.h*>

#### 49287 CHANGE HISTORY

49288 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 49289 Issue 4

49290 The <*sys/types.h*> header is now marked as optional (OH); this header need not be included on  
 49291 XSI-conformant systems.

49292 Error return values throughout the DESCRIPTION and RETURN VALUE sections are changed  
 49293 to show the proper casting (that is, **(pid\_t)-1**).

49294 The words “If the implementation supports job control” are removed from the description of  
 49295 WUNTRACED. This is because job control is defined as mandatory for Issue 4 conforming  
 49296 implementations.

49297 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 49298 • Text describing conditions under which 0 is returned when WNOHANG is set in *options* is  
 49299 added to the RETURN VALUE section.

#### 49300 Issue 4, Version 2

49301 The *waitpid()* function is added.

49302 The following changes are incorporated in the DESCRIPTION for X/OPEN UNIX conformance:

- 49303 • The WCONTINUED *options* flag and the WIFCONTINUED(*stat\_val*) macro are added.
- 49304 • Text following the list of *options* flags explains the implications of setting the  
 49305 SA\_NOCLDWAIT signal flag, or setting SIGCHLD to SIG\_IGN.
- 49306 • Text following the list of macros, which explains what macros return non-zero values in  
 49307 certain cases, is expanded and the value of the WCONTINUED flag on the previous call to  
 49308 *waitpid()* is taken into account.

49309 **Issue 5**

49310 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

49311 **Issue 6**

49312 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

49313 The following new requirements on POSIX implementations derive from alignment with the  
49314 Single UNIX Specification:

- 49315 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
49316 required for conforming implementations of previous POSIX specifications, it was not  
49317 required for UNIX applications.

49318 The following changes were made to align with the IEEE P1003.1a draft standard:

- 49319 • The processing of the SIGCHLD signal and the [ECHILD] error is clarified.

49320 The semantics of `WIFSTOPPED(stat_val)`, `WIFEXITED(stat_val)`, and `WIFSIGNALED(stat_val)`  
49321 are defined with respect to `posix_spawn()` or `posix_spawnp()` for alignment with  
49322 IEEE Std. 1003.1d-1999.

49323 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

49324 **NAME**

49325 waitid — wait for a child process to change state

49326 **SYNOPSIS**49327 XSI `#include <sys/wait.h>`49328 `int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);`

49329

49330 **DESCRIPTION**

49331 The *waitid()* function shall suspend the calling thread until one child of the process containing  
 49332 the calling thread changes state. It records the current state of a child in the structure pointed to  
 49333 by *infop*. If a child process changed state prior to the call to *waitid()*, *waitid()* returns  
 49334 immediately. If more than one thread is suspended in *wait()* or *waitpid()* waiting termination of  
 49335 the same process, exactly one thread returns the process status at the time of the target process  
 49336 termination

49337 The *idtype* and *id* arguments are used to specify which children *waitid()* waits for.

49338 If *idtype* is P\_PID, *waitid()* shall wait for the child with a process ID equal to (**pid\_t**)*id*.

49339 If *idtype* is P\_PGID, *waitid()* shall wait for any child with a process group ID equal to (**pid\_t**)*id*.

49340 If *idtype* is P\_ALL, *waitid()* shall wait for any children and *id* is ignored.

49341 The *options* argument is used to specify which state changes *waitid()* shall wait for. It is formed  
 49342 by OR'ing together one or more of the following flags:

49343 WEXITED Wait for processes that have exited.

49344 WSTOPPED Status shall be returned for any child that has stopped upon receipt of a signal.

49345 WCONTINUED Status shall be returned for any child that was stopped and has been  
 49346 continued.

49347 WNOHANG Return immediately if there are no children to wait for.

49348 WNOWAIT Keep the process whose status is returned in *infop* in a waitable state. This  
 49349 shall not affect the state of the process; the process may be waited for again  
 49350 after this call completes.

49351 The application shall ensure that the *infop* argument points to a **siginfo\_t** structure. If *waitid()*  
 49352 returns because a child process was found that satisfied the conditions indicated by the  
 49353 arguments *idtype* and *options*, then the structure pointed to by *infop* shall be filled in by the  
 49354 system with the status of the process. The *si\_signo* member shall always be equal to SIGCHLD.

49355 **RETURN VALUE**49356 **Notes to Reviewers**

49357 *This section with side shading will not appear in the final copy. - Ed.*

49358 D1, XSH, ERN 416 points out an omission. The following text is proposed: "If WNOHANG was  
 49359 specified and there are no children to wait for, 0 shall be returned."

49360 If *waitid()* returns due to the change of state of one of its children, 0 shall be returned. Otherwise,  
 49361 -1 shall be returned and *errno* set to indicate the error.

49362 **ERRORS**

49363 The *waitid()* function shall fail if:

49364 [ECHILD] The calling process has no existing unwaited-for child processes.

49365 [EINTR] The *waitid()* function was interrupted by a signal. |  
49366 [EINVAL] An invalid value was specified for *options*, or *idtype* and *id* specify an invalid |  
49367 set of processes. |

49368 **EXAMPLES**

49369 None.

49370 **APPLICATION USAGE**

49371 None.

49372 **RATIONALE**

49373 None.

49374 **FUTURE DIRECTIONS**

49375 None.

49376 **SEE ALSO**

49377 *exec*, *exit()*, *wait()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <sys/wait.h> |

49378 **CHANGE HISTORY**

49379 First released in Issue 4, Version 2.

49380 **Issue 5**

49381 Moved from X/OPEN UNIX extension to BASE.

49382 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

49383 **Issue 6**

49384 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49385 **NAME**

49386       waitpid — wait for a child process to stop or terminate

49387 **SYNOPSIS**

49388       #include <sys/wait.h>

49389       pid\_t waitpid(pid\_t *pid*, int *\*stat\_loc*, int *options*);

49390 **DESCRIPTION**

49391       Refer to *wait()*.

## 49392 NAME

49393 wrtomb — convert a wide-character code to a character (restartable)

## 49394 SYNOPSIS

49395 #include &lt;stdio.h&gt;

49396 size\_t wrtomb(char \*restrict *s*, wchar\_t *wc*, mbstate\_t \*restrict *ps*);

## 49397 DESCRIPTION

49398 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 49399 conflict between the requirements described here and the ISO C standard is unintentional. This  
 49400 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

49401 If *s* is a null pointer, the *wrtomb()* function shall be equivalent to the call:49402 `wrtomb(buf, L'\0', ps)`49403 where *buf* is an internal buffer.

49404 If *s* is not a null pointer, the *wrtomb()* function shall determine the number of bytes needed to  
 49405 represent the character that corresponds to the wide character given by *wc* (including any shift  
 49406 sequences), and stores the resulting bytes in the array whose first element is pointed to by *s*. At  
 49407 most {MB\_CUR\_MAX} bytes are stored. If *wc* is a null wide character, a null byte is stored,  
 49408 preceded by any shift sequence needed to restore the initial shift state. The resulting state  
 49409 described is the initial conversion state.

49410 If *ps* is a null pointer, the *wrtomb()* function uses its own internal **mbstate\_t** object, which is  
 49411 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
 49412 pointed to by *ps* is used to completely describe the current conversion state of the associated  
 49413 character sequence. The implementation shall behave as if no function defined in this volume of  
 49414 IEEE Std. 1003.1-200x calls *wrtomb()*.

49415 cx If the application uses any of the `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS`  
 49416 functions, the application shall ensure that the *wrtomb()* function is called with a non-NULL *ps*  
 49417 argument.

49418 xsi The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.

## 49419 RETURN VALUE

49420 The *wrtomb()* function shall return the number of bytes stored in the array object (including any  
 49421 shift sequences). When *wc* is not a valid wide character, an encoding error shall occur. In this  
 49422 case, the function shall store the value of the macros [EILSEQ] in *errno* and shall return  
 49423 (**size\_t**)-1; the conversion state shall be undefined.

## 49424 ERRORS

49425 The *wrtomb()* function may fail if:49426 cx [EINVAL] *ps* points to an object that contains an invalid conversion state.

49427 [EILSEQ] Invalid wide-character code is detected.

49428 **EXAMPLES**

49429 None.

49430 **APPLICATION USAGE**

49431 None.

49432 **RATIONALE**

49433 None.

49434 **FUTURE DIRECTIONS**

49435 None.

49436 **SEE ALSO**49437 *mbstinit()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**wchar.h**>49438 **CHANGE HISTORY**49439 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
49440 (E).49441 **Issue 6**

49442 In the DESCRIPTION, a note on using this function in a threaded application is added.

49443 Extensions beyond the ISO C standard are now marked.

49444 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49445 The *wrtomb()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.



49446 **NAME**49447 `wscat` — concatenate two wide-character strings49448 **SYNOPSIS**49449 `#include <wchar.h>`49450 `wchar_t *wscat(wchar_t *restrict ws1, const wchar_t *restrict ws2);`49451 **DESCRIPTION**

49452 `CX` The functionality described on this reference page is aligned with the ISO C standard. Any  
49453 conflict between the requirements described here and the ISO C standard is unintentional. This  
49454 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

49455 The `wscat()` function shall append a copy of the wide-character string pointed to by `ws2`  
49456 (including the terminating null wide-character code) to the end of the wide-character string  
49457 pointed to by `ws1`. The initial wide-character code of `ws2` overwrites the null wide-character  
49458 code at the end of `ws1`. If copying takes place between objects that overlap, the behavior is  
49459 undefined.

49460 **RETURN VALUE**49461 The `wscat()` function shall return `ws1`; no return value is reserved to indicate an error.49462 **ERRORS**

49463 No errors are defined.

49464 **EXAMPLES**

49465 None.

49466 **APPLICATION USAGE**

49467 None.

49468 **RATIONALE**

49469 None.

49470 **FUTURE DIRECTIONS**

49471 None.

49472 **SEE ALSO**49473 `wscncat()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<wchar.h>`49474 **CHANGE HISTORY**

49475 First released in Issue 4. Derived from the MSE working draft.

49476 **Issue 6**49477 The Open Group corrigenda item U040/2 has been applied. In the RETURN VALUE section, `s1`  
49478 is changed to `ws1`.49479 The `wscat()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49480 **NAME**49481 `wcschr` — wide-character string scanning operation49482 **SYNOPSIS**49483 `#include <wchar.h>`49484 `wchar_t *wcschr(const wchar_t *ws, wchar_t wc);`49485 **DESCRIPTION**

49486 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49487 conflict between the requirements described here and the ISO C standard is unintentional. This  
49488 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

49489 The `wcschr()` function shall locate the first occurrence of `wc` in the wide-character string pointed  
49490 to by `ws`. The application shall ensure that the value of `wc` is a character representable as a type  
49491 **wchar\_t** and a wide-character code corresponding to a valid character in the current locale. The  
49492 terminating null wide-character code is considered to be part of the wide-character string.

49493 **RETURN VALUE**

49494 Upon completion, `wcschr()` shall return a pointer to the wide-character code, or a null pointer if  
49495 the wide-character code is not found.

49496 **ERRORS**

49497 No errors are defined.

49498 **EXAMPLES**

49499 None.

49500 **APPLICATION USAGE**

49501 None.

49502 **RATIONALE**

49503 None.

49504 **FUTURE DIRECTIONS**

49505 None.

49506 **SEE ALSO**49507 `wcsrchr()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<wchar.h>`49508 **CHANGE HISTORY**

49509 First released in Issue 4. Derived from the MSE working draft.

49510 **Issue 6**

49511 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49512 **NAME**

49513           wcsncmp — compare two wide-character strings

49514 **SYNOPSIS**

49515           #include &lt;wchar.h&gt;

49516           int wcsncmp(const wchar\_t \*ws1, const wchar\_t \*ws2);

49517 **DESCRIPTION**

49518 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
49519 conflict between the requirements described here and the ISO C standard is unintentional. This  
49520 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

49521       The *wcsncmp()* function shall compare the wide-character string pointed to by *ws1* to the wide-  
49522 character string pointed to by *ws2*.

49523       The sign of a non-zero return value is determined by the sign of the difference between the  
49524 values of the first pair of wide-character codes that differ in the objects being compared.

49525 **RETURN VALUE**

49526       Upon completion, *wcsncmp()* shall return an integer greater than, equal to, or less than 0, if the  
49527 wide-character string pointed to by *ws1* is greater than, equal to, or less than the wide-character  
49528 string pointed to by *ws2*, respectively.

49529 **ERRORS**

49530       No errors are defined.

49531 **EXAMPLES**

49532       None.

49533 **APPLICATION USAGE**

49534       None.

49535 **RATIONALE**

49536       None.

49537 **FUTURE DIRECTIONS**

49538       None.

49539 **SEE ALSO**49540       *wcsncmp()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <wchar.h>49541 **CHANGE HISTORY**

49542       First released in Issue 4. Derived from the MSE working draft.

49543 **NAME**

49544 wscoll — wide-character string comparison using collating information

49545 **SYNOPSIS**

49546 #include &lt;wchar.h&gt;

49547 int wscoll(const wchar\_t \*ws1, const wchar\_t \*ws2);

49548 **DESCRIPTION**49549 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
49550 conflict between the requirements described here and the ISO C standard is unintentional. This  
49551 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.49552 The *wscoll()* function shall compare the wide-character string pointed to by *ws1* to the wide-  
49553 character string pointed to by *ws2*, both interpreted as appropriate to the *LC\_COLLATE* category  
49554 of the current locale.49555 CX The *wscoll()* function shall not change the setting of *errno* if successful.49556 An application wishing to check for error situations should set *errno* to 0 before calling *wscoll()*.  
49557 If *errno* is non-zero on return, an error has occurred.49558 **RETURN VALUE**49559 Upon successful completion, *wscoll()* shall return an integer greater than, equal to, or less than  
49560 0, according to whether the wide-character string pointed to by *ws1* is greater than, equal to, or  
49561 less than the wide-character string pointed to by *ws2*, when both are interpreted as appropriate  
49562 CX to the current locale. On error, *wscoll()* may set *errno*, but no return value is reserved to indicate  
49563 an error.49564 **ERRORS**49565 The *wscoll()* function may fail if:49566 [EINVAL] The *ws1* or *ws2* arguments contain wide-character codes outside the domain of  
49567 the collating sequence.49568 **EXAMPLES**

49569 None.

49570 **APPLICATION USAGE**49571 The *wcsxfrm()* and *wscmp()* functions should be used for sorting large lists.49572 **RATIONALE**

49573 None.

49574 **FUTURE DIRECTIONS**

49575 None.

49576 **SEE ALSO**49577 *wscmp()*, *wcsxfrm()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <wchar.h>49578 **CHANGE HISTORY**

49579 First released in Issue 4. Derived from the MSE working draft.

49580 **Issue 5**

49581 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

49582 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

49583 **NAME**

49584           wcscpy — copy a wide-character string

49585 **SYNOPSIS**

49586           #include &lt;wchar.h&gt;

49587           wchar\_t \*wcscpy(wchar\_t \*restrict *ws1*, const wchar\_t \*restrict *ws2*);49588 **DESCRIPTION**49589 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
49590 conflict between the requirements described here and the ISO C standard is unintentional. This  
49591 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.49592       The *wcscpy()* function shall copy the wide-character string pointed to by *ws2* (including the  
49593 terminating null wide-character code) into the array pointed to by *ws1*. If copying takes place  
49594 between objects that overlap, the behavior is undefined.49595 **RETURN VALUE**49596       The *wcscpy()* function shall return *ws1*; no return value is reserved to indicate an error.49597 **ERRORS**

49598       No errors are defined.

49599 **EXAMPLES**

49600       None.

49601 **APPLICATION USAGE**

49602       None.

49603 **RATIONALE**

49604       None.

49605 **FUTURE DIRECTIONS**

49606       None.

49607 **SEE ALSO**49608       *wscncpy()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**wchar.h**>49609 **CHANGE HISTORY**

49610       First released in Issue 4. Derived from the MSE working draft.

49611 **Issue 6**49612       The *wcscpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49613 **NAME**

49614 wscspn — get length of a complementary wide substring

49615 **SYNOPSIS**

49616 #include &lt;wchar.h&gt;

49617 size\_t wscspn(const wchar\_t \*ws1, const wchar\_t \*ws2);

49618 **DESCRIPTION**

49619 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
49620 conflict between the requirements described here and the ISO C standard is unintentional. This  
49621 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

49622 The *wscspn()* function shall compute the length of the maximum initial segment of the wide-  
49623 character string pointed to by *ws1* which consists entirely of wide-character codes *not* from the  
49624 wide-character string pointed to by *ws2*.

49625 **RETURN VALUE**

49626 The *wscspn()* function shall return the length of the initial substring of *ws1*; no return value is  
49627 reserved to indicate an error.

49628 **ERRORS**

49629 No errors are defined.

49630 **EXAMPLES**

49631 None.

49632 **APPLICATION USAGE**

49633 None.

49634 **RATIONALE**

49635 None.

49636 **FUTURE DIRECTIONS**

49637 None.

49638 **SEE ALSO**49639 *wcspn()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <wchar.h>49640 **CHANGE HISTORY**

49641 First released in Issue 4. Derived from the MSE working draft.

49642 **Issue 5**

49643 The RETURN VALUE section is updated to indicate that *wscspn()* returns the length of *ws1*,  
49644 rather than *ws1* itself.

49645 **NAME**49646 `wcsftime` — convert date and time to a wide-character string49647 **SYNOPSIS**49648 `#include <wchar.h>`49649 `size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,`  
49650 `const wchar_t *restrict format, const struct tm *restrict timptr);`49651 **DESCRIPTION**49652 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49653 conflict between the requirements described here and the ISO C standard is unintentional. This  
49654 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.49655 The `wcsftime()` function shall be equivalent to the `strftime()` function, except that:

- 49656 • The argument `wcs` points to the initial element of an array of wide characters into which the  
49657 generated output is to be placed.
- 49658 • The argument `maxsize` indicates the maximum number of wide characters to be placed in the  
49659 output array.
- 49660 • The argument `format` is a wide-character string and the conversion specifications are replaced  
49661 by corresponding sequences of wide characters.
- 49662 • The return value indicates the number of wide characters placed in the output array.

49663 If copying takes place between objects that overlap, the behavior is undefined.

49664 **RETURN VALUE**49665 If the total number of resulting wide-character codes including the terminating null wide-  
49666 character code is no more than `maxsize`, `wcsftime()` shall return the number of wide-character  
49667 codes placed into the array pointed to by `wcs`, not including the terminating null wide-character  
49668 code.49669 **ERRORS**

49670 No errors are defined.

49671 **EXAMPLES**

49672 None.

49673 **APPLICATION USAGE**

49674 None.

49675 **RATIONALE**

49676 None.

49677 **FUTURE DIRECTIONS**

49678 None.

49679 **SEE ALSO**49680 `strftime()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<wchar.h>`49681 **CHANGE HISTORY**

49682 First released in Issue 4.

49683 **Issue 5**

49684 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

49685 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of the `format`  
49686 argument is changed from `const char*` to `const wchar_t*`.

49687 **Issue 6**

49688

The *wcsftime()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.



49689 **NAME**49690 `wcslen` — get wide-character string length49691 **SYNOPSIS**49692 `#include <wchar.h>`49693 `size_t wcslen(const wchar_t *ws);`49694 **DESCRIPTION**

49695 *cx* The functionality described on this reference page is aligned with the ISO C standard. Any  
49696 conflict between the requirements described here and the ISO C standard is unintentional. This  
49697 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

49698 The `wcslen()` function shall compute the number of wide-character codes in the wide-character  
49699 string to which *ws* points, not including the terminating null wide-character code.

49700 **RETURN VALUE**

49701 The `wcslen()` function shall return the length of *ws*; no return value is reserved to indicate an  
49702 error.

49703 **ERRORS**

49704 No errors are defined.

49705 **EXAMPLES**

49706 None.

49707 **APPLICATION USAGE**

49708 None.

49709 **RATIONALE**

49710 None.

49711 **FUTURE DIRECTIONS**

49712 None.

49713 **SEE ALSO**49714 The Base Definitions volume of IEEE Std. 1003.1-200x, `<wchar.h>`49715 **CHANGE HISTORY**

49716 First released in Issue 4. Derived from the MSE working draft.

49717 **NAME**

49718       wcsncat — concatenate a wide-character string with part of another

49719 **SYNOPSIS**

49720       #include <wchar.h>

49721       wchar\_t \*wcsncat(wchar\_t \*restrict *ws1*, const wchar\_t \*restrict *ws2*,  
49722                   size\_t *n*);

49723 **DESCRIPTION**

49724 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
49725       conflict between the requirements described here and the ISO C standard is unintentional. This  
49726       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

49727       The *wcsncat()* function shall append not more than *n* wide-character codes (a null wide-  
49728       character code and wide-character codes that follow it are not appended) from the array pointed  
49729       to by *ws2* to the end of the wide-character string pointed to by *ws1*. The initial wide-character  
49730       code of *ws2* overwrites the null wide-character code at the end of *ws1*. A terminating null wide-  
49731       character code is always appended to the result. If copying takes place between objects that  
49732       overlap, the behavior is undefined.

49733 **RETURN VALUE**

49734       The *wcsncat()* function shall return *ws1*; no return value is reserved to indicate an error.

49735 **ERRORS**

49736       No errors are defined.

49737 **EXAMPLES**

49738       None.

49739 **APPLICATION USAGE**

49740       None.

49741 **RATIONALE**

49742       None.

49743 **FUTURE DIRECTIONS**

49744       None.

49745 **SEE ALSO**

49746       *wscat()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <wchar.h>

49747 **CHANGE HISTORY**

49748       First released in Issue 4. Derived from the MSE working draft.

49749 **Issue 6**

49750       The *wcsncat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49751 **NAME**49752        `wcsncmp` — compare part of two wide-character strings49753 **SYNOPSIS**49754        `#include <wchar.h>`49755        `int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);`49756 **DESCRIPTION**49757 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
49758 conflict between the requirements described here and the ISO C standard is unintentional. This  
49759 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.49760        The `wcsncmp()` function shall compare not more than *n* wide-character codes (wide-character  
49761 codes that follow a null wide-character code are not compared) from the array pointed to by *ws1*  
49762 to the array pointed to by *ws2*.49763        The sign of a non-zero return value is determined by the sign of the difference between the  
49764 values of the first pair of wide-character codes that differ in the objects being compared.49765 **RETURN VALUE**49766        Upon successful completion, `wcsncmp()` shall return an integer greater than, equal to, or less  
49767 than 0, if the possibly null-terminated array pointed to by *ws1* is greater than, equal to, or less  
49768 than the possibly null-terminated array pointed to by *ws2*, respectively.49769 **ERRORS**

49770        No errors are defined.

49771 **EXAMPLES**

49772        None.

49773 **APPLICATION USAGE**

49774        None.

49775 **RATIONALE**

49776        None.

49777 **FUTURE DIRECTIONS**

49778        None.

49779 **SEE ALSO**49780        `wscmp()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<wchar.h>`49781 **CHANGE HISTORY**

49782        First released in Issue 4. Derived from the MSE working draft.

49783 **NAME**49784 `wcsncpy` — copy part of a wide-character string49785 **SYNOPSIS**49786 `#include <wchar.h>`49787 `wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,`  
49788 `size_t n);`49789 **DESCRIPTION**49790 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49791 conflict between the requirements described here and the ISO C standard is unintentional. This  
49792 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.49793 The `wcsncpy()` function shall copy not more than *n* wide-character codes (wide-character codes  
49794 that follow a null wide-character code are not copied) from the array pointed to by *ws2* to the  
49795 array pointed to by *ws1*. If copying takes place between objects that overlap, the behavior is  
49796 undefined.49797 If the array pointed to by *ws2* is a wide-character string that is shorter than *n* wide-character  
49798 codes, null wide-character codes are appended to the copy in the array pointed to by *ws1*, until *n*  
49799 wide-character codes in all are written.49800 **RETURN VALUE**49801 The `wcsncpy()` function shall return *ws1*; no return value is reserved to indicate an error.49802 **ERRORS**

49803 No errors are defined.

49804 **EXAMPLES**

49805 None.

49806 **APPLICATION USAGE**49807 If there is no null wide-character code in the first *n* wide-character codes of the array pointed to  
49808 by *ws2*, the result is not null-terminated.49809 **RATIONALE**

49810 None.

49811 **FUTURE DIRECTIONS**

49812 None.

49813 **SEE ALSO**49814 `wscpy()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<wchar.h>`49815 **CHANGE HISTORY**

49816 First released in Issue 4. Derived from the MSE working draft.

49817 **Issue 6**49818 The `wcsncpy()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49819 **NAME**

49820 wcpbrk — scan wide-character string for a wide-character code

49821 **SYNOPSIS**

49822 #include <wchar.h>

49823 wchar\_t \*wcpbrk(const wchar\_t \*ws1, const wchar\_t \*ws2);

49824 **DESCRIPTION**

49825 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
49826 conflict between the requirements described here and the ISO C standard is unintentional. This  
49827 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

49828 The *wcpbrk()* function shall locate the first occurrence in the wide-character string pointed to by  
49829 *ws1* of any wide-character code from the wide-character string pointed to by *ws2*.

49830 **RETURN VALUE**

49831 Upon successful completion, *wcpbrk()* shall return a pointer to the wide-character code or a null  
49832 pointer if no wide-character code from *ws2* occurs in *ws1*.

49833 **ERRORS**

49834 No errors are defined.

49835 **EXAMPLES**

49836 None.

49837 **APPLICATION USAGE**

49838 None.

49839 **RATIONALE**

49840 None.

49841 **FUTURE DIRECTIONS**

49842 None.

49843 **SEE ALSO**

49844 *wchr()*, *wchr()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <wchar.h>

49845 **CHANGE HISTORY**

49846 First released in Issue 4. Derived from the MSE working draft.

49847 **NAME**

49848           wcsrchr — wide-character string scanning operation

49849 **SYNOPSIS**

49850           #include &lt;wchar.h&gt;

49851           wchar\_t \*wcsrchr(const wchar\_t \*ws, wchar\_t wc);

49852 **DESCRIPTION**

49853 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
49854       conflict between the requirements described here and the ISO C standard is unintentional. This  
49855       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

49856       The *wcsrchr()* function shall locate the last occurrence of *wc* in the wide-character string pointed  
49857       to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type  
49858       **wchar\_t** and a wide-character code corresponding to a valid character in the current locale. The  
49859       terminating null wide-character code is considered to be part of the wide-character string.

49860 **RETURN VALUE**

49861       Upon successful completion, *wcsrchr()* shall return a pointer to the wide-character code or a null  
49862       pointer if *wc* does not occur in the wide-character string.

49863 **ERRORS**

49864       No errors are defined.

49865 **EXAMPLES**

49866       None.

49867 **APPLICATION USAGE**

49868       None.

49869 **RATIONALE**

49870       None.

49871 **FUTURE DIRECTIONS**

49872       None.

49873 **SEE ALSO**49874       *wcscr()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**wchar.h**>49875 **CHANGE HISTORY**

49876       First released in Issue 4. Derived from the MSE working draft.

49877 **Issue 6**

49878       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49879 **NAME**49880 `wcsrtombs` — convert a wide-character string to a character string (restartable)49881 **SYNOPSIS**49882 `#include <wchar.h>`49883 `size_t wcsrtombs(char *restrict dst, const wchar_t **restrict src,`  
49884 `size_t len, mbstate_t *restrict ps);`49885 **DESCRIPTION**49886 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49887 conflict between the requirements described here and the ISO C standard is unintentional. This  
49888 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.49889 The `wcsrtombs()` function shall convert a sequence of wide characters from the array indirectly  
49890 pointed to by `src` into a sequence of corresponding characters, beginning in the conversion state  
49891 described by the object pointed to by `ps`. If `dst` is not a null pointer, the converted characters are  
49892 then stored into the array pointed to by `dst`. Conversion continues up to and including a  
49893 terminating null wide character, which is also stored. Conversion stops earlier in the following  
49894 cases:

- 49895
- When a code is reached that does not correspond to a valid character
  - When the next character would exceed the limit of `len` total bytes to be stored in the array  
49896 pointed to by `dst` (and `dst` is not a null pointer)
- 49897

49898 Each conversion takes place as if by a call to the `wcrtomb()` function.49899 If `dst` is not a null pointer, the pointer object pointed to by `src` is assigned either a null pointer (if  
49900 conversion stopped due to reaching a terminating null wide character) or the address just past  
49901 the last wide character converted (if any). If conversion stopped due to reaching a terminating  
49902 null wide character, the resulting state described is the initial conversion state.49903 If `ps` is a null pointer, the `wcsrtombs()` function uses its own internal `mbstate_t` object, which is  
49904 initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t` object  
49905 pointed to by `ps` is used to completely describe the current conversion state of the associated  
49906 character sequence. The implementation shall behave as if no function defined in this volume of  
49907 IEEE Std. 1003.1-200x calls `wcsrtombs()`.49908 **CX** If the application uses any of the `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS`  
49909 functions, the application shall ensure that the `wcsrtombs()` function is called with a non-NULL  
49910 `ps` argument.49911 **XSI** The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.49912 **RETURN VALUE**49913 If conversion stops because a code is reached that does not correspond to a valid character, an  
49914 encoding error occurs. In this case, the `wcsrtombs()` function shall store the value of the macro  
49915 `[EILSEQ]` in `errno` and return `(size_t)-1`; the conversion state is undefined. Otherwise, it shall  
49916 return the number of bytes in the resulting character sequence, not including the terminating  
49917 null (if any).49918 **ERRORS**49919 The `wcsrtombs()` function may fail if:

- 49920
- CX**
- `[EINVAL]`
- `ps`
- points to an object that contains an invalid conversion state.
- 
- 49921
- `[EILSEQ]`
- A wide-character code does not correspond to a valid character.

49922 **EXAMPLES**

49923 None.

49924 **APPLICATION USAGE**

49925 None.

49926 **RATIONALE**

49927 None.

49928 **FUTURE DIRECTIONS**

49929 None.

49930 **SEE ALSO**49931 *mbsinit()*, *wcrtomb()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <wchar.h>49932 **CHANGE HISTORY**49933 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
49934 (E).49935 **Issue 6**

49936 In the DESCRIPTION, a note on using this function in a threaded application is added.

49937 Extensions beyond the ISO C standard are now marked.

49938 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49939 The *wcsrombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.



49940 **NAME**49941 `wcsspnp` — get length of a wide substring49942 **SYNOPSIS**49943 `#include <wchar.h>`49944 `size_t wcsspnp(const wchar_t *ws1, const wchar_t *ws2);`49945 **DESCRIPTION**

49946 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49947 conflict between the requirements described here and the ISO C standard is unintentional. This  
49948 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

49949 The `wcsspnp()` function shall compute the length of the maximum initial segment of the wide-  
49950 character string pointed to by `ws1` which consists entirely of wide-character codes from the  
49951 wide-character string pointed to by `ws2`.

49952 **RETURN VALUE**

49953 The `wcsspnp()` function shall return the length of the initial substring of `ws1`; no return value is  
49954 reserved to indicate an error.

49955 **ERRORS**

49956 No errors are defined.

49957 **EXAMPLES**

49958 None.

49959 **APPLICATION USAGE**

49960 None.

49961 **RATIONALE**

49962 None.

49963 **FUTURE DIRECTIONS**

49964 None.

49965 **SEE ALSO**49966 `wcscspnp()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<wchar.h>`49967 **CHANGE HISTORY**

49968 First released in Issue 4. Derived from the MSE working draft.

49969 **Issue 5**

49970 The RETURN VALUE section is updated to indicate that `wcsspnp()` returns the length of `ws1`  
49971 rather than `ws1` itself.

49972 **NAME**49973 `wcsstr` — find a wide-character substring49974 **SYNOPSIS**49975 `#include <wchar.h>`49976 `wchar_t *wcsstr(const wchar_t *restrict ws1, const wchar_t *restrict ws2);`49977 **DESCRIPTION**49978 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49979 conflict between the requirements described here and the ISO C standard is unintentional. This  
49980 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.49981 The `wcsstr()` function shall locate the first occurrence in the wide-character string pointed to by  
49982 `ws1` of the sequence of wide characters (excluding the terminating null wide character) in the  
49983 wide-character string pointed to by `ws2`.49984 **RETURN VALUE**49985 Upon successful completion, `wcsstr()` shall return a pointer to the located wide-character string,  
49986 or a null pointer if the wide-character string is not found.49987 If `ws2` points to a wide-character string with zero length, the function shall return `ws1`.49988 **ERRORS**

49989 No errors are defined.

49990 **EXAMPLES**

49991 None.

49992 **APPLICATION USAGE**

49993 None.

49994 **RATIONALE**

49995 None.

49996 **FUTURE DIRECTIONS**

49997 None.

49998 **SEE ALSO**49999 `wcschr()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<wchar.h>`50000 **CHANGE HISTORY**50001 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50002 (E).50003 **Issue 6**50004 The `wcsstr()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

## 50005 NAME

50006 `wcstod`, `wcstof`, `wcstold` — convert a wide-character string to a double-precision number

## 50007 SYNOPSIS

50008 `#include <wchar.h>`

50009 `double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);`

50010 `float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);`

50011 `long double wcstold(const wchar_t *restrict nptr,`

50012 `wchar_t **restrict endptr);`

## 50013 DESCRIPTION

50014 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50015 conflict between the requirements described here and the ISO C standard is unintentional. This  
50016 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

50017 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to  
50018 **double**, **float**, and **long double** representation, respectively. First, they decompose the input  
50019 wide-character string into three parts:

- 50020 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by  
50021 `iswspace()`)
- 50022 2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
- 50023 3. A final wide-character string of one or more unrecognized wide-character codes, including  
50024 the terminating null wide-character code of the input wide-character string

50025 Then it attempts to convert the subject sequence to a floating-point number, and returns the  
50026 result.

50027 The expected form of the subject sequence is an optional plus or minus sign, then one of the  
50028 following:

- 50029 • A non-empty sequence of decimal digits optionally containing a radix character, then an  
50030 optional exponent part
- 50031 • A `0x` or `0X`, then a non-empty sequence of hexadecimal digits optionally containing a radix  
50032 character, then an optional binary exponent part
- 50033 • One of `INF` or `INFINITY`, or any other wide string equivalent except for case
- 50034 • One of `NAN` or `NAN(n-wchar-sequenceopt)`, or any other wide string ignoring case in the NAN  
50035 part, where:

50036 `n-wchar-sequence:`

50037 `digit`

50038 `nondigit`

50039 `n-wchar-sequence digit`

50040 `n-wchar-sequence nondigit`

50041 The subject sequence is defined as the longest initial subsequence of the input wide string,  
50042 starting with the first non-white-space wide character, that is of the expected form. The subject  
50043 sequence contains no wide characters if the input wide string is not of the expected form.

50044 If the subject sequence has the expected form for a floating-point number, the sequence of wide  
50045 characters starting with the first digit or the radix character (whichever occurs first) is  
50046 interpreted as a floating constant according to the rules of the C language, except that the radix  
50047 character is used in place of a period, and that if neither an exponent part nor a radix character  
50048 appears in a decimal floating-point number, or if a binary exponent part does not appear in a

50049 hexadecimal floating-point number, an exponent part of the appropriate type with value zero is  
 50050 assumed to follow the last digit in the string. If the subject sequence begins with a minus sign,  
 50051 the sequence is interpreted as negated. A wide-character sequence INF or INFINITY is  
 50052 interpreted as an infinity, if representable in the return type, else like a floating constant that is  
 50053 too large for the range of the return type. A wide-character sequence NAN or NAN(*n-wchar-*  
 50054 *sequence<sub>opt</sub>*) is interpreted as a quiet NaN, if supported in the return type, else like a subject  
 50055 sequence part that does not have the expected form; the meaning of the *n-wchar* sequences is  
 50056 implementation-defined. A pointer to the final wide string is stored in the object pointed to by  
 50057 *endptr*, provided that *endptr* is not a null pointer.

50058 If the subject sequence has the hexadecimal form and FLT\_RADIX is a power of 2, the value  
 50059 resulting from the conversion is correctly rounded.

50060 CX The radix character is defined in the program's locale (category *LC\_NUMERIC*). In the POSIX  
 50061 locale, or in a locale where the radix character is not defined, the radix character shall default to a  
 50062 period ('.').

50063 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
 50064 accepted.

50065 If the subject sequence is empty or does not have the expected form, no conversion is performed;  
 50066 the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null  
 50067 pointer.

50068 The *wcstod()* function shall not change the setting of *errno* if successful.

50069 Because 0 is returned on error and is also a valid return on success, an application wishing to  
 50070 check for error situations should set *errno* to 0, then call *wcstod()*, then check *errno*.

#### 50071 RETURN VALUE

50072 Upon successful completion, these functions shall return the converted value. If no conversion  
 50073 could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

50074 If the correct value is outside the range of representable values, HUGE\_VAL, HUGE\_VALF, or  
 50075 HUGE\_VALL shall be returned (according to the sign of the value), and *errno* shall be set to  
 50076 [ERANGE].

50077 If the correct value would cause underflow, a value whose magnitude is no greater than the  
 50078 smallest normalized positive number in the return type shall be returned and *errno* set to  
 50079 [ERANGE].

#### 50080 ERRORS

50081 The *wcstod()* function shall fail if:

50082 [ERANGE] The value to be returned would cause overflow or underflow.

50083 The *wcstod()* function may fail if:

50084 CX [EINVAL] No conversion could be performed.

50085 **EXAMPLES**

50086 None.

50087 **APPLICATION USAGE**

50088 If the subject sequence has the hexadecimal form and FLT\_RADIX is not a power of 2, the result  
 50089 should be one of the two numbers in the appropriate internal format that are adjacent to the  
 50090 hexadecimal floating source value, with the extra stipulation that the error should have a correct  
 50091 sign for the current rounding direction.

50092 If the subject sequence has the decimal form and at most DECIMAL\_DIG (defined in <float.h>)  
 50093 significant digits, the result should be correctly rounded. If the subject sequence *D* has the  
 50094 decimal form and more than DECIMAL\_DIG significant digits, consider the two bounding,  
 50095 adjacent decimal strings *L* and *U*, both having DECIMAL\_DIG significant digits, such that the  
 50096 values of *L*, *D*, and *U* satisfy " $L \leq D \leq U$ ". The result should be one of the (equal or  
 50097 adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current  
 50098 rounding direction, with the extra stipulation that the error with respect to *D* should have a  
 50099 correct sign for the current rounding direction.

50100 **RATIONALE**

50101 None.

50102 **FUTURE DIRECTIONS**

50103 None.

50104 **SEE ALSO**

50105 *iswspace()*, *localeconv()*, *scanf()*, *setlocale()*, *wcstol()*, the Base Definitions volume of  
 50106 IEEE Std. 1003.1-200x, <float.h>, <wchar.h>, the Base Definitions volume of  
 50107 IEEE Std. 1003.1-200x, Chapter 7, Locale

50108 **CHANGE HISTORY**

50109 First released in Issue 4. Derived from the MSE working draft.

50110 **Issue 5**50111 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.50112 **Issue 6**

50113 Extensions beyond the ISO C standard are now marked.

50114 The following new requirements on POSIX implementations derive from alignment with the  
 50115 Single UNIX Specification:

50116 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
 50117 added if no conversion could be performed.

50118 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

50119 • The *wcstod()* prototype is updated.50120 • The *wcstof()* and *wcstold()* functions are added.

50121 • The DESCRIPTION, RETURN VALUE, and APPLICATION USAGE sections are extensively  
 50122 updated.

50123 **NAME**

50124           wcstoimax, wcstoumax — convert wide-character string to integer type

50125 **SYNOPSIS**

50126           #include <stddef.h>

50127           #include <inttypes.h>

50128           intmax\_t wcstoimax(const wchar\_t \*restrict nptr,

50129                            wchar\_t \*\*restrict endptr, int base);

50130           uintmax\_t wcstoumax(const wchar\_t \*restrict nptr,

50131                            wchar\_t \*\*restrict endptr, int base);

50132 **DESCRIPTION**

50133 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
50134 conflict between the requirements described here and the ISO C standard is unintentional. This  
50135 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

50136       These functions shall be equivalent to the *wcstol()*, *wcstoll()*, *wcstoul()*, and *wcstoull()* functions,  
50137 respectively, except that the initial portion of the wide string shall be converted to **intmax\_t** and  
50138 **uintmax\_t** representation, respectively.

50139 **RETURN VALUE**

50140       These functions shall return the converted value, if any.

50141       If no conversion could be performed, zero shall be returned. If the correct value is outside the  
50142 range of representable values, {INTMAX\_MAX}, {INTMAX\_MIN}, or {UINTMAX\_MAX} shall  
50143 be returned (according to the return type and sign of the value, if any), and *errno* shall be set to  
50144 [ERANGE].

50145 **ERRORS**

50146       These functions shall fail if:

50147       [EINVAL]       The value of *base* is not supported.

50148       [ERANGE]      The value to be returned is not representable.

50149       These functions may fail if:

50150       [EINVAL]      No conversion could be performed.

50151 **EXAMPLES**

50152       None.

50153 **APPLICATION USAGE**

50154       None.

50155 **RATIONALE**

50156       None.

50157 **FUTURE DIRECTIONS**

50158       None.

50159 **SEE ALSO**

50160       *wcstol()*, *wcstoul()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <inttypes.h>,  
50161 <stddef.h>

50162 **CHANGE HISTORY**

50163       First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

50164 **NAME**50165 `wcstok` — split wide-character string into tokens50166 **SYNOPSIS**50167 `#include <wchar.h>`50168 `wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict ws2,`  
50169 `wchar_t **restrict ptr);`50170 **DESCRIPTION**50171 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
50172 conflict between the requirements described here and the ISO C standard is unintentional. This  
50173 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.50174 A sequence of calls to `wcstok()` breaks the wide-character string pointed to by `ws1` into a  
50175 sequence of tokens, each of which is delimited by a wide-character code from the wide-character  
50176 string pointed to by `ws2`. The third argument points to a caller-provided `wchar_t` pointer into  
50177 which the `wcstok()` function stores information necessary for it to continue scanning the same  
50178 wide-character string.50179 The first call in the sequence has `ws1` as its first argument, and is followed by calls with a null  
50180 pointer as their first argument. The separator string pointed to by `ws2` may be different from call  
50181 to call.50182 The first call in the sequence searches the wide-character string pointed to by `ws1` for the first  
50183 wide-character code that is *not* contained in the current separator string pointed to by `ws2`. If no  
50184 such wide-character code is found, then there are no tokens in the wide-character string pointed  
50185 to by `ws1` and `wcstok()` returns a null pointer. If such a wide-character code is found, it is the start  
50186 of the first token.50187 The `wcstok()` function then searches from there for a wide-character code that *is* contained in the  
50188 current separator string. If no such wide-character code is found, the current token extends to  
50189 the end of the wide-character string pointed to by `ws1`, and subsequent searches for a token shall  
50190 return a null pointer. If such a wide-character code is found, it is overwritten by a null wide-  
50191 character, which terminates the current token. The `wcstok()` function saves a pointer to the  
50192 following wide-character code, from which the next search for a token shall start.50193 Each subsequent call, with a null pointer as the value of the first argument, starts searching from  
50194 the saved pointer and behaves as described above.50195 The implementation shall behave as if no function calls `wcstok()`.50196 **RETURN VALUE**50197 Upon successful completion, the `wcstok()` function shall return a pointer to the first wide-  
50198 character code of a token. Otherwise, if there is no token, `wcstok()` shall return a null pointer.50199 **ERRORS**

50200 No errors are defined.

50201 **EXAMPLES**

50202           None.

50203 **APPLICATION USAGE**

50204           None.

50205 **RATIONALE**

50206           None.

50207 **FUTURE DIRECTIONS**

50208           None.

50209 **SEE ALSO**50210           The Base Definitions volume of IEEE Std. 1003.1-200x, <**wchar.h**>50211 **CHANGE HISTORY**

50212           First released in Issue 4.

50213 **Issue 5**50214           Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, a third argument is  
50215           added to the definition of this function in the SYNOPSIS.50216 **Issue 6**50217           The *wcstok()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.



## 50218 NAME

50219 `wcstol`, `wcstoll` — convert a wide-character string to a long integer

## 50220 SYNOPSIS

50221 `#include <wchar.h>`

50222 `long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr,`  
 50223 `int base);`

50224 `long long wcstoll(const wchar_t *restrict nptr,`  
 50225 `wchar_t **restrict endptr, int base);`

## 50226 DESCRIPTION

50227 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 50228 conflict between the requirements described here and the ISO C standard is unintentional. This  
 50229 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

50230 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to  
 50231 **long**, **long long**, **unsigned long**, and **unsigned long long** representation, respectively. First, they  
 50232 decompose the input string into three parts:

- 50233 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by  
 50234 `iswspace()`)
- 50235 2. A subject sequence interpreted as an integer represented in some radix determined by the  
 50236 value of *base*
- 50237 3. A final wide-character string of one or more unrecognized wide-character codes, including  
 50238 the terminating null wide-character code of the input wide-character string

50239 Then it attempts to convert the subject sequence to an integer, and returns the result.

50240 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,  
 50241 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal  
 50242 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal  
 50243 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'  
 50244 only. A hexadecimal constant consists of the prefix "0x" or "0X" followed by a sequence of the  
 50245 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

50246 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
 50247 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
 50248 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'  
 50249 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less  
 50250 than that of *base* are permitted. If the value of *base* is 16, the wide-character code representations  
 50251 of "0x" or "0X" may optionally precede the sequence of letters and digits, following the sign if  
 50252 present.

50253 The subject sequence is defined as the longest initial subsequence of the input wide-character  
 50254 string, starting with the first non-white-space wide-character code that is of the expected form.  
 50255 The subject sequence contains no wide-character codes if the input wide-character string is  
 50256 empty or consists entirely of white-space wide-character code, or if the first non-white-space  
 50257 wide-character code is other than a sign or a permissible letter or digit.

50258 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes  
 50259 starting with the first digit is interpreted as an integer constant. If the subject sequence has the  
 50260 expected form and the value of *base* is between 2 and 36, it is used as the base for conversion,  
 50261 ascribing to each letter its value as given above. If the subject sequence begins with a minus sign,  
 50262 the value resulting from the conversion is negated. A pointer to the final wide-character string is  
 50263 stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

- 50264 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
50265 accepted.
- 50266 If the subject sequence is empty or does not have the expected form, no conversion is performed;  
50267 the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null  
50268 pointer.
- 50269 The *wcstol()* function shall not change the setting of *errno* if successful.
- 50270 Because 0, {LONG\_MIN} or {LLONG\_MIN} and {LONG\_MAX} or {LLONG\_MAX} are returned  
50271 on error and are also valid returns on success, an application wishing to check for error  
50272 situations should set *errno* to 0, then call *wcstol()*, then check *errno*.
- 50273 **RETURN VALUE**
- 50274 Upon successful completion, these functions shall return the converted value, if any. If no  
50275 CX conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If  
50276 the correct value is outside the range of representable values, {LONG\_MAX} or {LONG\_MIN}  
50277 shall be returned (according to the sign of the value), and *errno* set to [ERANGE].
- 50278 **ERRORS**
- 50279 These functions shall fail if:
- 50280 CX [EINVAL] The value of *base* is not supported.
- 50281 [ERANGE] The value to be returned is not representable.
- 50282 These functions may fail if:
- 50283 CX [EINVAL] No conversion could be performed.
- 50284 **EXAMPLES**
- 50285 None.
- 50286 **APPLICATION USAGE**
- 50287 None.
- 50288 **RATIONALE**
- 50289 None.
- 50290 **FUTURE DIRECTIONS**
- 50291 None.
- 50292 **SEE ALSO**
- 50293 *iswalph()*, *scanf()*, *wcstod()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <wchar.h>
- 50294 **CHANGE HISTORY**
- 50295 First released in Issue 4. Derived from the MSE working draft.
- 50296 **Issue 5**
- 50297 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
- 50298 **Issue 6**
- 50299 Extensions beyond the ISO C standard are now marked.
- 50300 The following new requirements on POSIX implementations derive from alignment with the  
50301 Single UNIX Specification:
- 50302 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
  - 50303 added if no conversion could be performed.
- 50304 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

50305

- The *wcstol()* prototype is updated.

50306

- The *wcstoll()* function is added.

50307 **NAME**

50308 `wcstombs` — convert a wide-character string to a character string

50309 **SYNOPSIS**

50310 `#include <stdlib.h>`

50311 `size_t wcstombs(char *restrict s, const wchar_t *restrict pwcs,`  
 50312 `size_t n);`

50313 **DESCRIPTION**

50314 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 50315 conflict between the requirements described here and the ISO C standard is unintentional. This  
 50316 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

50317 The `wcstombs()` function shall convert the sequence of wide-character codes that are in the array  
 50318 pointed to by `pwcs` into a sequence of characters that begins in the initial shift state and stores  
 50319 these characters into the array pointed to by `s`, stopping if a character would exceed the limit of `n`  
 50320 total bytes or if a null byte is stored. Each wide-character code is converted as if by a call to  
 50321 `wctomb()`, except that the shift state of `wctomb()` is not affected.

50322 The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.

50323 No more than `n` bytes shall be modified in the array pointed to by `s`. If copying takes place  
 50324 **CX** between objects that overlap, the behavior is undefined. If `s` is a null pointer, `wcstombs()` shall  
 50325 return the length required to convert the entire array regardless of the value of `n`, but no values  
 50326 are stored.

50327 The `wcstombs()` function need not be reentrant. A function that is not required to be reentrant is  
 50328 not required to be thread-safe.

50329 **RETURN VALUE**

50330 If a wide-character code is encountered that does not correspond to a valid character (of one or  
 50331 more bytes each), `wcstombs()` shall return `(size_t)-1`. Otherwise, `wcstombs()` shall return the  
 50332 number of bytes stored in the character array, not including any terminating null byte. The array  
 50333 shall not be null-terminated if the value returned is `n`.

50334 **ERRORS**

50335 The `wcstombs()` function may fail if:

50336 **CX** `[EILSEQ]` A wide-character code does not correspond to a valid character.

50337 **EXAMPLES**

50338 None.

50339 **APPLICATION USAGE**

50340 None.

50341 **RATIONALE**

50342 None.

50343 **FUTURE DIRECTIONS**

50344 None.

50345 **SEE ALSO**

50346 `mblen()`, `mbtowc()`, `mbstowcs()`, `wctomb()`, the Base Definitions volume of IEEE Std. 1003.1-200x,  
 50347 `<stdlib.h>`

50348 **CHANGE HISTORY**

50349 First released in Issue 4. Derived from the ISO C standard.

50350 **Issue 6**

50351 The following new requirements on POSIX implementations derive from alignment with the  
50352 Single UNIX Specification:

- 50353 • The DESCRIPTION states the effect of when *s* is a null pointer.
- 50354 • The [EILSEQ] error condition is added.

50355 The *wcstombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

## 50356 NAME

50357 wcstoul, wcstoull — convert a wide-character string to an unsigned long

## 50358 SYNOPSIS

50359 #include <wchar.h>

50360 long wcstoul(const wchar\_t \*restrict *nptr*, wchar\_t \*\*restrict *endptr*,  
50361 int *base*);

50362 long long wcstoull(const wchar\_t \*restrict *nptr*,  
50363 wchar\_t \*\*restrict *endptr*, int *base*);

## 50364 DESCRIPTION

50365 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50366 conflict between the requirements described here and the ISO C standard is unintentional. This  
50367 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

50368 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to  
50369 **long**, **long long**, **unsigned long**, and **unsigned long long** representation, respectively. First, they  
50370 decompose the input wide-character string into three parts:

- 50371 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by  
50372 *iswspace()*)
- 50373 2. A subject sequence interpreted as an integer represented in some radix determined by the  
50374 value of *base*
- 50375 3. A final wide-character string of one or more unrecognized wide-character codes, including  
50376 the terminating null wide-character code of the input wide-character string

50377 Then it attempts to convert the subject sequence to an unsigned integer, and returns the result.

50378 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,  
50379 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal  
50380 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal  
50381 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'  
50382 only. A hexadecimal constant consists of the prefix "0x" or "0X" followed by a sequence of the  
50383 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

50384 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
50385 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
50386 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'  
50387 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less  
50388 than that of *base* are permitted. If the value of *base* is 16, the wide-character codes "0x" or "0X"  
50389 may optionally precede the sequence of letters and digits, following the sign if present.

50390 The subject sequence is defined as the longest initial subsequence of the input wide-character  
50391 string, starting with the first wide-character code that is not white space and is of the expected  
50392 form. The subject sequence contains no wide-character codes if the input wide-character string is  
50393 empty or consists entirely of white-space wide-character codes, or if the first wide-character  
50394 code that is not white space is other than a sign or a permissible letter or digit.

50395 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes  
50396 starting with the first digit is interpreted as an integer constant. If the subject sequence has the  
50397 expected form and the value of *base* is between 2 and 36, it is used as the base for conversion,  
50398 ascribing to each letter its value as given above. If the subject sequence begins with a minus sign,  
50399 the value resulting from the conversion is negated. A pointer to the final wide-character string is  
50400 stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

50401 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
50402 accepted.

50403 If the subject sequence is empty or does not have the expected form, no conversion is performed;  
50404 the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null  
50405 pointer.

50406 The *wcstoul()* function shall not change the setting of *errno* if successful.

50407 Because 0, {ULONG\_MAX}, and {ULLONG\_MAX} are returned on error and 0 is also a valid  
50408 return on success, an application wishing to check for error situations should set *errno* to 0, then  
50409 call *wcstoul()*, then check *errno*.

#### 50410 RETURN VALUE

50411 Upon successful completion, these functions shall return the converted value, if any. If no  
50412 CX conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If  
50413 the correct value is outside the range of representable values, {ULONG\_MAX} shall be returned  
50414 and *errno* set to [ERANGE].

#### 50415 ERRORS

50416 These functions shall fail if:

50417 CX [EINVAL] The value of *base* is not supported.

50418 [ERANGE] The value to be returned is not representable.

50419 These functions may fail if:

50420 CX [EINVAL] No conversion could be performed.

#### 50421 EXAMPLES

50422 None.

#### 50423 APPLICATION USAGE

50424 None.

#### 50425 RATIONALE

50426 None.

#### 50427 FUTURE DIRECTIONS

50428 None.

#### 50429 SEE ALSO

50430 *iswalpha()*, *scanf()*, *wcstod()*, *wcstol()*, the Base Definitions volume of IEEE Std. 1003.1-200x,  
50431 <wchar.h>

#### 50432 CHANGE HISTORY

50433 First released in Issue 4. Derived from the MSE working draft.

#### 50434 Issue 5

50435 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

#### 50436 Issue 6

50437 Extensions beyond the ISO C standard are now marked.

50438 The following new requirements on POSIX implementations derive from alignment with the  
50439 Single UNIX Specification:

- 50440 • The [EINVAL] error condition is added for when the value of *base* is not supported.

50441 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
50442 added if no conversion could be performed.

50443 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

50444 • The *wcstoul()* prototype is updated.

50445 • The *wcstoull()* function is added.



50446 **NAME**50447           wcswcs — find a wide substring (**LEGACY**)50448 **SYNOPSIS**

50449 xSI       #include &lt;wchar.h&gt;

50450           wchar\_t \*wcswcs(const wchar\_t \*ws1, const wchar\_t \*ws2);

50451

50452 **DESCRIPTION**

50453           The `wcswcs()` function shall locate the first occurrence in the wide-character string pointed to by  
50454           `ws1` of the sequence of wide-character codes (excluding the terminating null wide-character  
50455           code) in the wide-character string pointed to by `ws2`.

50456 **RETURN VALUE**

50457           Upon successful completion, `wcswcs()` shall return a pointer to the located wide-character string  
50458           or a null pointer if the wide-character string is not found.

50459           If `ws2` points to a wide-character string with zero length, the function shall return `ws1`.

50460 **ERRORS**

50461           No errors are defined.

50462 **EXAMPLES**

50463           None.

50464 **APPLICATION USAGE**

50465           This function was not included in the final ISO/IEC 9899:1990/Amendment 1:1995 (E).

50466           Application developers are strongly encouraged to use the `wcsstr()` function instead.50467 **RATIONALE**

50468           None.

50469 **FUTURE DIRECTIONS**

50470           This function may be withdrawn in a future version.

50471 **SEE ALSO**50472           `wcschr()`, `wcsstr()`, the Base Definitions volume of IEEE Std. 1003.1-200x, <**wchar.h**>50473 **CHANGE HISTORY**

50474           First released in Issue 4. Derived from the MSE working draft.

50475 **Issue 5**

50476           Marked EX.

50477 **Issue 6**

50478           This function is marked LEGACY.

50479 **NAME**

50480 `wcswidth` — number of column positions of a wide-character string

50481 **SYNOPSIS**

```
50482 xSI #include <wchar.h>
```

```
50483 int wcswidth(const wchar_t *pwcs, size_t n);
```

50484

50485 **DESCRIPTION**

50486 The `wcswidth()` function shall determine the number of column positions required for  $n$  wide-  
50487 character codes (or fewer than  $n$  wide-character codes if a null wide-character code is  
50488 encountered before  $n$  wide-character codes are exhausted) in the string pointed to by `pwcs`.

50489 **RETURN VALUE**

50490 The `wcswidth()` function either shall return 0 (if `pwcs` points to a null wide-character code), or  
50491 return the number of column positions to be occupied by the wide-character string pointed to by  
50492 `pwcs`, or return -1 (if any of the first  $n$  wide-character codes in the wide-character string pointed  
50493 to by `pwcs` is not a printing wide-character code).

50494 **ERRORS**

50495 No errors are defined.

50496 **EXAMPLES**

50497 None.

50498 **APPLICATION USAGE**

50499 This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the  
50500 return value for a non-printable wide character is not specified.

50501 **RATIONALE**

50502 None.

50503 **FUTURE DIRECTIONS**

50504 None.

50505 **SEE ALSO**

50506 `wcwidth()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<wchar.h>`, the Base Definitions  
50507 volume of IEEE Std. 1003.1-200x, Section 3.106, Column Position

50508 **CHANGE HISTORY**

50509 First released in Issue 4. Derived from the MSE working draft.

50510 **Issue 6**

50511 The Open Group corrigenda item U021/11 has been applied. The function is marked as an  
50512 extension.

50513 **NAME**

50514        wcsxfrm — wide-character string transformation

50515 **SYNOPSIS**

50516        #include &lt;wchar.h&gt;

50517        size\_t wcsxfrm(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2,  
50518                       size\_t n);50519 **DESCRIPTION**50520 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
50521 conflict between the requirements described here and the ISO C standard is unintentional. This  
50522 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.50523        The *wcsxfrm()* function shall transform the wide-character string pointed to by *ws2* and place the  
50524 resulting wide-character string into the array pointed to by *ws1*. The transformation is such that  
50525 if *wscmp()* is applied to two transformed wide strings, it returns a value greater than, equal to,  
50526 or less than 0, corresponding to the result of *wscoll()* applied to the same two original wide-  
50527 character strings. No more than *n* wide-character codes are placed into the resulting array  
50528 pointed to by *ws1*, including the terminating null wide-character code. If *n* is 0, *ws1* is permitted  
50529 to be a null pointer. If copying takes place between objects that overlap, the behavior is  
50530 undefined.50531 **CX**        The *wcsxfrm()* function shall not change the setting of *errno* if successful.50532        Because no return value is reserved to indicate an error, an application wishing to check for error  
50533 situations should set *errno* to 0, then call *wcsxfrm()*, then check *errno*.50534 **RETURN VALUE**50535        The *wcsxfrm()* function shall return the length of the transformed wide-character string (not  
50536 including the terminating null wide-character code). If the value returned is *n* or more, the  
50537 contents of the array pointed to by *ws1* are indeterminate.50538        On error, the *wcsxfrm()* function may set *errno*, but no return value is reserved to indicate an  
50539 error.50540 **ERRORS**50541        The *wcsxfrm()* function may fail if:50542 **CX**        [EINVAL]        The wide-character string pointed to by *ws2* contains wide-character codes  
50543 outside the domain of the collating sequence.50544 **EXAMPLES**

50545        None.

50546 **APPLICATION USAGE**50547        The transformation function is such that two transformed wide-character strings can be ordered  
50548 by *wscmp()* as appropriate to collating sequence information in the program's locale (category  
50549 *LC\_COLLATE*).50550        The fact that when *n* is 0 *ws1* is permitted to be a null pointer is useful to determine the size of  
50551 the *ws1* array prior to making the transformation.50552 **RATIONALE**

50553        None.

50554 **FUTURE DIRECTIONS**

50555 None.

50556 **SEE ALSO**50557 `wscmp()`, `wscoll()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<wchar.h>` |50558 **CHANGE HISTORY**

50559 First released in Issue 4. Derived from the MSE working draft. |

50560 **Issue 5**

50561 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

50562 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.50563 **Issue 6**

50564 In previous versions, this function was required to return -1 on error.

50565 Extensions beyond the ISO C standard are now marked.

50566 The following new requirements on POSIX implementations derive from alignment with the  
50567 Single UNIX Specification:

- 50568
- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
50569 added if no conversion could be performed.

50570 The `wcsxfrm()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard. |

50571 **NAME**

50572 wctob — wide-character to single-byte conversion

50573 **SYNOPSIS**

50574 #include &lt;stdio.h&gt;

50575 #include &lt;wchar.h&gt;

50576 int wctob(wint\_t c);

50577 **DESCRIPTION**

50578 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50579 conflict between the requirements described here and the ISO C standard is unintentional. This  
50580 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

50581 The *wctob()* function shall determine whether *c* corresponds to a member of the extended  
50582 character set whose character representation is a single byte when in the initial shift state.

50583 The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.

50584 **RETURN VALUE**

50585 The *wctob()* function shall return EOF if *c* does not correspond to a character with length one in  
50586 the initial shift state. Otherwise, it shall return the single-byte representation of that character as  
50587 an **unsigned char** converted to **int**.

50588 **ERRORS**

50589 No errors are defined.

50590 **EXAMPLES**

50591 None.

50592 **APPLICATION USAGE**

50593 None.

50594 **RATIONALE**

50595 None.

50596 **FUTURE DIRECTIONS**

50597 None.

50598 **SEE ALSO**50599 *btowc()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <**wchar.h**>50600 **CHANGE HISTORY**

50601 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50602 (E).

50603 **NAME**

50604 wctomb — convert a wide-character code to a character

50605 **SYNOPSIS**

50606 #include &lt;stdlib.h&gt;

50607 int wctomb(char \*s, wchar\_t wchar);

50608 **DESCRIPTION**

50609 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50610 conflict between the requirements described here and the ISO C standard is unintentional. This  
50611 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

50612 The *wctomb()* function shall determine the number of bytes needed to represent the character  
50613 corresponding to the wide-character code whose value is *wchar* (including any change in the  
50614 shift state). It stores the character representation (possibly multiple bytes and any special bytes  
50615 to change shift state) in the array object pointed to by *s* (if *s* is not a null pointer). At most  
50616 {MB\_CUR\_MAX} bytes are stored. If *wchar* is 0, a null byte is stored, preceded by any shift  
50617 sequence needed to restore the initial shift state, and *wctomb()* is left in the initial shift state.

50618 cx The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
50619 state-dependent encoding, this function is placed into its initial state by a call for which its  
50620 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null  
50621 pointer cause the internal state of the function to be altered as necessary. A call with *s* as a null  
50622 pointer causes this function to return a non-zero value if encodings have state dependency, and  
50623 0 otherwise. Changing the *LC\_CTYPE* category causes the shift state of this function to be  
50624 indeterminate.

50625 The *wctomb()* function need not be reentrant. A function that is not required to be reentrant is  
50626 not required to be thread-safe.

50627 The implementation shall behave as if no function defined in this volume of  
50628 IEEE Std. 1003.1-200x calls *wctomb()*.

50629 **RETURN VALUE**

50630 If *s* is a null pointer, *wctomb()* shall return a non-zero or 0 value, if character encodings,  
50631 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *wctomb()*  
50632 shall return -1 if the value of *wchar* does not correspond to a valid character, or return the  
50633 number of bytes that constitute the character corresponding to the value of *wchar*.

50634 In no case shall the value returned be greater than the value of the {MB\_CUR\_MAX} macro.

50635 **ERRORS**

50636 No errors are defined.

50637 **EXAMPLES**

50638 None.

50639 **APPLICATION USAGE**

50640 None.

50641 **RATIONALE**

50642 None.

50643 **FUTURE DIRECTIONS**

50644 None.

50645 **SEE ALSO**

50646            *mblen()*, *mbtowc()*, *mbstowcs()*, *wcstombs()*, the Base Definitions volume of IEEE Std. 1003.1-200x, |  
50647            <stdlib.h>

50648 **CHANGE HISTORY**

50649            First released in Issue 4. Derived from the ANSI C standard. |

50650 **Issue 6**

50651            Extensions beyond the ISO C standard are now marked.

50652            In the DESCRIPTION, a note about reentrancy and thread-safety is added.

50653 **NAME**

50654 wctrans — define character mapping

50655 **SYNOPSIS**

50656 #include &lt;wctype.h&gt;

50657 wctrans\_t wctrans(const char \*charclass);

50658 **DESCRIPTION**

50659 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50660 conflict between the requirements described here and the ISO C standard is unintentional. This  
50661 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

50662 The *wctrans()* function is defined for valid character mapping names identified in the current  
50663 locale. The *charclass* is a string identifying a generic character mapping name for which codeset-  
50664 specific information is required. The following character mapping names are defined in all  
50665 locales: **tolower** and **toupper**.

50666 The function shall return a value of type **wctrans\_t**, which can be used as the second argument  
50667 to subsequent calls of *towctrans()*. The *wctrans()* function determines values of **wctrans\_t**  
50668 according to the rules of the coded character set defined by character mapping information in  
50669 the program's locale (category *LC\_CTYPE*). The values returned by *wctrans()* are valid until a  
50670 call to *setlocale()* that modifies the category *LC\_CTYPE*.

50671 **RETURN VALUE**

50672 cx The *wctrans()* function shall return 0 and may set *errno* to indicate the error if the given character  
50673 mapping name is not valid for the current locale (category *LC\_CTYPE*); otherwise, it shall return  
50674 a non-zero object of type **wctrans\_t** that can be used in calls to *towctrans()*.

50675 **ERRORS**50676 The *wctrans()* function may fail if:

50677 cx [EINVAL] The character mapping name pointed to by *charclass* is not valid in the current  
50678 locale.

50679 **EXAMPLES**

50680 None.

50681 **APPLICATION USAGE**

50682 None.

50683 **RATIONALE**

50684 None.

50685 **FUTURE DIRECTIONS**

50686 None.

50687 **SEE ALSO**50688 *towctrans()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <wctype.h>50689 **CHANGE HISTORY**

50690 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).



50691 **NAME**

50692           wctype — define character class

50693 **SYNOPSIS**

50694           #include &lt;wctype.h&gt;

50695           wctype\_t wctype(const char \*property);

50696 **DESCRIPTION**

50697 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
 50698       conflict between the requirements described here and the ISO C standard is unintentional. This  
 50699       volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

50700       The *wctype()* function is defined for valid character class names as defined in the current locale.  
 50701       The *property* is a string identifying a generic character class for which codeset-specific type  
 50702       information is required. The following character class names are defined in all locales:

|       |              |              |               |
|-------|--------------|--------------|---------------|
| 50703 | <b>alnum</b> | <b>digit</b> | <b>punct</b>  |
| 50704 | <b>alpha</b> | <b>graph</b> | <b>space</b>  |
| 50705 | <b>blank</b> | <b>lower</b> | <b>upper</b>  |
| 50706 | <b>cntrl</b> | <b>print</b> | <b>xdigit</b> |

50707       Additional character class names defined in the locale definition file (category *LC\_CTYPE*) can  
 50708       also be specified.

50709       The function shall return a value of type **wctype\_t**, which can be used as the second argument to  
 50710       subsequent calls of *iswctype()*. The *wctype()* function determines values of **wctype\_t** according  
 50711       to the rules of the coded character set defined by character type information in the program's  
 50712       locale (category *LC\_CTYPE*). The values returned by *wctype()* are valid until a call to *setlocale()*  
 50713       that modifies the category *LC\_CTYPE*.

50714 **RETURN VALUE**

50715       The *wctype()* function shall return 0 if the given character class name is not valid for the current  
 50716       locale (category *LC\_CTYPE*); otherwise, it shall return an object of type **wctype\_t** that can be  
 50717       used in calls to *iswctype()*.

50718 **ERRORS**

50719       No errors are defined.

50720 **EXAMPLES**

50721       None.

50722 **APPLICATION USAGE**

50723       None.

50724 **RATIONALE**

50725       None.

50726 **FUTURE DIRECTIONS**

50727       None.

50728 **SEE ALSO**50729       *iswctype()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <wctype.h>, <wchar.h>50730 **CHANGE HISTORY**

50731       First released in Issue 4.

50732 **Issue 5**

50733 The following change has been made in this issue for alignment with  
50734 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 50735       • The SYNOPSIS has been changed to indicate that this function and associated data types are  
50736       now made visible by inclusion of the header `<wctype.h>` rather than `<wchar.h>`.

50737 **NAME**50738 `wcwidth` — number of column positions of a wide-character code50739 **SYNOPSIS**50740 XSI `#include <wchar.h>`50741 `int wcwidth(wchar_t wc);`

50742

50743 **DESCRIPTION**

50744 The `wcwidth()` function shall determine the number of column positions required for the wide  
50745 character `wc`. The application shall ensure that the value of `wc` is a character representable as a  
50746 **wchar\_t**, and a wide-character code corresponding to a valid character in the current locale.

50747 **RETURN VALUE**

50748 The `wcwidth()` function shall either return 0 (if `wc` is a null wide-character code), or return the  
50749 number of column positions to be occupied by the wide-character code `wc`, or return -1 (if `wc`  
50750 does not correspond to a printing wide-character code).

50751 **ERRORS**

50752 No errors are defined.

50753 **EXAMPLES**

50754 None.

50755 **APPLICATION USAGE**

50756 This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the  
50757 return value for a non-printable wide character is not specified.

50758 **RATIONALE**

50759 None.

50760 **FUTURE DIRECTIONS**

50761 None.

50762 **SEE ALSO**50763 `wcswidth()`, the Base Definitions volume of IEEE Std. 1003.1-200x, `<wchar.h>`50764 **CHANGE HISTORY**

50765 First released as a World-wide Portability Interface in Issue 4. Derived from MSE working draft.

50766 **Issue 6**

50767 The Open Group corrigenda item U021/12 has been applied. This function is marked as an  
50768 extension.

50769 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50770 **NAME**

50771 wmemchr — find a wide character in memory

50772 **SYNOPSIS**

50773 #include <wchar.h>

50774 wchar\_t \*wmemchr(const wchar\_t \*ws, wchar\_t wc, size\_t n);

50775 **DESCRIPTION**

50776 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50777 conflict between the requirements described here and the ISO C standard is unintentional. This  
50778 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

50779 The *wmemchr()* function shall locate the first occurrence of *wc* in the initial *n* wide characters of  
50780 the object pointed to by *ws*. This function is not affected by locale and all **wchar\_t** values are  
50781 treated identically. The null wide character and **wchar\_t** values not corresponding to valid  
50782 characters are not treated specially.

50783 If *n* is zero, the application shall ensure that *ws* is a valid pointer and the function behaves as if  
50784 no valid occurrence of *wc* is found.

50785 **RETURN VALUE**

50786 The *wmemchr()* function shall return a pointer to the located wide character, or a null pointer if  
50787 the wide character does not occur in the object.

50788 **ERRORS**

50789 No errors are defined.

50790 **EXAMPLES**

50791 None.

50792 **APPLICATION USAGE**

50793 None.

50794 **RATIONALE**

50795 None.

50796 **FUTURE DIRECTIONS**

50797 None.

50798 **SEE ALSO**

50799 *wmemcmp()*, *wmemcpy()*, *wmemmove()*, *wmemset()*, the Base Definitions volume of  
50800 IEEE Std. 1003.1-200x, <wchar.h>

50801 **CHANGE HISTORY**

50802 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50803 (E).

50804 **Issue 6**

50805 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50806 **NAME**

50807 wmemcmp — compare wide characters in memory

50808 **SYNOPSIS**

50809 #include &lt;wchar.h&gt;

50810 int wmemcmp(const wchar\_t \*ws1, const wchar\_t \*ws2, size\_t n);

50811 **DESCRIPTION**

50812 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50813 conflict between the requirements described here and the ISO C standard is unintentional. This  
50814 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

50815 The *wmemcmp()* function shall compare the first *n* wide characters of the object pointed to by  
50816 *ws1* to the first *n* wide characters of the object pointed to by *ws2*. This function is not affected by  
50817 locale and all **wchar\_t** values are treated identically. The null wide character and **wchar\_t** values  
50818 not corresponding to valid characters are not treated specially.

50819 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
50820 behaves as if the two objects compare equal.

50821 **RETURN VALUE**

50822 The *wmemcmp()* function shall return an integer greater than, equal to, or less than zero,  
50823 respectively, as the object pointed to by *ws1* is greater than, equal to, or less than the object  
50824 pointed to by *ws2*.

50825 **ERRORS**

50826 No errors are defined.

50827 **EXAMPLES**

50828 None.

50829 **APPLICATION USAGE**

50830 None.

50831 **RATIONALE**

50832 None.

50833 **FUTURE DIRECTIONS**

50834 None.

50835 **SEE ALSO**

50836 *wmemchr()*, *wmemcpy()*, *wmemmove()*, *wmemset()*, the Base Definitions volume of  
50837 IEEE Std. 1003.1-200x, <wchar.h>

50838 **CHANGE HISTORY**

50839 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50840 (E).

50841 **Issue 6**

50842 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50843 **NAME**

50844 wmemcpy — copy wide characters in memory

50845 **SYNOPSIS**

50846 #include &lt;wchar.h&gt;

50847 wchar\_t \*wmemcpy(wchar\_t \*restrict *ws1*, const wchar\_t \*restrict *ws2*,  
50848 size\_t *n*);50849 **DESCRIPTION**50850 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50851 conflict between the requirements described here and the ISO C standard is unintentional. This  
50852 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.50853 The *wmemcpy()* function shall copy *n* wide characters from the object pointed to by *ws2* to the  
50854 object pointed to by *ws1*. This function is not affected by locale and all **wchar\_t** values are  
50855 treated identically. The null wide character and **wchar\_t** values not corresponding to valid  
50856 characters are not treated specially.50857 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
50858 copies zero wide characters.50859 **RETURN VALUE**50860 The *wmemcpy()* function shall return the value of *ws1*.50861 **ERRORS**

50862 No errors are defined.

50863 **EXAMPLES**

50864 None.

50865 **APPLICATION USAGE**

50866 None.

50867 **RATIONALE**

50868 None.

50869 **FUTURE DIRECTIONS**

50870 None.

50871 **SEE ALSO**50872 *wmemchr()*, *wmemcmp()*, *wmemmove()*, *wmemset()*, the Base Definitions volume of  
50873 IEEE Std. 1003.1-200x, <wchar.h>50874 **CHANGE HISTORY**50875 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50876 (E).50877 **Issue 6**

50878 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50879 The *wmemcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

50880 **NAME**

50881 wmemmove — copy wide characters in memory with overlapping areas

50882 **SYNOPSIS**

50883 #include &lt;wchar.h&gt;

50884 wchar\_t \*wmemmove(wchar\_t \*ws1, const wchar\_t \*ws2, size\_t n);

50885 **DESCRIPTION**

50886 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50887 conflict between the requirements described here and the ISO C standard is unintentional. This  
50888 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

50889 The *wmemmove()* function shall copy *n* wide characters from the object pointed to by *ws2* to the  
50890 object pointed to by *ws1*. Copying takes place as if the *n* wide characters from the object pointed  
50891 to by *ws2* are first copied into a temporary array of *n* wide characters that does not overlap the  
50892 objects pointed to by *ws1* or *ws2*, and then the *n* wide characters from the temporary array are  
50893 copied into the object pointed to by *ws1*.

50894 This function is not affected by locale and all **wchar\_t** values are treated identically. The null  
50895 wide character and **wchar\_t** values not corresponding to valid characters are not treated  
50896 specially.

50897 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
50898 copies zero wide characters.

50899 **RETURN VALUE**50900 The *wmemmove()* function shall return the value of *ws1*.50901 **ERRORS**

50902 No errors are defined

50903 **EXAMPLES**

50904 None.

50905 **APPLICATION USAGE**

50906 None.

50907 **RATIONALE**

50908 None.

50909 **FUTURE DIRECTIONS**

50910 None.

50911 **SEE ALSO**

50912 *wmemchr()*, *wmemcmp()*, *wmemcpy()*, *wmemset()*, the Base Definitions volume of  
50913 IEEE Std. 1003.1-200x, <wchar.h>

50914 **CHANGE HISTORY**

50915 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50916 (E).

50917 **Issue 6**

50918 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50919 **NAME**

50920 wmemset — set wide characters in memory

50921 **SYNOPSIS**

50922 #include &lt;wchar.h&gt;

50923 wchar\_t \*wmemset(wchar\_t \*ws, wchar\_t wc, size\_t n);

50924 **DESCRIPTION**

50925 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50926 conflict between the requirements described here and the ISO C standard is unintentional. This  
50927 volume of IEEE Std. 1003.1-200x defers to the ISO C standard.

50928 The *wmemset()* function shall copy the value of *wc* into each of the first *n* wide characters of the  
50929 object pointed to by *ws*. This function is not affected by locale and all **wchar\_t** values are treated  
50930 identically. The null wide character and **wchar\_t** values not corresponding to valid characters  
50931 are not treated specially.

50932 If *n* is zero, the application shall ensure that *ws* is a valid pointer, and the function copies zero  
50933 wide characters.

50934 **RETURN VALUE**50935 The *wmemset()* functions shall return the value of *ws*.50936 **ERRORS**

50937 No errors are defined.

50938 **EXAMPLES**

50939 None.

50940 **APPLICATION USAGE**

50941 None.

50942 **RATIONALE**

50943 None.

50944 **FUTURE DIRECTIONS**

50945 None.

50946 **SEE ALSO**

50947 *wmemchr()*, *wmemcmp()*, *wmemcpy()*, *wmemmove()*, the Base Definitions volume of  
50948 IEEE Std. 1003.1-200x, <**wchar.h**>

50949 **CHANGE HISTORY**

50950 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50951 (E).

50952 **Issue 6**

50953 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.



50954 **NAME**

50955 wordexp, wordfree — perform word expansions

50956 **SYNOPSIS**

50957 #include &lt;wordexp.h&gt;

50958 int wordexp(const char \*restrict words, wordexp\_t \*restrict pwordexp,  
50959 int flags);

50960 void wordfree(wordexp\_t \*pwordexp);

50961 **DESCRIPTION**50962 The *wordexp()* function shall perform word expansions and place the list of expanded words  
50963 into *pwordexp*.50964 If the implementation supports the utilities defined in the Shell and Utilities volume of  
50965 IEEE Std. 1003.1-200x, the *wordexp()* function performs word expansions as described in the  
50966 Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.6, Word Expansions, subject to  
50967 quoting as in the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.2, Quoting, and  
50968 places the list of expanded words into the structure pointed to by *pwordexp*.50969 The *words* argument is a pointer to a string containing one or more words to be expanded. The  
50970 expansions shall be the same as would be performed by the command line interpreter if *words*  
50971 were the part of a command line representing the arguments to a utility. Therefore, the  
50972 application shall ensure that *words* does not contain an unquoted <newline> or any of the  
50973 unquoted shell special characters '|', '&', ';', '<', '>' except in the context of command  
50974 substitution as specified in the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.6.3,  
50975 Command Substitution. It also shall not contain unquoted parentheses or braces, except in the  
50976 context of command or variable substitution. If the argument *words* contains an unquoted  
50977 comment character (number sign) that is the beginning of a token, *wordexp()* shall either treat the  
50978 comment character as a regular character, or interpret it as a comment indicator and ignore the  
50979 remainder of *words*.50980 If the implementation does not support the utilities defined in the Shell and Utilities volume of  
50981 IEEE Std. 1003.1-200x, the word expansion is unspecified, but should be the same as that used by  
50982 the command language interpreter used by the *system()* and *popen()* functions.50983 The structure type **wordexp\_t** is defined in the header <**wordexp.h**> and includes at least the  
50984 following members:

50985

50986

50987

50988

50989

| Member Type | Member Name | Description                                                         |
|-------------|-------------|---------------------------------------------------------------------|
| size_t      | we_wordc    | Count of words matched by <i>words</i> .                            |
| char **     | we_wordv    | Pointer to list of expanded words.                                  |
| size_t      | we_offs     | Slots to reserve at the beginning of <i>pwordexp-&gt;we_wordv</i> . |

50990 The *wordexp()* function stores the number of generated words into *pwordexp->we\_wordc* and a  
50991 pointer to a list of pointers to words in *pwordexp->we\_wordv*. If the implementation supports the  
50992 utilities defined in the Shell and Utilities volume of IEEE Std. 1003.1-200x, each individual field  
50993 created during field splitting (see the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section  
50994 2.6.5, Field Splitting) or path name expansion (see the Shell and Utilities volume of  
50995 IEEE Std. 1003.1-200x, Section 2.6.6, Path Name Expansion) is a separate word in the *pwordexp-*  
50996 *>we\_wordv* list. The words are in order as described in the Shell and Utilities volume of  
50997 IEEE Std. 1003.1-200x, Section 2.6, Word Expansions. The first pointer after the last word pointer  
50998 shall be a null pointer. The expansion of special parameters described in the Shell and Utilities  
50999 volume of IEEE Std. 1003.1-200x, Section 2.5.2, Special Parameters is unspecified.

51000 It is the caller's responsibility to allocate the storage pointed to by *pwordexp*. The *wordexp()*  
 51001 function allocates other space as needed, including memory pointed to by *pwordexp->we\_wordv*.  
 51002 The *wordfree()* function frees any memory associated with *pwordexp* from a previous call to  
 51003 *wordexp()*.

51004 The *flags* argument is used to control the behavior of *wordexp()*. The value of *flags* is the  
 51005 bitwise-inclusive OR of zero or more of the following constants, which are defined in  
 51006 **<wordexp.h>**:

51007 **WRDE\_APPEND** Append words generated to the ones from a previous call to *wordexp()*.

51008 **WRDE\_DOOFFS** Make use of *pwordexp->we\_offs*. If this flag is set, *pwordexp->we\_offs* is used  
 51009 to specify how many null pointers to add to the beginning of *pwordexp->we\_wordv*. In other words,  
 51010 *pwordexp->we\_wordv* shall point to *pwordexp->we\_offs* null pointers, followed by *pwordexp->we\_wordc*  
 51011 word pointers, followed by a null pointer.  
 51012

51013 **WRDE\_NOCMD** If the implementation supports the utilities defined in the Shell and  
 51014 Utilities volume of IEEE Std. 1003.1-200x, fail if command substitution, as  
 51015 specified in the Shell and Utilities volume of IEEE Std. 1003.1-200x,  
 51016 Section 2.6.3, Command Substitution, is requested.

51017 **WRDE\_REUSE** The *pwordexp* argument was passed to a previous successful call to  
 51018 *wordexp()*, and has not been passed to *wordfree()*. The result shall be the  
 51019 same as if the application had called *wordfree()* and then called *wordexp()*  
 51020 without **WRDE\_REUSE**.

51021 **WRDE\_SHOWERR** Do not redirect *stderr* to **/dev/null**.

51022 **WRDE\_UNDEF** Report error on an attempt to expand an undefined shell variable.

51023 The **WRDE\_APPEND** flag can be used to append a new set of words to those generated by a  
 51024 previous call to *wordexp()*. The following rules apply to applications when two or more calls to  
 51025 *wordexp()* are made with the same value of *pwordexp* and without intervening calls to *wordfree()*:

- 51026 1. The first such call shall not set **WRDE\_APPEND**. All subsequent calls shall set it.
- 51027 2. All of the calls shall set **WRDE\_DOOFFS**, or all shall not set it.
- 51028 3. After the second and each subsequent call, *pwordexp->we\_wordv* shall point to a list  
 51029 containing the following:
  - 51030 a. Zero or more null pointers, as specified by **WRDE\_DOOFFS** and *pwordexp->we\_offs*
  - 51031 b. Pointers to the words that were in the *pwordexp->we\_wordv* list before the call, in the  
 51032 same order as before
  - 51033 c. Pointers to the new words generated by the latest call, in the specified order
- 51034 4. The count returned in *pwordexp->we\_wordc* shall be the total number of words from all of  
 51035 the calls.
- 51036 5. The application can change any of the fields after a call to *wordexp()*, but if it does it shall  
 51037 reset them to the original value before a subsequent call, using the same *pwordexp* value, to  
 51038 *wordfree()* or *wordexp()* with the **WRDE\_APPEND** or **WRDE\_REUSE** flag.

51039 If the implementation supports the utilities defined in the Shell and Utilities volume of  
 51040 IEEE Std. 1003.1-200x, and *words* contains an unquoted character—**<newline>**, **'|'**, **'&'**, **';'**,  
 51041 **'<'**, **'>'**, **'('**, **')'**, **'{'**, **'}'**—in an inappropriate context, *wordexp()* shall fail, and the number  
 51042 of expanded words shall be 0.

51043 Unless WRDE\_SHOWERR is set in *flags*, *wordexp()* shall redirect *stderr* to */dev/null* for any  
 51044 utilities executed as a result of command substitution while expanding *words*. If  
 51045 WRDE\_SHOWERR is set, *wordexp()* may write messages to *stderr* if syntax errors are detected  
 51046 while expanding *words*.

51047 The application shall ensure that if WRDE\_DOOFFS is set, then *pwordexp->we\_offs* has the same  
 51048 value for each *wordexp()* call and *wordfree()* call using a given *pwordexp*.

51049 The following constants are defined as error return values:

51050 WRDE\_BADCHAR One of the unquoted characters—<newline>, ' | ', '&', ';', '<', '>',  
 51051 ' ( ', ' ) ', ' { ', ' } '—appears in *words* in an inappropriate context.

51052 WRDE\_BADVAL Reference to undefined shell variable when WRDE\_UNDEF is set in *flags*.

51053 WRDE\_CMDSUB Command substitution requested when WRDE\_NOCMD was set in *flags*.

51054 WRDE\_NOSPACE Attempt to allocate memory failed.

51055 WRDE\_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated  
 51056 string.

#### 51057 RETURN VALUE

51058 Upon successful completion, *wordexp()* shall return 0. Otherwise, a non-zero value, as described  
 51059 in <*wordexp.h*>, shall be returned to indicate an error. If *wordexp()* returns the value  
 51060 WRDE\_NOSPACE, then *pwordexp->we\_wordc* and *pwordexp->we\_wordv* shall be updated to  
 51061 reflect any words that were successfully expanded. In other cases, they shall not be modified.

51062 The *wordfree()* function shall return no value.

#### 51063 ERRORS

51064 No errors are defined.

#### 51065 EXAMPLES

51066 None.

#### 51067 APPLICATION USAGE

51068 The *wordexp()* function is intended to be used by an application that wants to do all of the shell's  
 51069 expansions on a word or words obtained from a user. For example, if the application prompts  
 51070 for a file name (or list of file names) and then uses *wordexp()* to process the input, the user could  
 51071 respond with anything that would be valid as input to the shell.

51072 The WRDE\_NOCMD flag is provided for applications that, for security or other reasons, want to  
 51073 prevent a user from executing shell commands. Disallowing unquoted shell special characters  
 51074 also prevents unwanted side effects, such as executing a command or writing a file.

#### 51075 RATIONALE

51076 This function was included as an alternative to *glob()*. There had been continuing controversy  
 51077 over exactly what features should be included in *glob()*. It is hoped that by providing *wordexp()*  
 51078 (which provides all of the shell word expansions, but which may be slow to execute) and *glob()*  
 51079 (which is faster, but which only performs path name expansion, without tilde or parameter  
 51080 expansion) this will satisfy the majority of applications.

51081 While *wordexp()* could be implemented entirely as a library routine, it is expected that most  
 51082 implementations run a shell in a subprocess to do the expansion.

51083 Two different approaches have been proposed for how the required information might be  
 51084 presented to the shell and the results returned. They are presented here as examples.

51085 One proposal is to extend the *echo* utility by adding a *-q* option. This option would cause *echo* to  
 51086 add a backslash before each backslash and <blank> character that occurs within an argument.

51087 The *wordexp()* function could then invoke the shell as follows:

```
51088 (void) strcpy(buffer, "echo -q");
51089 (void) strcat(buffer, words);
51090 if ((flags & WRDE_SHOWERR) == 0)
51091 (void) strcat(buffer, "2>/dev/null");
51092 f = popen(buffer, "r");
```

51093 The *wordexp()* function would read the resulting output, remove unquoted backslashes, and  
 51094 break into words at unquoted <blank>s. If the WRDE\_NOCMD flag was set, *wordexp()* would  
 51095 have to scan *words* before starting the subshell to make sure that there would be no command  
 51096 substitution. In any case, it would have to scan *words* for unquoted special characters.

51097 Another proposal is to add the following options to *sh*:

51098 **-w wordlist**

51099 This option provides a wordlist expansion service to applications. The words in *wordlist*  
 51100 shall be expanded and the following written to standard output:

- 51101 1. The count of the number of words after expansion, in decimal, followed by a null byte
- 51102 2. The number of bytes needed to represent the expanded words (not including null  
 51103 separators), in decimal, followed by a null byte
- 51104 3. The expanded words, each terminated by a null byte

51105 If an error is encountered during word expansion, *sh* exits with a non-zero status after  
 51106 writing the former to report any words successfully expanded

51107 **-P** Run in “protected” mode. If specified with the **-w** option, no command substitution shall  
 51108 be performed.

51109 With these options, *wordexp()* could be implemented fairly simply by creating a subprocess  
 51110 using *fork()* and executing *sh* using the line:

```
51111 execl(<shell path>, "sh", "-P", "-w", words, (char *)0);
```

51112 after directing standard error to **/dev/null**.

51113 It seemed objectionable for a library routine to write messages to standard error, unless  
 51114 explicitly requested, so *wordexp()* is required to redirect standard error to **/dev/null** to ensure  
 51115 that no messages are generated, even for commands executed for command substitution. The  
 51116 WRDE\_SHOWERR flag can be specified to request that error messages be written.

51117 The WRDE\_REUSE flag allows the implementation to avoid the expense of freeing and  
 51118 reallocating memory, if that is possible. A minimal implementation can call *wordfree()* when  
 51119 WRDE\_REUSE is set.

#### 51120 FUTURE DIRECTIONS

51121 None.

#### 51122 SEE ALSO

51123 *fnmatch()*, *glob()*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<wordexp.h>**, the Shell  
 51124 and Utilities volume of IEEE Std. 1003.1-200x

#### 51125 CHANGE HISTORY

51126 First released in Issue 4. Derived from the ISO POSIX-2 standard.

51127 **Issue 5**

51128 Moved from POSIX2 C-language Binding to BASE.

51129 **Issue 6**

51130 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

51131 The **restrict** keyword is added to the *wordexp()* prototype for alignment with the

51132 ISO/IEC 9899:1999 standard.

51133 **NAME**

51134        `wprintf` — print formatted wide-character output

51135 **SYNOPSIS**

51136        `#include <stdio.h>`

51137        `#include <wchar.h>`

51138        `int wprintf(const wchar_t *format, ...);`

51139 **DESCRIPTION**

51140        Refer to *fwprintf()*.

## 51141 NAME

51142 pwrite, write, writev — write on a file

## 51143 SYNOPSIS

51144 #include &lt;unistd.h&gt;

51145 XSI ssize\_t pwrite(int *fildev*, const void \**buf*, size\_t *nbyte*,  
51146 off\_t *offset*);51147 ssize\_t write(int *fildev*, const void \**buf*, size\_t *nbyte*);

51148 XSI #include &lt;sys/uio.h&gt;

51149 ssize\_t writev(int *fildev*, const struct iovec \**iov*, int *iovcnt*);

51150

## 51151 DESCRIPTION

51152 XSI The *pwrite()* function performs the same action as *write()*, except that it writes into a given  
51153 position without changing the file pointer. The first three arguments to *pwrite()* are the same as  
51154 *write()* with the addition of a fourth argument *offset* for the desired position inside the file.51155 **Notes to Reviewers**51156 *This section with side shading will not appear in the final copy. - Ed.*51157 D3, XSH, ERN 676 says that *pwrite()* (and *pread()*) need to be limited to seekable devices or have  
51158 an explicit statement that the *offset* argument is ignored. This item is still open.51159 The *write()* function shall attempt to write *nbyte* bytes from the buffer pointed to by *buf* to the  
51160 file associated with the open file descriptor, *fildev*.51161 If *nbyte* is zero and the file is a regular file, the *write()* function may detect and return errors as  
51162 described below. In the absence of errors, or if error detection is not performed, the *write()*  
51163 function shall return zero and have no other results. If *nbyte* is zero and the file is not a regular  
51164 file, the results are unspecified.51165 On a regular file or other file capable of seeking, the actual writing of data proceeds from the  
51166 position in the file indicated by the file offset associated with *fildev*. Before successful return  
51167 from *write()*, the file offset is incremented by the number of bytes actually written. On a regular  
51168 file, if this incremented file offset is greater than the length of the file, the length of the file shall  
51169 be set to this file offset.51170 On a file not capable of seeking, writing always takes place starting at the current position. The  
51171 value of a file offset associated with such a device is undefined.51172 If the O\_APPEND flag of the file status flags is set, the file offset shall be set to the end of the file  
51173 prior to each write and no intervening file modification operation shall occur between changing  
51174 the file offset and the write operation.51175 XSI If a *write()* requests that more bytes be written than there is room for (for example, the process'  
51176 file size limit or the physical end of a medium), only as many bytes as there is room for shall be  
51177 written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A  
51178 write of 512 bytes shall return 20. The next write of a non-zero number of bytes shall give a  
51179 XSI failure return (except as noted below) and the implementation shall generate a SIGXFSZ signal  
51180 for the thread.51181 If *write()* is interrupted by a signal before it writes any data, it shall return -1 with *errno* set to  
51182 [EINTR].51183 If *write()* is interrupted by a signal after it successfully writes some data, it shall return the  
51184 number of bytes written.

- 51185 If the value of *nbyte* is greater than {SSIZE\_MAX}, the result is implementation-defined.
- 51186 After a *write()* to a regular file has successfully returned:
- 51187 • Any successful *read()* from each byte position in the file that was modified by that write shall
  - 51188 return the data specified by the *write()* for that position until such byte positions are again
  - 51189 modified.
  - 51190 • Any subsequent successful *write()* to the same byte position in the file shall overwrite that
  - 51191 file data.
- 51192 Write requests to a pipe or FIFO shall be handled the same as a regular file with the following
- 51193 exceptions:
- 51194 • There is no file offset associated with a pipe, hence each write request shall append to the
  - 51195 end of the pipe.
  - 51196 • Write requests of {PIPE\_BUF} bytes or less shall not be interleaved with data from other
  - 51197 processes doing writes on the same pipe. Writes of greater than {PIPE\_BUF} bytes may have
  - 51198 data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the
  - 51199 O\_NONBLOCK flag of the file status flags is set.
  - 51200 • If the O\_NONBLOCK flag is clear, a write request may cause the thread to block, but on
  - 51201 normal completion it shall return *nbyte*.
  - 51202 • If the O\_NONBLOCK flag is set, *write()* requests shall be handled differently, in the
  - 51203 following ways:
    - 51204 — The *write()* function shall not block the thread.
    - 51205 — A write request for {PIPE\_BUF} or fewer bytes shall have the following effect: if there is
    - 51206 sufficient space available in the pipe, *write()* shall transfer all the data and return the
    - 51207 number of bytes requested. Otherwise, *write()* shall transfer no data and return  $-1$  with
    - 51208 *errno* set to [EAGAIN].
    - 51209 — A write request for more than {PIPE\_BUF} bytes shall cause one of the following:
      - 51210 — When at least one byte can be written, transfer what it can and return the number of
      - 51211 bytes written. When all data previously written to the pipe is read, it shall transfer at
      - 51212 least {PIPE\_BUF} bytes.
      - 51213 — When no data can be written, transfer no data, and return  $-1$  with *errno* set to
      - 51214 [EAGAIN].
- 51215 When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-
- 51216 blocking writes and cannot accept the data immediately:
- 51217 • If the O\_NONBLOCK flag is clear, *write()* shall block the calling thread until the data can be
  - 51218 accepted.
  - 51219 • If the O\_NONBLOCK flag is set, *write()* shall not block the thread. If some data can be
  - 51220 written without blocking the thread, *write()* shall write what it can and return the number of
  - 51221 bytes written. Otherwise, it shall return  $-1$  and set *errno* to [EAGAIN].
- 51222 Upon successful completion, where *nbyte* is greater than 0, *write()* shall mark for update the
- 51223 *st\_ctime* and *st\_mtime* fields of the file, and if the file is a regular file, the S\_ISUID and S\_ISGID
- 51224 bits of the file mode may be cleared.
- 51225 XSR If *fildev* refers to a STREAM, the operation of *write()* shall be determined by the values of the
- 51226 minimum and maximum *nbyte* range (packet size) accepted by the STREAM. These values are
- 51227 determined by the topmost STREAM module. If *nbyte* falls within the packet size range, *nbyte*



51228 bytes shall be written. If *nbyte* does not fall within the range and the minimum packet size value  
 51229 is 0, *write()* shall break the buffer into maximum packet size segments prior to sending the data  
 51230 downstream (the last segment may contain less than the maximum packet size). If *nbyte* does not  
 51231 fall within the range and the minimum value is non-zero, *write()* shall fail with *errno* set to  
 51232 [ERANGE]. Writing a zero-length buffer (*nbyte* is 0) to a STREAMS device sends 0 bytes with 0  
 51233 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no  
 51234 message and 0 is returned. The process may issue `I_SWROPT ioctl()` to enable zero-length  
 51235 messages to be sent across the pipe or FIFO.

51236 When writing to a STREAM, data messages are created with a priority band of 0. When writing  
 51237 to a STREAM that is not a pipe or FIFO:

- If `O_NONBLOCK` is clear, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), *write()* shall block until data can be accepted.

- If `O_NONBLOCK` is set and the STREAM cannot accept data, *write()* shall return `-1` and set *errno* to [EAGAIN].

- If `O_NONBLOCK` is set and part of the buffer has been written while a condition in which the STREAM cannot accept additional data occurs, *write()* shall terminate and return the number of bytes written.

51245 In addition, *write()* and *writew()* shall fail if the STREAM head has processed an asynchronous  
 51246 error before the call. In this case, the value of *errno* does not reflect the result of *write()* or  
 51247 *writew()*, but reflects the prior error.

51248 XSI The *writew()* function is equivalent to *write()*, but gathers the output data from the *iovcnt* buffers  
 51249 specified by the members of the *iov* array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*-1]. *iovcnt* is valid if  
 51250 greater than 0 and less than or equal to {IOV\_MAX}, defined in <limits.h>.

51251 Each *iovec* entry specifies the base address and length of an area in memory from which data  
 51252 should be written. The *writew()* function shall always write a complete area before proceeding to  
 51253 the next.

51254 If *fildev* refers to a regular file and all of the *iov\_len* members in the array pointed to by *iov* are 0,  
 51255 *writew()* shall return 0 and have no other effect. For other file types, the behavior is unspecified.

51256 If the sum of the *iov\_len* values is greater than {SSIZE\_MAX}, the operation fails and no data is  
 51257 transferred.

51258 SIO If the `O_DSYNC` bit has been set, write I/O operations on the file descriptor complete as defined  
 51259 by synchronized I/O data integrity completion.

51260 If the `O_SYNC` bit has been set, write I/O operations on the file descriptor complete as defined  
 51261 by synchronized I/O file integrity completion.

51262 SHM If *fildev* refers to a shared memory object, the result of the *write()* function is unspecified.

51263 TYM If *fildev* refers to a typed memory object, the result of the *write()* function is unspecified.

51264 For regular files, no data transfer shall occur past the offset maximum established in the open  
 51265 file description associated with *fildev*.

51266 If *fildev* refers to a socket, *write()* is equivalent to *send()* with no flags set.

#### 51267 RETURN VALUE

51268 XSI Upon successful completion, *write()* and *pwrite()* shall return the number of bytes actually  
 51269 written to the file associated with *fildev*. This number shall never be greater than *nbyte*.  
 51270 Otherwise, `-1` shall be returned and *errno* set to indicate the error.

51271 XSI Upon successful completion, `writew()` shall return the number of bytes actually written.  
 51272 Otherwise, it shall return a value of `-1`, the file-pointer shall remain unchanged, and `errno` shall  
 51273 be set to indicate an error.

51274 **ERRORS**

51275 XSI The `write()`, `pwrite()`, and `writew()` functions shall fail if:

51276 [EAGAIN] The `O_NONBLOCK` flag is set for the file descriptor and the thread would be  
 51277 delayed in the `write()` operation.

51278 [EBADF] The `fildev` argument is not a valid file descriptor open for writing.

51279 [EFBIG] An attempt was made to write a file that exceeds the implementation-defined  
 51280 XSI maximum file size or the process' file size limit.

51281 [EFBIG] The file is a regular file, `nbyte` is greater than 0, and the starting position is  
 51282 greater than or equal to the offset maximum established in the open file  
 51283 description associated with `fildev`.

51284 [EINTR] The write operation was terminated due to the receipt of a signal, and no data  
 51285 was transferred.

51286 [EIO] The process is a member of a background process group attempting to write  
 51287 to its controlling terminal, `TOSTOP` is set, the process is neither ignoring nor  
 51288 blocking `SIGTTOU`, and the process group of the process is orphaned. This  
 51289 error may also be returned under implementation-defined conditions.

51290 [ENOSPC] There was no free space remaining on the device containing the file.

51291 [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
 51292 any process, or that only has one end open. A `SIGPIPE` signal shall also be sent  
 51293 to the thread.

51294 XSR [ERANGE] The transfer request size was outside the range supported by the `STREAMS`  
 51295 file associated with `fildev`.

51296 MAN The `write()` function shall fail if:

51297 [EAGAIN] or [EWOULDBLOCK]

51298 The file descriptor is for a connection-mode socket, is marked  
 51299 `O_NONBLOCK`, and write would block.

51300 [ECONNRESET] A write was attempted on a connection-mode socket that is not connected.

51301 [EPIPE] A write was attempted on a connection-mode socket that is shut down for  
 51302 writing, or is no longer connected. In the latter case, if the socket is of type  
 51303 `SOCK_STREAM`, the `SIGPIPE` signal is generated to the calling process.  
 51304

51305 The `writew()` function shall fail if:

51306 XSI [EINVAL] The sum of the `iov_len` values in the `iov` array would overflow an `ssize_t`.

51307 XSI The `write()`, `pwrite()`, and `writew()` functions may fail if:

51308 XSR [EINVAL] The `STREAM` or multiplexer referenced by `fildev` is linked (directly or  
 51309 indirectly) downstream from a multiplexer.

51310 MAN [EIO] A physical I/O error has occurred.

51311 MAN [ENOBUFS] Insufficient resources were available in the system to perform the operation.

- 51312 [ENXIO] A request was made of a nonexistent device, or the request was outside the  
51313 capabilities of the device.
- 51314 XSR [ENXIO] A hangup occurred on the STREAM being written to.
- 51315 XSR A write to a STREAMS file may fail if an error message has been received at the STREAM head.  
51316 In this case, *errno* is set to the value included in the error message.
- 51317 MAN The *write()* function may fail if:
- 51318 [EACCES] A write was attempted on a connection-mode socket and the calling process  
51319 does not have appropriate privileges.
- 51320 [ENETDOWN] A write was attempted on a connection-mode socket and the local network  
51321 interface used to reach the destination is down.
- 51322 [ENETUNREACH] A write was attempted on a connection-mode socket and no route to the  
51323 network is present.  
51324
- 51325 The *writew()* function may fail and set *errno* to:
- 51326 XSI [EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV\_MAX}.
- 51327 XSI The *pwrite()* function shall fail and the file pointer remain unchanged if:
- 51328 XSI [EINVAL] The *offset* argument is invalid. The value is negative.
- 51329 XSI [ESPIPE] *fildev* is associated with a pipe or FIFO.

51330 **EXAMPLES**51331 **Writing from a Buffer**

51332 The following example writes data from the buffer pointed to by *buf* to the file associated with  
51333 the file descriptor *fd*.

```
51334 #include <sys/types.h>
51335 #include <string.h>
51336 ...
51337 char buf[20];
51338 size_t nbytes;
51339 ssize_t bytes_written;
51340 int fd;
51341 ...
51342 strcpy(buf, "This is a test\n");
51343 nbytes = strlen(buf);
51344 bytes_written = write(fd, buf, nbytes);
51345 ...
```

51346 **Writing Data from an Array**

51347 The following example writes data from the buffers specified by members of the *iov* array to the  
51348 file associated with the file descriptor *fd*.

```
51349 #include <sys/types.h>
51350 #include <sys/uio.h>
51351 #include <unistd.h>
51352 ...
51353 ssize_t bytes_written;
51354 int fd;
51355 char *buf0 = "short string\n";
51356 char *buf1 = "This is a longer string\n";
51357 char *buf2 = "This is the longest string in this example\n";
51358 int iovcnt;
51359 struct iovec iov[3];

51360 iov[0].iov_base = buf0;
51361 iov[0].iov_len = strlen(buf0);
51362 iov[1].iov_base = buf1;
51363 iov[1].iov_len = strlen(buf1);
51364 iov[2].iov_base = buf2;
51365 iov[2].iov_len = strlen(buf2);
51366 ...
51367 iovcnt = sizeof(iov) / sizeof(struct iovec);

51368 bytes_written = writev(fd, iov, iovcnt);
51369 ...
```

51370 **APPLICATION USAGE**

51371 None.

51372 **RATIONALE**

51373 See also the RATIONALE section in *read()*.

51374 An attempt to write to a pipe or FIFO has several major characteristics:

51375 • *Atomic/non-atomic*: A write is atomic if the whole amount written in one operation is not  
51376 interleaved with data from any other process. This is useful when there are multiple writers  
51377 sending data to a single reader. Applications need to know how large a write request can be  
51378 expected to be performed atomically. This maximum is called {PIPE\_BUF}. This volume of  
51379 IEEE Std. 1003.1-200x does not say whether write requests for more than {PIPE\_BUF} bytes  
51380 are atomic, but requires that writes of {PIPE\_BUF} or fewer bytes shall be atomic.

51381 • *Blocking/immediate*: Blocking is only possible with O\_NONBLOCK clear. If there is enough  
51382 space for all the data requested to be written immediately, the implementation should do so.  
51383 Otherwise, the process may block; that is, pause until enough space is available for writing.  
51384 The effective size of a pipe or FIFO (the maximum amount that can be written in one  
51385 operation without blocking) may vary dynamically, depending on the implementation, so it  
51386 is not possible to specify a fixed value for it.

51387 • *Complete/partial/deferred*: A write request:

```
51388 int fildes;
51389 size_t nbyte;
51390 ssize_t ret;
51391 char *buf;
```

51392           ret = write(fildev, buf, nbyte);

51393           may return:

51394           complete    ret=nbyte

51395           partial     ret<nbyte

51396                       This shall never happen if *nbyte*≤{PIPE\_BUF}. If it does happen (with  
51397                       *nbyte*>{PIPE\_BUF}), this volume of IEEE Std. 1003.1-200x does not guarantee  
51398                       atomicity, even if *ret*≤{PIPE\_BUF}, because atomicity is guaranteed according  
51399                       to the amount *requested*, not the amount *written*.

51400           deferred:   ret=-1, *errno*=[EAGAIN]

51401                       This error indicates that a later request may succeed. It does not indicate that it  
51402                       *shall* succeed, even if *nbyte*≤{PIPE\_BUF}, because if no process reads from the  
51403                       pipe or FIFO, the write never succeeds. An application could usefully count the  
51404                       number of times [EAGAIN] is caused by a particular value of  
51405                       *nbyte*>{PIPE\_BUF} and perhaps do later writes with a smaller value, on the  
51406                       assumption that the effective size of the pipe may have decreased.

51407           Partial and deferred writes are only possible with O\_NONBLOCK set.

51408           The relations of these properties are shown in the following tables:

51409

51410

51411

51412

51413

51414

| Write to a Pipe or FIFO with O_NONBLOCK clear |                                 |                                 |                                  |
|-----------------------------------------------|---------------------------------|---------------------------------|----------------------------------|
| Immediately Writable:                         | None                            | Some                            | <i>nbyte</i>                     |
| <i>nbyte</i> ≤{PIPE_BUF}                      | Atomic blocking<br><i>nbyte</i> | Atomic blocking<br><i>nbyte</i> | Atomic immediate<br><i>nbyte</i> |
| <i>nbyte</i> >{PIPE_BUF}                      | Blocking <i>nbyte</i>           | Blocking <i>nbyte</i>           | Blocking <i>nbyte</i>            |

51415           If the O\_NONBLOCK flag is clear, a write request shall block if the amount writable  
51416           immediately is less than that requested. If the flag is set (by *fcntl()*), a write request shall never  
51417           block.

51418

51419

51420

51421

51422

51423

| Write to a Pipe or FIFO with O_NONBLOCK set |              |                                   |                                   |
|---------------------------------------------|--------------|-----------------------------------|-----------------------------------|
| Immediately Writable:                       | None         | Some                              | <i>nbyte</i>                      |
| <i>nbyte</i> ≤{PIPE_BUF}                    | -1, [EAGAIN] | -1, [EAGAIN]                      | Atomic <i>nbyte</i>               |
| <i>nbyte</i> >{PIPE_BUF}                    | -1, [EAGAIN] | < <i>nbyte</i> or -1,<br>[EAGAIN] | ≤ <i>nbyte</i> or -1,<br>[EAGAIN] |

51424           There is no exception regarding partial writes when O\_NONBLOCK is set. With the exception  
51425           of writing to an empty pipe, this volume of IEEE Std. 1003.1-200x does not specify exactly when  
51426           a partial write is performed since that would require specifying internal details of the  
51427           implementation. Every application should be prepared to handle partial writes when  
51428           O\_NONBLOCK is set and the requested amount is greater than {PIPE\_BUF}, just as every  
51429           application should be prepared to handle partial writes on other kinds of file descriptors.

51430           The intent of forcing writing at least one byte if any can be written is to assure that each write  
51431           makes progress if there is any room in the pipe. If the pipe is empty, {PIPE\_BUF} bytes must be  
51432           written; if not, at least some progress must have been made.

51433           Where this volume of IEEE Std. 1003.1-200x requires -1 to be returned and *errno* set to  
51434           [EAGAIN], most historical implementations return zero (with the O\_NDELAY flag set, which is  
51435           the historical predecessor of O\_NONBLOCK, but is not itself in this volume of

- 51436 IEEE Std. 1003.1-200x). The error indications in this volume of IEEE Std. 1003.1-200x were chosen  
 51437 so that an application can distinguish these cases from end-of-file. While *write()* cannot receive  
 51438 an indication of end-of-file, *read()* can, and the two functions have similar return values. Also,  
 51439 some existing systems (for example, Eighth Edition) permit a write of zero bytes to mean that  
 51440 the reader should get an end-of-file indication; for those systems, a return value of zero from  
 51441 *write()* indicates a successful write of an end-of-file indication.
- 51442 Implementations are allowed, but not required, to perform error checking for *write()* requests of  
 51443 zero bytes.
- 51444 The concept of a {PIPE\_MAX} limit (indicating the maximum number of bytes that can be  
 51445 written to a pipe in a single operation) was considered, but rejected, because this concept would  
 51446 unnecessarily limit application writing.
- 51447 See also the discussion of O\_NONBLOCK in *read()*.
- 51448 Writes can be serialized with respect to other reads and writes. If a *read()* of file data can be  
 51449 proven (by any means) to occur after a *write()* of the data, it must reflect that *write()*, even if the  
 51450 calls are made by different processes. A similar requirement applies to multiple write operations  
 51451 to the same file position. This is needed to guarantee the propagation of data from *write()* calls  
 51452 to subsequent *read()* calls. This requirement is particularly significant for networked file  
 51453 systems, where some caching schemes violate these semantics.
- 51454 Note that this is specified in terms of *read()* and *write()*. Additional calls, such as the common  
 51455 *readv()* and *writew()*, would want to obey these semantics. A new “high-performance” write  
 51456 analog that did not follow these serialization requirements would also be permitted by this  
 51457 wording. This volume of IEEE Std. 1003.1-200x is also silent about any effects of application-  
 51458 level caching (such as that done by *stdio*).
- 51459 This volume of IEEE Std. 1003.1-200x does not specify the value of the file offset after an error is  
 51460 returned; there are too many cases. For programming errors, such as [EBADF], the concept is  
 51461 meaningless since no file is involved. For errors that are detected immediately, such as  
 51462 [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however,  
 51463 an updated value would be very useful and is the behavior of many implementations.
- 51464 This volume of IEEE Std. 1003.1-200x does not specify behavior of concurrent writes to a file  
 51465 from multiple processes. Applications should use some form of concurrency control.
- 51466 **FUTURE DIRECTIONS**
- 51467 None.
- 51468 **SEE ALSO**
- 51469 *chmod()*, *creat()*, *dup()*, *fcntl()*, *getrlimit()*, *lseek()*, *open()*, *pipe()*, *ulimit()*, the Base Definitions  
 51470 volume of IEEE Std. 1003.1-200x, <limits.h>, <stropts.h>, <sys/uio.h>, <unistd.h>
- 51471 **CHANGE HISTORY**
- 51472 First released in Issue 1. Derived from Issue 1 of the SVID.
- 51473 **Issue 4**
- 51474 The <unistd.h> header is added to the SYNOPSIS section.
- 51475 Reference to *ulimit* in the DESCRIPTION is marked as an extension.
- 51476 Reference to the process' file size limit and the *ulimit()* function are marked as extensions in the  
 51477 description of the [EFBIG] error.
- 51478 The [ENXIO] error is marked as an extension.
- 51479 The APPLICATION USAGE section is removed.

51480 The description of [EINTR] is amended.

51481 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 51482 • The type of the argument *buf* is changed from **char\*** to **const void\***, and the type of the
- 51483 argument *nbyte* is changed from **unsigned** to **size\_t**.
- 51484 • The DESCRIPTION is changed as follows:
  - 51485 — Writing at end-of-file is atomic.
  - 51486 — {SSIZE\_MAX} is now used to determine the maximum value of *nbyte*.
  - 51487 — The consequences of activities after a call to the *write()* function are added.
  - 51488 — To improve clarity, the text describing operations on pipes or FIFOs when
  - 51489 O\_NONBLOCK is set is restructured.

#### 51490 **Issue 4, Version 2**

51491 The following changes are incorporated for X/OPEN UNIX conformance:

- 51492 • The *writew()* function is added to the SYNOPSIS.
- 51493 • The DESCRIPTION is updated to describe the writing of data to STREAMS files, an
- 51494 operational description of the *writew()* function is included, and a statement is added
- 51495 indicating that SIGXFSZ is generated if an attempted write operation would cause the
- 51496 maximum file size to be exceeded.
- 51497 • The RETURN VALUE section is updated to describe values returned by the *writew()* function.
- 51498 • The ERRORS section has been restructured to describe errors that apply to both *write()* and
- 51499 *writew()* apart from those that apply to *writew()* specifically. The [EIO], [ERANGE], and
- 51500 [EINVAL] errors are also added.

#### 51501 **Issue 5**

51502 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX

51503 Threads Extension.

51504 Large File Summit extensions are added.

51505 The *pwrite()* function is added.

#### 51506 **Issue 6**

51507 The DESCRIPTION states that the *write()* function does not block the thread. Previously this

51508 said “process” rather than “thread”.

51509 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are

51510 marked as part of the XSI STREAMS Option Group.

51511 The following new requirements on POSIX implementations derive from alignment with the

51512 Single UNIX Specification:

- 51513 • The DESCRIPTION now states that if *write()* is interrupted by a signal after it has
- 51514 successfully written some data, it returns the number of bytes written. In earlier versions of
- 51515 this volume of IEEE Std. 1003.1-200x, it was optional whether *write()* returned the number of
- 51516 bytes written, or whether it returned  $-1$  with *errno* set to [EINTR]. This is a FIPS requirement.
- 51517 • The following changes are made to support large files:
  - 51518 — For regular files, no data transfer occurs past the offset maximum established in the open
  - 51519 file description associated with the *files*.

- 51520 — A second [EFBIG] error condition is added.
- 51521 • The [EIO] error condition is added.
- 51522 • The [EPIPE] error condition is added for when a pipe has only one end open.
- 51523 • The [ENXIO] optional error condition is added.
- 51524 Text referring to sockets is added to the DESCRIPTION.
- 51525 The following changes were made to align with the IEEE P1003.1a draft standard:
- 51526 • The effect of reading zero bytes is clarified.
- 51527 The DESCRIPTION is updated for alignment with IEEE Std. 1003.1j-2000 by specifying that  
51528 *write()* results are unspecified for typed memory objects.
- 51529 The following error conditions are added for operations on sockets: [EAGAIN],  
51530 [EWOULDBLOCK], [ECONNRESET], [ENOTCONN], and [EPIPE].
- 51531 The [EIO] error is changed to “may fail”.
- 51532 The [ENOBUFS] error is added for sockets.
- 51533 The following error conditions are added for operations on sockets: [EACCES], [ENETDOWN],  
51534 and [ENETUNREACH].



51535 **NAME**

51536           wscanf — convert formatted wide-character input

51537 **SYNOPSIS**

51538           #include &lt;stdio.h&gt;

51539           #include &lt;wchar.h&gt;

51540           int wscanf(const wchar\_t \**format*, ... );51541 **DESCRIPTION**51542           Refer to *fwscanf()*.

51543 **NAME**

51544 y0, y1, yn — Bessel functions of the second kind

51545 **SYNOPSIS**

```
51546 xSI #include <math.h>
51547 double y0(double x);
51548 double y1(double x);
51549 double yn(int n, double x);
51550
```

51551 **DESCRIPTION**

51552 The *y0()*, *y1()*, and *yn()* functions shall compute Bessel functions of *x* of the second kind of  
 51553 orders 0, 1, and *n* respectively. The application shall ensure that the value of *x* is positive.

51554 An application wishing to check for error situations should set *errno* to 0 before calling *y0()*,  
 51555 *y1()*, or *yn()*. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

51556 **RETURN VALUE**

51557 Upon successful completion, *y0()*, *y1()*, and *yn()* shall return the relevant Bessel value of *x* of  
 51558 the second kind.

51559 If *x* is NaN, NaN shall be returned and *errno* may be set to [EDOM].

51560 If the *x* argument to *y0()*, *y1()*, or *yn()* is negative, *-HUGE\_VAL* or NaN shall be returned, and  
 51561 *errno* may be set to [EDOM].

51562 If *x* is 0.0, *-HUGE\_VAL* shall be returned and *errno* may be set to [ERANGE] or [EDOM].

51563 If the correct result would cause underflow, 0.0 shall be returned and *errno* may be set to  
 51564 [ERANGE].

51565 If the correct result would cause overflow, *-HUGE\_VAL* or 0.0 shall be returned and *errno* may  
 51566 be set to [ERANGE].

51567 **ERRORS**

51568 The *y0()*, *y1()*, and *yn()* functions may fail if:

51569 [EDOM]           The value of *x* is negative or NaN. |

51570 [ERANGE]         The value of *x* is too large in magnitude, or *x* is 0.0, or the correct result would |  
 51571 cause overflow or underflow.

51572 No other errors shall occur.

51573 **EXAMPLES**

51574 None.

51575 **APPLICATION USAGE**

51576 None.

51577 **RATIONALE**

51578 None.

51579 **FUTURE DIRECTIONS**

51580 None.

51581 **SEE ALSO**

51582 *isnan()*, *j0()*, the Base Definitions volume of IEEE Std. 1003.1-200x, <math.h> |

51583 **CHANGE HISTORY**

51584 First released in Issue 1. Derived from Issue 1 of the SVID.

51585 **Issue 4**

51586 References to *matherr()* are removed.

51587 The RETURN VALUE and ERRORS sections are substantially rewritten to rationalize error  
51588 handling in the mathematics functions.

51589 **Issue 5**

51590 The DESCRIPTION is updated to indicate how an application should check for an error. This  
51591 text was previously published in the APPLICATION USAGE section.

51592 **Issue 6**

51593 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.



# Index

1

|    |                                                    |            |                                             |               |
|----|----------------------------------------------------|------------|---------------------------------------------|---------------|
| 2  | <code>±0</code> .....                              | 871, 876   | <code>_PC_VDISABLE</code> .....             | 892           |
| 3  | <code>_CS_PATH</code> .....                        | 714        | <code>_POSIX2 constants</code>              |               |
| 4  | <code>_CS_XBS5_ILP32_OFF32_CFLAGS</code> .....     | 714        | in <code>sysconf</code> .....               | 1980          |
| 5  | <code>_CS_XBS5_ILP32_OFF32_LDFLAGS</code> .....    | 714        | <code>_POSIX2_CHAR_TERM</code> .....        | 1982          |
| 6  | <code>_CS_XBS5_ILP32_OFF32_LIBS</code> .....       | 714        | <code>_POSIX2_C_BIND</code> .....           | 1982          |
| 7  | <code>_CS_XBS5_ILP32_OFF32_LINTFLAGS</code> .....  | 714        | <code>_POSIX2_C_DEV</code> .....            | 1982          |
| 8  | <code>_CS_XBS5_ILP32_OFFBIG_CFLAGS</code> .....    | 714        | <code>_POSIX2_C_VERSION</code> .....        | 1982          |
| 9  | <code>_CS_XBS5_ILP32_OFFBIG_LDFLAGS</code> .....   | 714        | <code>_POSIX2_FORT_DEV</code> .....         | 1982          |
| 10 | <code>_CS_XBS5_ILP32_OFFBIG_LIBS</code> .....      | 714        | <code>_POSIX2_FORT_RUN</code> .....         | 1982          |
| 11 | <code>_CS_XBS5_ILP32_OFFBIG_LINTFLAGS</code> ..... | 714        | <code>_POSIX2_LOCALEDEF</code> .....        | 1982          |
| 12 | <code>_CS_XBS5_LP64_OFF64_CFLAGS</code> .....      | 714        | <code>_POSIX2_PBS</code> .....              | 1982          |
| 13 | <code>_CS_XBS5_LP64_OFF64_LDFLAGS</code> .....     | 714        | <code>_POSIX2_PBS_ACCOUNTING</code> .....   | 1982          |
| 14 | <code>_CS_XBS5_LP64_OFF64_LIBS</code> .....        | 714        | <code>_POSIX2_PBS_LOCATE</code> .....       | 1982          |
| 15 | <code>_CS_XBS5_LP64_OFF64_LINTFLAGS</code> .....   | 714        | <code>_POSIX2_PBS_MESSAGE</code> .....      | 1982          |
| 16 | <code>_CS_XBS5_LPBIG_OFFBIG_CFLAGS</code> .....    | 714        | <code>_POSIX2_PBS_TRACK</code> .....        | 1982          |
| 17 | <code>_CS_XBS5_LPBIG_OFFBIG_LDFLAGS</code> .....   | 714        | <code>_POSIX2_SW_DEV</code> .....           | 1982          |
| 18 | <code>_CS_XBS5_LPBIG_OFFBIG_LIBS</code> .....      | 714        | <code>_POSIX2_UPE</code> .....              | 1982          |
| 19 | <code>_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS</code> ..... | 714        | <code>_POSIX2_VERSION</code> .....          | 1982          |
| 20 | <code>_exit</code> .....                           | 801, 2100  | <code>_POSIX_ADVISORY_INFO</code> .....     | 1980          |
| 21 | <code>_Exit()</code> .....                         | <b>587</b> | <code>_POSIX_ASYNCHRONOUS_IO</code> .....   | 1980          |
| 22 | <code>_IOFBF</code> .....                          | 1790, 1834 | <code>_POSIX_ASYNC_IO</code> .....          | 892           |
| 23 | <code>_IOLBF</code> .....                          | 870, 1834  | <code>_POSIX_BARRIERS</code> .....          | 1980          |
| 24 | <code>_IONBF</code> .....                          | 1790, 1834 | <code>_POSIX_BASE</code> .....              | 1981          |
| 25 | <code>_longjmp()</code> .....                      | <b>588</b> | <code>_POSIX_CHOWN_RESTRICTED</code> .....  | 688, 892, 894 |
| 26 | <code>_LVL</code> .....                            | 515        | <code>_POSIX_CLOCK_SELECTION</code> .....   | 1981          |
| 27 | <code>_MAX</code> .....                            | 514        | <code>_POSIX_CPUTIME</code> .....           | 1981          |
| 28 | <code>_PC constants</code>                         |            | <code>_POSIX_C_LANG_SUPPORT</code> .....    | 1981          |
| 29 | used in <code>pathconf</code> .....                | 892        | <code>_POSIX_C_LANG_SUPPORT_R</code> .....  | 1981          |
| 30 | <code>_PC_ALLOC_SIZE_MIN</code> .....              | 892        | <code>_POSIX_C_SOURCE</code> .....          | 512-513       |
| 31 | <code>_PC_ASYNC_IO</code> .....                    | 892        | <code>_POSIX_DEVICE_IO</code> .....         | 1981          |
| 32 | <code>_PC_CHOWN_RESTRICTED</code> .....            | 892        | <code>_POSIX_DEVICE_SPECIFIC</code> .....   | 1981          |
| 33 | <code>_PC_FILESIZEBITS</code> .....                | 892        | <code>_POSIX_DEVICE_SPECIFIC_R</code> ..... | 1981          |
| 34 | <code>_PC_LINK_MAX</code> .....                    | 892        | <code>_POSIX_FD_MGMT</code> .....           | 1981          |
| 35 | <code>_PC_MAX_CANON</code> .....                   | 892        | <code>_POSIX_FIFO</code> .....              | 1981          |
| 36 | <code>_PC_MAX_INPUT</code> .....                   | 892        | <code>_POSIX_FILE_ATTRIBUTES</code> .....   | 1981          |
| 37 | <code>_PC_NAME_MAX</code> .....                    | 892        | <code>_POSIX_FILE_LOCKING</code> .....      | 1981          |
| 38 | <code>_PC_NO_TRUNC</code> .....                    | 892        | <code>_POSIX_FILE_SYSTEM</code> .....       | 1981          |
| 39 | <code>_PC_PATH_MAX</code> .....                    | 892        | <code>_POSIX_FSYNC</code> .....             | 1981          |
| 40 | <code>_PC_PIPE_BUF</code> .....                    | 892        | <code>_POSIX_JOB_CONTROL</code> .....       | 1981          |
| 41 | <code>_PC_PRIO_IO</code> .....                     | 892        | <code>_POSIX_MAPPED_FILES</code> .....      | 1981          |
| 42 | <code>_PC_REC_INCR_XFER_SIZE</code> .....          | 892        | <code>_POSIX_MEMLOCK</code> .....           | 1981          |
| 43 | <code>_PC_REC_MAX_XFER_SIZE</code> .....           | 892        | <code>_POSIX_MEMLOCK_RANGE</code> .....     | 1981          |
| 44 | <code>_PC_REC_MIN_XFER_SIZE</code> .....           | 892        | <code>_POSIX_MEMORY_PROTECTION</code> ..... | 1981          |
| 45 | <code>_PC_REC_XFER_ALIGN</code> .....              | 892        | <code>_POSIX_MESSAGE_PASSING</code> .....   | 1981          |
| 46 | <code>_PC_SYMLINK_MAX</code> .....                 | 892        | <code>_POSIX_MONOTONIC_CLOCK</code> .....   | 1981          |
| 47 | <code>_PC_SYNC_IO</code> .....                     | 892        | <code>_POSIX_MULTIPLE_PROCESS</code> .....  | 1981          |

|    |                                        |                 |                             |                  |
|----|----------------------------------------|-----------------|-----------------------------|------------------|
| 48 | _POSIX_NETWORKING.....                 | 1981            | _SC_2_C_DEV.....            | 1982             |
| 49 | _POSIX_NO_TRUNC.....                   | 892             | _SC_2_C_VERSION.....        | 1982             |
| 50 | _POSIX_OPEN_MAX.....                   | 1062            | _SC_2_FORT_DEV.....         | 1982             |
| 51 | _POSIX_PIPE.....                       | 1981            | _SC_2_FORT_RUN.....         | 1982             |
| 52 | _POSIX_PRIORITIZED_IO.....             | 544, 1981       | _SC_2_LOCALEDEF.....        | 1982             |
| 53 | _POSIX_PRIORITY_SCHEDULING.....        | 544, 1981       | _SC_2_PBS_ACCOUNTING.....   | 1982             |
| 54 | _POSIX_PRIO_IO.....                    | 892             | _SC_2_PBS_LOCATE.....       | 1982             |
| 55 | _POSIX_READER_WRITER_LOCKS.....        | 1981            | _SC_2_PBS_MESSAGE.....      | 1982             |
| 56 | _POSIX_REALTIME_SIGNALS.....           | 1981            | _SC_2_PBS_TRACK.....        | 1982             |
| 57 | _POSIX_REGEX.....                      | 1981            | _SC_2_SW_DEV.....           | 1982             |
| 58 | _POSIX_SAVED_IDS.....                  | 1981            | _SC_2_UPE.....              | 1982             |
| 59 | _POSIX_SEMAPHORES.....                 | 1981            | _SC_2_VERSION.....          | 1378, 1982       |
| 60 | _POSIX_SHARED_MEMORY_OBJECTS.....      | 1981            | _SC_ADVISORY_INFO.....      | 1980             |
| 61 | _POSIX_SHELL.....                      | 1981            | _SC_AIO_LISTIO_MAX.....     | 1980             |
| 62 | _POSIX_SIGNALS.....                    | 1981            | _SC_AIO_MAX.....            | 1980             |
| 63 | _POSIX_SINGLE_PROCESS.....             | 1981            | _SC_AIO_PRIO_DELTA_MAX..... | 1980             |
| 64 | _POSIX_SOURCE.....                     | 513             | _SC_ARG_MAX.....            | 1980             |
| 65 | _POSIX_SPAWN.....                      | 1981            | _SC_ASYNCIO.....            | 1980             |
| 66 | _POSIX_SPIN_LOCKS.....                 | 1981            | _SC_ATEXIT_MAX.....         | 1980             |
| 67 | _POSIX_SPARADIC_SERVER.....            | 1981            | _SC_BARRIERS.....           | 1980             |
| 68 | _POSIX_SYNCHRONIZED_IO.....            | 1981            | _SC_BASE.....               | 1981             |
| 69 | _POSIX_SYNC_IO.....                    | 892             | _SC_BC_BASE_MAX.....        | 1980             |
| 70 | _POSIX_SYSTEM_DATABASE.....            | 1981            | _SC_BC_DIM_MAX.....         | 1980             |
| 71 | _POSIX_SYSTEM_DATABASE_R.....          | 1981            | _SC_BC_SCALE_MAX.....       | 1980             |
| 72 | _POSIX_THREADS.....                    | 738, 1982, 2041 | _SC_BC_STRING_MAX.....      | 1980             |
| 73 | _POSIX_THREAD_ATTR_STACKADDR.....      | 1982            | _SC_CHILD_MAX.....          | 1980             |
| 74 | _POSIX_THREAD_ATTR_STACKSIZE.....      | 1982            | _SC_CLK_TCK.....            | 1980, 2036       |
| 75 | _POSIX_THREAD_CPUTIME.....             | 1982            | _SC_CLOCK_SELECTION.....    | 1981             |
| 76 | _POSIX_THREAD_PRIORITY_SCHEDULING..... | 1982            | _SC_COLL_WEIGHTS_MAX.....   | 1980             |
| 77 | .....                                  | 1982            | _SC_CPUTIME.....            | 1981             |
| 78 | _POSIX_THREAD_PRIO_INHERIT.....        | 1982            | _SC_C_LANG_SUPPORT.....     | 1981             |
| 79 | _POSIX_THREAD_PRIO_PROTECT.....        | 1982            | _SC_C_LANG_SUPPORT_R.....   | 1981             |
| 80 | _POSIX_THREAD_PROCESS_SHARED.....      | 1599            | _SC_DELAYTIMER_MAX.....     | 1980             |
| 81 | .....                                  | 1982            | _SC_DEVICE_IO.....          | 1981             |
| 82 | _POSIX_THREAD_SAFE_FUNCTIONS.....      | 738             | _SC_DEVICE_SPECIFIC.....    | 1981             |
| 83 | .....                                  | 1982, 2041      | _SC_DEVICE_SPECIFIC_R.....  | 1981             |
| 84 | _POSIX_THREAD_SPARADIC_SERVER.....     | 1982            | _SC_EXPR_NEST_MAX.....      | 1980             |
| 85 | _POSIX_THREAD_THREADS_MAX.....         | 1573            | _SC_FD_MGMT.....            | 1981             |
| 86 | _POSIX_TIMEOUTS.....                   | 1982            | _SC_FIFO.....               | 1981             |
| 87 | _POSIX_TIMERS.....                     | 1982            | _SC_FILE_ATTRIBUTES.....    | 1981             |
| 88 | _POSIX_TYPED_MEMORY_OBJECTS.....       | 1982            | _SC_FILE_LOCKING.....       | 1981             |
| 89 | _POSIX_USER_GROUPS.....                | 1982            | _SC_FILE_SYSTEM.....        | 1981             |
| 90 | _POSIX_USER_GROUPS_R.....              | 1982            | _SC_FSYNC.....              | 1981             |
| 91 | _POSIX_VDISABLE.....                   | 892             | _SC_GETGR_R_SIZE_MAX.....   | 1005, 1008, 1980 |
| 92 | _POSIX_VERSION.....                    | 1982, 2065      | _SC_GETPW_R_SIZE_MAX.....   | 1055, 1058, 1980 |
| 93 | _PROCESS.....                          | 515             | _SC_IOV_MAX.....            | 1980             |
| 94 | _REGEX_VERSION.....                    | 1982            | _SC_JOB_CONTROL.....        | 1981             |
| 95 | _SC constants                          |                 | _SC_LINE_MAX.....           | 1980             |
| 96 | in sysconf.....                        | 1980            | _SC_LOGIN_NAME_MAX.....     | 1980             |
| 97 | _SC_2_CHAR_TERM.....                   | 1982            | _SC_MEMLOCK.....            | 1981             |
| 98 | _SC_2_C_BIND.....                      | 1982            | _SC_MEMLOCK_RANGE.....      | 1981             |

## Index

|     |                                       |                  |                                 |            |
|-----|---------------------------------------|------------------|---------------------------------|------------|
| 99  | _SC_MEMORY_PROTECTION.....            | 1981             | _SC_THREAD_THREADS_MAX.....     | 1983       |
| 100 | _SC_MESSAGE_PASSING.....              | 1981             | _SC_TIMEOUTS.....               | 1982       |
| 101 | _SC_MONOTONIC_CLOCK.....              | 1981             | _SC_TIMERS.....                 | 1982       |
| 102 | _SC_MQ_OPEN_MAX.....                  | 1980             | _SC_TIMER_MAX.....              | 1983       |
| 103 | _SC_MQ_PRIO_MAX.....                  | 1980             | _SC_TTY_NAME_MAX.....           | 1983       |
| 104 | _SC_MULTIPLE_PROCESS.....             | 1981             | _SC_TYPED_MEMORY_OBJECTS.....   | 1982       |
| 105 | _SC_NETWORKING.....                   | 1981             | _SC_TZNAME_MAX.....             | 1983       |
| 106 | _SC_NGROUPS_MAX.....                  | 1980             | _SC_USER_GROUPS.....            | 1982       |
| 107 | _SC_OPEN_MAX.....                     | 1980             | _SC_USER_GROUPS_R.....          | 1982       |
| 108 | _SC_PAGESIZE.....                     | 1289, 1384, 1982 | _SC_VERSION.....                | 1982       |
| 109 | _SC_PAGE_SIZE.....                    | 1289, 1982       | _SC_XBS5_ILP32_OFF32.....       | 1983       |
| 110 | _SC_PIPE.....                         | 1981             | _SC_XBS5_ILP32_OFFBIG.....      | 1983       |
| 111 | _SC_PRIORITIZED_IO.....               | 1981             | _SC_XBS5_LP64_OFF64.....        | 1983       |
| 112 | _SC_PRIORITY_SCHEDULING.....          | 1981             | _SC_XBS5_LPBIG_OFFBIG.....      | 1983       |
| 113 | _SC_READER_WRITER_LOCKS.....          | 1981             | _SC_XOPEN_CRYPT.....            | 1983       |
| 114 | _SC_REALTIME_SIGNALS.....             | 1981             | _SC_XOPEN_ENH_I18N.....         | 1983       |
| 115 | _SC_REGEX.....                        | 1981             | _SC_XOPEN_LEGACY.....           | 1983       |
| 116 | _SC_REGEX_VERSION.....                | 1982             | _SC_XOPEN_REALTIME.....         | 1983       |
| 117 | _SC_RE_DUP_MAX.....                   | 1983             | _SC_XOPEN_REALTIME_THREADS..... | 1983       |
| 118 | _SC_RTSIG_MAX.....                    | 1983             | _SC_XOPEN_SHM.....              | 1983       |
| 119 | _SC_SAVED_IDS.....                    | 1981             | _SC_XOPEN_UNIX.....             | 1983       |
| 120 | _SC_SEMAPHORES.....                   | 1981             | _SC_XOPEN_VERSION.....          | 1983       |
| 121 | _SC_SEM_NSEMS_MAX.....                | 1983             | _SC_XOPEN_XCU_VERSION.....      | 1983       |
| 122 | _SC_SEM_VALUE_MAX.....                | 1983             | _setjmp.....                    | 588        |
| 123 | _SC_SHARED_MEMORY_OBJECTS.....        | 1981             | _setjmp().....                  | <b>590</b> |
| 124 | _SC_SHELL.....                        | 1981             | _TIME.....                      | 515        |
| 125 | _SC_SIGNALS.....                      | 1981             | _tolower().....                 | <b>591</b> |
| 126 | _SC_SIGQUEUE_MAX.....                 | 1983             | _toupper().....                 | <b>592</b> |
| 127 | _SC_SINGLE_PROCESS.....               | 1981             | _XBS5_ILP32_OFF32.....          | 1983       |
| 128 | _SC_SPAWN.....                        | 1981             | _XBS5_ILP32_OFFBIG.....         | 1983       |
| 129 | _SC_SPIN_LOCKS.....                   | 1981             | _XBS5_LP64_OFF64.....           | 1983       |
| 130 | _SC_SPORADIC_SERVER.....              | 1981             | _XBS5_LPBIG_OFFBIG.....         | 1983       |
| 131 | _SC_STREAM_MAX.....                   | 1983             | _XOPEN_CRYPT.....               | 1983       |
| 132 | _SC_SYMLOOP_MAX.....                  | 1983             | _XOPEN_ENH_I18N.....            | 1983       |
| 133 | _SC_SYNCHRONIZED_IO.....              | 1981             | _XOPEN_LEGACY.....              | 1983       |
| 134 | _SC_SYSTEM_DATABASE.....              | 1981             | _XOPEN_REALTIME.....            | 831, 1983  |
| 135 | _SC_SYSTEM_DATABASE_R.....            | 1981             | _XOPEN_REALTIME_THREADS.....    | 1983       |
| 136 | _SC_THREADS.....                      | 1982             | _XOPEN_SHM.....                 | 1983       |
| 137 | _SC_THREAD_ATTR_STACKADDR.....        | 1982             | _XOPEN_SOURCE.....              | 512-513    |
| 138 | _SC_THREAD_ATTR_STACKSIZE.....        | 1982             | _XOPEN_UNIX.....                | 1983       |
| 139 | _SC_THREAD_CPUTIME.....               | 1982             | _XOPEN_VERSION.....             | 1983       |
| 140 | _SC_THREAD_DESTRUCTOR_ITERATIONS..... | 1983             | _XOPEN_XCU_VERSION.....         | 1983       |
| 141 | .....                                 | 1983             | a64l().....                     | <b>593</b> |
| 142 | _SC_THREAD_KEYS_MAX.....              | 1983             | ABDAY_1.....                    | 1346       |
| 143 | _SC_THREAD_PRIORITY_SCHEDULING.....   | 1982             | abort().....                    | <b>595</b> |
| 144 | _SC_THREAD_PRIO_INHERIT.....          | 1982             | abs().....                      | <b>597</b> |
| 145 | _SC_THREAD_PRIO_PROTECT.....          | 1982             | accept().....                   | <b>598</b> |
| 146 | _SC_THREAD_PROCESS_SHARED.....        | 1982             | access().....                   | <b>600</b> |
| 147 | _SC_THREAD_SAFE_FUNCTIONS.....        | 1982             | acos().....                     | <b>603</b> |
| 148 | _SC_THREAD_SPORADIC_SERVER.....       | 1982             | acosh().....                    | <b>605</b> |
| 149 | _SC_THREAD_STACK_MIN.....             | 1983             | ACTION.....                     | 1094       |

|     |                        |                          |                                 |                                    |
|-----|------------------------|--------------------------|---------------------------------|------------------------------------|
| 150 | address information    | 921                      | BC_BASE_MAX                     | 1980                               |
| 151 | address string         | 921                      | BC_DIM_MAX                      | 1980                               |
| 152 | addrinfo structure     | 921                      | BC_SCALE_MAX                    | 1980                               |
| 153 | ADV                    | 501                      | BC_STRING_MAX                   | 1980                               |
| 154 | AF_                    | 516                      | BE                              | 501                                |
| 155 | AIO                    | 501                      | bind()                          | 645                                |
| 156 | AIO_                   | 514                      | BOOT_TIME                       | 781-782                            |
| 157 | AIO_ALLDONE            | 607                      | broadcasting a condition        | 1536                               |
| 158 | aio_cancel()           | 607                      | BSD                             | 621, 689, 804, 828, 836, 894       |
| 159 | AIO_CANCELED           | 607                      |                                 | 1044, 1194, 1269, 1336, 1684, 1723 |
| 160 | aio_error()            | 609                      |                                 | 1732, 1811, 1856, 1883, 2004, 2026 |
| 161 | aio_fsync()            | 610                      |                                 | 2065, 2100                         |
| 162 | AIO_LISTIO_MAX         | 1213, 1980               | bsd_signal()                    | 647                                |
| 163 | AIO_MAX                | 1213, 1980               | bsearch()                       | 649                                |
| 164 | AIO_NOTCANCELED        | 607                      | btowc()                         | 652                                |
| 165 | AIO_PRIO_DELTA_MAX     | 544, 1980                | buffer cache                    | 950                                |
| 166 | aio_read()             | 612                      | BUFSIZ                          | 1790                               |
| 167 | aio_return()           | 615                      | BUS_                            | 516                                |
| 168 | aio_suspend()          | 616                      | byte-oriented stream            | 537                                |
| 169 | aio_write()            | 618                      | byte-stream mode                | 1680                               |
| 170 | alarm()                | 621                      | bzero()                         | 653                                |
| 171 | anycast                | 570                      | cabs()                          | 654                                |
| 172 | ANYMARK                | 1135                     | cacos()                         | 655                                |
| 173 | appropriate privileges | 601, 893                 | cacosh()                        | 656                                |
| 174 | argc                   | 796                      | calloc()                        | 657                                |
| 175 | ARG_MAX                | 521, 791, 793, 797, 1980 | can                             | 497                                |
| 176 | asctime()              | 623                      | cancel-safe                     | 1641                               |
| 177 | asctime_r              | 623                      | cancelability state             | 1574, 1641                         |
| 178 | asin()                 | 626                      | cancelability states            | 557                                |
| 179 | asinh                  | 605                      | cancelability type              | 1574, 1641                         |
| 180 | asinh()                | 628                      | cancelation cleanup handler     | 1533, 1545                         |
| 181 | assert()               | 629                      |                                 | 1564, 1578                         |
| 182 | async-signal-safe      | 1486                     | cancelation points              | 558                                |
| 183 | atan()                 | 630                      | canceling execution of a thread | 1528                               |
| 184 | atan2()                | 632                      | canonical name                  | 922                                |
| 185 | atanh                  | 605                      | carg()                          | 659                                |
| 186 | atanh()                | 634                      | casin()                         | 660                                |
| 187 | atexit()               | 635                      | casinh()                        | 661                                |
| 188 | ATEXIT_MAX             | 635, 1980                | catan()                         | 662                                |
| 189 | atof()                 | 636                      | catanh()                        | 663                                |
| 190 | atoi()                 | 637                      | catclose()                      | 664                                |
| 191 | atol()                 | 639                      | catgets()                       | 665                                |
| 192 | background             | 1811                     | catopen()                       | 667                                |
| 193 | background process     | 2012                     | cbrt()                          | 669                                |
| 194 | BAR                    | 501                      | ccos()                          | 670                                |
| 195 | basename()             | 641                      | ccosh()                         | 671                                |
| 196 | baud rate functions    | 675                      | CD                              | 501                                |
| 197 | bcmp()                 | 643                      | ceil()                          | 672                                |
| 198 | bcopy()                | 644                      | cexp()                          | 674                                |
| 199 | BC_ constants          |                          | cfgetispeed()                   | 675                                |
| 200 | in sysconf             | 1980                     | cfgetospeed()                   | 677                                |



## Index

|     |                                              |                        |                                         |                |
|-----|----------------------------------------------|------------------------|-----------------------------------------|----------------|
| 201 | cfsetospeed()                                | 678                    | copysign()                              | 721            |
| 202 | cfsetospeed()                                | 680                    | core                                    | 2101           |
| 203 | change current working directory             | 683                    | core file                               | 803            |
| 204 | change file modes                            | 686                    | cos()                                   | 722            |
| 205 | change owner and group of file               | 689                    | cosh()                                  | 724            |
| 206 | CHAR_MAX                                     | 1219, 1221             | covert channel                          | 1194           |
| 207 | chdir()                                      | 682                    | cpow()                                  | 726            |
| 208 | CHILD_MAX                                    | 888, 1980              | cproj()                                 | 727            |
| 209 | chmod()                                      | 684                    | CPT                                     | 501            |
| 210 | chown()                                      | 688                    | creal()                                 | 728            |
| 211 | cimag()                                      | 692                    | creat()                                 | 729            |
| 212 | CLD_                                         | 516                    | create a per-process timer              | 2029           |
| 213 | clearerr()                                   | 693                    | create an interprocess channel          | 1371           |
| 214 | clock tick                                   | 621, 1984, 2036        | create session and set process group ID | 1824           |
| 215 | clock ticks/second                           | 1980                   | CRYPT                                   | 731, 767, 1803 |
| 216 | clock()                                      | 694                    | crypt()                                 | 731            |
| 217 | CLOCKS_PER_SEC                               | 694                    | CS                                      | 502            |
| 218 | CLOCK_                                       | 515                    | csin()                                  | 733            |
| 219 | clock_getcpuclockid()                        | 696                    | csinh()                                 | 734            |
| 220 | clock_getres()                               | 697                    | csqrt()                                 | 735            |
| 221 | clock_gettime                                | 697                    | ctan()                                  | 736            |
| 222 | CLOCK_MONOTONIC                              | 551, 701               | ctanh()                                 | 737            |
| 223 | clock_nanosleep()                            | 700                    | ctermid()                               | 738            |
| 224 | CLOCK_PROCESS_CPUTIME_ID                     | 551                    | ctime()                                 | 740            |
| 225 | CLOCK_REALTIME                               | 551, 697, 701          | ctime_r                                 | 740            |
| 226 |                                              | 1336, 1595, 1764, 2028 | CX                                      | 502            |
| 227 | clock_settime                                | 697                    | data key creation                       | 1578           |
| 228 | CLOCK_THREAD_CPUTIME_ID                      | 552                    | data messages                           | 539            |
| 229 | clog()                                       | 703                    | data type                               | 583            |
| 230 | close a file                                 | 706                    | DATEMSK                                 | 993            |
| 231 | close()                                      | 704                    | daylight                                | 742, 2057      |
| 232 | closedir()                                   | 708                    | DBL_MANT_DIG                            | 672, 871       |
| 233 | closelog()                                   | 710                    | DBL_MAX_EXP                             | 672, 871       |
| 234 | CMSG_                                        | 516                    | DBM                                     | 743-744        |
| 235 | COLL_WEIGHTS_MAX                             | 1980                   | DBM_                                    | 516            |
| 236 | command interpreter                          |                        | dbm_clearerr()                          | 743            |
| 237 | portable                                     | 2100                   | dbm_close                               | 743            |
| 238 | compare thread IDs                           | 1563                   | dbm_delete                              | 743            |
| 239 | compilation environment                      | 512                    | dbm_error                               | 743            |
| 240 | condition variable initialization attributes | 1549                   | dbm_fetch                               | 743            |
| 241 | conforming application                       | 1899                   | dbm_firstkey                            | 743            |
| 242 | conforming application, strictly             | 621, 796               | DBM_INSERT                              | 743            |
| 243 | confstr()                                    | 714                    | dbm_nextkey                             | 743            |
| 244 | conj()                                       | 717                    | dbm_open                                | 743            |
| 245 | connect()                                    | 718                    | DBM_REPLACE                             | 743            |
| 246 | control data                                 | 539                    | dbm_store                               | 743            |
| 247 | control-normal                               | 1680                   | DEAD_PROCESS                            | 781-782        |
| 248 | conversion descriptor                        | 791, 796, 1101-1104    | deferred cancelability                  | 1574           |
| 249 | conversion specification                     | 898, 932, 966          | delay process execution                 | 1898           |
| 250 |                                              | 975, 1932, 1936, 1948  | DELAYTIMER_MAX                          | 1980, 2032     |
| 251 | modified                                     | 1937                   | dependency ordering                     | 755            |

|     |                                      |                                              |  |
|-----|--------------------------------------|----------------------------------------------|--|
| 252 | DES                                  |                                              |  |
| 253 | decryption algorithm.....            | 767                                          |  |
| 254 | descriptive name .....               | 921                                          |  |
| 255 | destroying a mutex .....             | 1584                                         |  |
| 256 | destroying condition variables.....  | 1540                                         |  |
| 257 | destructor functions.....            | 1578                                         |  |
| 258 | detaching a thread.....              | 1561                                         |  |
| 259 | difftime().....                      | <b>746</b>                                   |  |
| 260 | DIR.....                             | 583, 708, 1359, 1361, 1687, 1726, 1752, 2020 |  |
| 261 | directive.....                       | 898, 932, 966, 975, 1948                     |  |
| 262 | modified.....                        | 1949                                         |  |
| 263 | directory operations.....            | 1360                                         |  |
| 264 | dirent structure .....               | 1360                                         |  |
| 265 | dirname() .....                      | <b>747</b>                                   |  |
| 266 | div().....                           | <b>749</b>                                   |  |
| 267 | dlclose().....                       | <b>750</b>                                   |  |
| 268 | dlopen().....                        | <b>752</b>                                   |  |
| 269 | dlopen().....                        | <b>754</b>                                   |  |
| 270 | dlsym() .....                        | <b>757</b>                                   |  |
| 271 | dot .....                            | 1360, 1723                                   |  |
| 272 | dot-dot.....                         | 1360, 1723                                   |  |
| 273 | drand48() .....                      | <b>759</b>                                   |  |
| 274 | dup() .....                          | <b>762</b>                                   |  |
| 275 | dup2.....                            | 762                                          |  |
| 276 | dynamic package initialization ..... | 1616                                         |  |
| 277 | E2BIG .....                          | <b>521</b>                                   |  |
| 278 | EACCESS.....                         | <b>522</b>                                   |  |
| 279 | EADDRINUSE .....                     | <b>522</b>                                   |  |
| 280 | EADDRNOTAVAIL.....                   | <b>522</b>                                   |  |
| 281 | EAFNOSUPPORT .....                   | <b>522</b>                                   |  |
| 282 | EAGAIN.....                          | <b>522</b> , 527                             |  |
| 283 | EALREADY .....                       | <b>522</b>                                   |  |
| 284 | EBADF .....                          | <b>522</b>                                   |  |
| 285 | EBADMSG .....                        | <b>522</b>                                   |  |
| 286 | EBUSY .....                          | <b>522</b>                                   |  |
| 287 | ECANCELED .....                      | <b>522</b>                                   |  |
| 288 | ECHILD .....                         | <b>522</b>                                   |  |
| 289 | ECONNABORTED .....                   | <b>522</b>                                   |  |
| 290 | ECONNREFUSED.....                    | <b>522</b>                                   |  |
| 291 | ECONNRESET .....                     | <b>523</b>                                   |  |
| 292 | ecvt().....                          | <b>765</b>                                   |  |
| 293 | EDEADLK.....                         | <b>523</b>                                   |  |
| 294 | EDESTADDRREQ .....                   | <b>523</b>                                   |  |
| 295 | EDOM .....                           | <b>523</b>                                   |  |
| 296 | EDQUOT .....                         | <b>523</b>                                   |  |
| 297 | EEXIST .....                         | <b>523</b>                                   |  |
| 298 | EFAULT .....                         | <b>523</b>                                   |  |
| 299 | EFBIG .....                          | <b>523</b>                                   |  |
| 300 | effective group ID.....              | 689, 797, 1012                               |  |
| 301 | effective user ID .....              | 601, 797, 1194                               |  |
| 302 | EHOSTUNREACH .....                   | 523                                          |  |
|     | EIDRM.....                           | <b>523</b>                                   |  |
|     | Eighth Edition UNIX .....            | 2170                                         |  |
|     | EILSEQ.....                          | 538                                          |  |
|     | EINPROGRESS.....                     | <b>523</b> , 544                             |  |
|     | EINTR.....                           | <b>523</b> , 560                             |  |
|     | EINVAL .....                         | <b>524</b>                                   |  |
|     | EIO .....                            | <b>524</b>                                   |  |
|     | EISCONN.....                         | <b>524</b>                                   |  |
|     | EISDIR .....                         | <b>524</b>                                   |  |
|     | ELOOP .....                          | <b>524</b>                                   |  |
|     | ELSIZE .....                         | 1239                                         |  |
|     | EMFILE .....                         | <b>524</b>                                   |  |
|     | EMLINK .....                         | <b>524</b>                                   |  |
|     | EMPTY.....                           | 782                                          |  |
|     | EMSGSIZE .....                       | <b>524</b>                                   |  |
|     | EMULTIHOP .....                      | <b>524</b>                                   |  |
|     | ENAMETOOLONG.....                    | <b>524</b>                                   |  |
|     | encrypt().....                       | <b>767</b>                                   |  |
|     | endgrent() .....                     | <b>769</b>                                   |  |
|     | endhostent() .....                   | 771                                          |  |
|     | endnetent() .....                    | 773                                          |  |
|     | endprotoent() .....                  | 775                                          |  |
|     | endpwent() .....                     | 777                                          |  |
|     | endservent() .....                   | 779                                          |  |
|     | endutent().....                      | <b>781</b>                                   |  |
|     | ENETDOWN.....                        | <b>524</b>                                   |  |
|     | ENETRESET .....                      | <b>524</b>                                   |  |
|     | ENETUNREACH .....                    | <b>524</b>                                   |  |
|     | ENFILE .....                         | <b>524</b>                                   |  |
|     | ENOBUFS.....                         | <b>525</b>                                   |  |
|     | ENODATA .....                        | <b>525</b>                                   |  |
|     | ENODEV .....                         | <b>525</b>                                   |  |
|     | ENOENT .....                         | <b>525</b>                                   |  |
|     | ENOEXEC .....                        | <b>525</b>                                   |  |
|     | ENOLCK .....                         | <b>525</b>                                   |  |
|     | ENOLINK .....                        | <b>525</b>                                   |  |
|     | ENOMEM.....                          | <b>525</b>                                   |  |
|     | ENOMSG.....                          | <b>525</b>                                   |  |
|     | ENOPROTOPT .....                     | <b>525</b>                                   |  |
|     | ENOSPC .....                         | <b>525</b>                                   |  |
|     | ENOSR.....                           | <b>525</b>                                   |  |
|     | ENOSTR .....                         | <b>525</b>                                   |  |
|     | ENOSYS.....                          | <b>525</b>                                   |  |
|     | ENOTCONN.....                        | <b>525</b>                                   |  |
|     | ENOTDIR.....                         | <b>526</b>                                   |  |
|     | ENOTEMPTY .....                      | <b>526</b>                                   |  |
|     | ENOTSOCK.....                        | <b>526</b>                                   |  |
|     | ENOTSUP .....                        | <b>526</b>                                   |  |
|     | ENOTTY.....                          | <b>526</b>                                   |  |
|     | ENTRY.....                           | 1094                                         |  |
|     | environ.....                         | <b>784</b> , 796                             |  |

## Index

|     |                                         |                       |                              |                                       |
|-----|-----------------------------------------|-----------------------|------------------------------|---------------------------------------|
| 303 | envp.....                               | 796                   | extension                    |                                       |
| 304 | ENXIO.....                              | <b>526</b>            | CX.....                      | 502                                   |
| 305 | EOPNOTSUPP.....                         | <b>526</b>            | OH.....                      | 503                                   |
| 306 | EOVERFLOW.....                          | <b>526</b>            | XSI.....                     | 507                                   |
| 307 | EPERM.....                              | <b>526</b>            | extensions to setlocale..... | 1806                                  |
| 308 | EPIPE.....                              | <b>526</b>            | fabs().....                  | <b>811</b>                            |
| 309 | EPROTO.....                             | <b>526</b>            | fattach().....               | <b>813</b>                            |
| 310 | EPROTONOSUPPORT.....                    | <b>526</b>            | fchdir().....                | <b>816</b>                            |
| 311 | EPROTOTYPE.....                         | <b>526</b>            | fchmod().....                | <b>817</b>                            |
| 312 | erand48.....                            | 759                   | fchown().....                | <b>819</b>                            |
| 313 | erand48().....                          | <b>785</b>            | fclose().....                | <b>821</b>                            |
| 314 | ERANGE.....                             | <b>526</b>            | fcntl().....                 | <b>823</b>                            |
| 315 | erf().....                              | <b>786</b>            | fcvt.....                    | 765                                   |
| 316 | erfc.....                               | 786                   | fcvt().....                  | <b>830</b>                            |
| 317 | EROFS.....                              | <b>527</b>            | FD.....                      | <b>502</b>                            |
| 318 | errno.....                              | <b>788</b> , 827      | fdatasync().....             | <b>831</b>                            |
| 319 | error descriptions.....                 | 981                   | fdetach().....               | <b>832</b>                            |
| 320 | error numbers.....                      | 521                   | fdim().....                  | <b>834</b>                            |
| 321 | additional.....                         | 527                   | fdopen().....                | <b>835</b>                            |
| 322 | ESPIPE.....                             | <b>527</b>            | FD_CLOEXEC.....              | 536, 667, 791, 823, 1104              |
| 323 | ESRCH.....                              | <b>527</b>            | .....                        | 1351, 1360, 1371, 1390, 1397, 1836    |
| 324 | EST.....                                | 2057                  | FD_CLR().....                | <b>586</b>                            |
| 325 | establishing cancelation handlers.....  | 1533                  | FD_ISSET.....                | 586                                   |
| 326 | ESTALE.....                             | <b>527</b>            | FD_SET.....                  | 586                                   |
| 327 | ETIME.....                              | <b>527</b>            | FD_ZERO.....                 | 586                                   |
| 328 | ETIMEDOUT.....                          | <b>527</b>            | feature test macro.....      | 512, 986                              |
| 329 | ETXTBSY.....                            | <b>527</b>            | _POSIX_C_SOURCE.....         | 512                                   |
| 330 | EWouldBlock.....                        | <b>527</b>            | _XOPEN_SOURCE.....           | 512                                   |
| 331 | examine and change blocked signals..... | 1646                  | feclearexcept().....         | <b>838</b>                            |
| 332 | examine and change signal action.....   | 1856                  | fegetenv().....              | <b>839</b>                            |
| 333 | EXDEV.....                              | <b>527</b>            | fegetexceptflag().....       | <b>840</b>                            |
| 334 | exec.....                               | <b>790</b>            | fegetround().....            | <b>841</b>                            |
| 335 | of shell scripts.....                   | 796                   | fehldexcept().....           | <b>843</b>                            |
| 336 | exec family.....                        | 601, 706, 827, 868    | feof().....                  | <b>844</b>                            |
| 337 | .....                                   | 890, 1486, 1812, 2100 | feraiseexcept().....         | <b>845</b>                            |
| 338 | execl.....                              | 790                   | ferror().....                | <b>846</b>                            |
| 339 | execle.....                             | 790                   | fesetenv().....              | <b>847</b>                            |
| 340 | execlp.....                             | 790                   | fesetexceptflag().....       | <b>848</b>                            |
| 341 | execute a file.....                     | 796                   | fesetround().....            | <b>849</b>                            |
| 342 | execution time monitoring.....          | 551                   | fetestexcept().....          | <b>850</b>                            |
| 343 | execv.....                              | 790                   | feupdateenv().....           | <b>852</b>                            |
| 344 | execve.....                             | 790                   | fflush().....                | <b>853</b>                            |
| 345 | execvp.....                             | 790                   | ffs().....                   | <b>856</b>                            |
| 346 | exit().....                             | <b>801</b>            | fgetc().....                 | <b>857</b>                            |
| 347 | EXIT_FAILURE.....                       | 801                   | fgetpos().....               | <b>860</b>                            |
| 348 | EXIT_SUCCESS.....                       | 801, 804              | fgets().....                 | <b>862</b>                            |
| 349 | exp().....                              | <b>806</b>            | fgetwc().....                | <b>864</b>                            |
| 350 | exp2().....                             | <b>808</b>            | fgetws().....                | <b>866</b>                            |
| 351 | expm1().....                            | <b>810</b>            | FIFO.....                    | 1271-1272, 1355, 2168                 |
| 352 | EXPR_NEST_MAX.....                      | 1980                  | FILE.....                    | 583, 693, 821, 835, 844, 846          |
| 353 |                                         |                       | .....                        | 853, 857, 860, 862, 864, 866, 868-869 |

|     |                                               |                                  |                        |
|-----|-----------------------------------------------|----------------------------------|------------------------|
| 354 | .....883, 898, 910, 912, 914, 916-917, 926    | FR.....                          | 502                    |
| 355 | .....932, 939, 942, 952, 964-966, 973, 975    | fread().....                     | 917                    |
| 356 | .....984-985, 1083, 1367, 1377, 1656-1657     | free().....                      | 919                    |
| 357 | .....1669, 1725, 1790, 1834, 1917, 2039, 2067 | freeaddrinfo().....              | 921                    |
| 358 | .....2069, 2088, 2091, 2093, 2095             | freehostent.....                 | 771                    |
| 359 | file                                          | freehostent().....               | 925                    |
| 360 | locking.....                                  | freopen().....                   | 926                    |
| 361 | file accessibility.....                       | frexp().....                     | 930                    |
| 362 | file control.....                             | FSC.....                         | 502                    |
| 363 | FILE object.....                              | fscanf().....                    | 932                    |
| 364 | file permission bits.....                     | fseek().....                     | 939                    |
| 365 | file permissions.....                         | fseeko.....                      | 939                    |
| 366 | file position indicator.....                  | fsetpos().....                   | 942                    |
| 367 | fileno().....                                 | fstat().....                     | 944                    |
| 368 | FILESIZEBITS.....                             | fstatvfs().....                  | 947                    |
| 369 | FIND.....                                     | fsync().....                     | 950                    |
| 370 | find string token.....                        | ftell().....                     | 952                    |
| 371 | flockfile().....                              | ftello.....                      | 952                    |
| 372 | floor().....                                  | ftime().....                     | 954                    |
| 373 | FLUSH.....                                    | ftok().....                      | 956                    |
| 374 | FLUSHR.....                                   | ftruncate().....                 | 958                    |
| 375 | FLUSHRW.....                                  | ftrylockfile.....                | 869                    |
| 376 | FLUSHW.....                                   | ftrylockfile().....              | 960                    |
| 377 | fma().....                                    | FTW.....                         | 516, 1341-1342         |
| 378 | fmax().....                                   | ftw().....                       | 961                    |
| 379 | fmin().....                                   | FTW_CHDIR.....                   | 1341                   |
| 380 | FMNAMESZ.....                                 | FTW_D.....                       | 961, 1341              |
| 381 | fmod().....                                   | FTW_DEPTH.....                   | 1341                   |
| 382 | fmtmsg().....                                 | FTW_DNR.....                     | 961, 1341-1342         |
| 383 | fnmatch().....                                | FTW_DP.....                      | 1341                   |
| 384 | FNM.....                                      | FTW_F.....                       | 961, 1341              |
| 385 | FNM_NOESCAPE.....                             | FTW_MOUNT.....                   | 1341                   |
| 386 | FNM_NOMATCH.....                              | FTW_NS.....                      | 961, 1341-1342         |
| 387 | FNM_PATHNAME.....                             | FTW_PHYS.....                    | 1341                   |
| 388 | FNM_PERIOD.....                               | FTW_SL.....                      | 961, 1341              |
| 389 | fopen().....                                  | FTW_SLN.....                     | 1341                   |
| 390 | FOPEN_MAX.....                                | fully-qualified domain name..... | 1032                   |
| 391 | foreground.....                               | functions.....                   | 511                    |
| 392 | fork handler.....                             | implementation.....              | 511                    |
| 393 | fork().....                                   | use.....                         | 511                    |
| 394 | forkall.....                                  | funlockfile.....                 | 869                    |
| 395 | format of entries.....                        | funlockfile().....               | 964                    |
| 396 | fpathconf().....                              | fwide().....                     | 965                    |
| 397 | fpclassify().....                             | fwprintf().....                  | 966                    |
| 398 | FPE.....                                      | fwrite().....                    | 973                    |
| 399 | fprintf().....                                | fwscanf().....                   | 975                    |
| 400 | fputc().....                                  | F_DUPFD.....                     | 762, 764, 823, 825-826 |
| 401 | fputs().....                                  | F_GETFD.....                     | 823, 825-826           |
| 402 | fputwc().....                                 | F_GETFL.....                     | 823, 825-826           |
| 403 | fputws().....                                 | F_GETLK.....                     | 824-826                |
| 404 | FQDN.....                                     | F_GETOWN.....                    | 823, 825               |

## Index

|     |                                           |              |                         |           |
|-----|-------------------------------------------|--------------|-------------------------|-----------|
| 405 | F_LOCK.....                               | 1226         | gethostid() .....       | 1019      |
| 406 | F_RDLCK.....                              | 826          | gethostname().....      | 1020      |
| 407 | F_SETFD.....                              | 823, 825-826 | getipnodebyaddr.....    | 1013      |
| 408 | F_SETFL.....                              | 823, 825-826 | getipnodebyaddr().....  | 1021      |
| 409 | F_SETLK.....                              | 824-827      | getipnodebyname.....    | 1013      |
| 410 | F_SETLKW.....                             | 558, 824-827 | getipnodebyname() ..... | 1022      |
| 411 | F_SETOWN.....                             | 823, 825     | getitimer().....        | 1023      |
| 412 | F_TEST.....                               | 1226         | getlogin().....         | 1025      |
| 413 | F_TLOCK.....                              | 1226         | getlogin_r.....         | 1025      |
| 414 | F_ULOCK.....                              | 1226         | getmsg().....           | 1028      |
| 415 | F_UNLCK.....                              | 824-825      | getnameinfo() .....     | 1032      |
| 416 | F_WRLCK.....                              | 826          | GETNCNT.....            | 1771-1772 |
| 417 | gai_strerror().....                       | 981          | getnetbyaddr.....       | 773       |
| 418 | gcvt.....                                 | 765          | getnetbyaddr().....     | 1034      |
| 419 | gcvt().....                               | 982          | getnetbyname.....       | 773       |
| 420 | get configurable path name variables..... | 894          | getnetbyname() .....    | 1035      |
| 421 | get configurable system variables.....    | 1984         | getnetent.....          | 773       |
| 422 | get file status.....                      | 1914         | getnetent() .....       | 1036      |
| 423 | get process times.....                    | 2036         | getopt().....           | 1037      |
| 424 | get supplementary group IDs.....          | 1011         | getpeername().....      | 1042      |
| 425 | get system time.....                      | 2026         | getpgid() .....         | 1043      |
| 426 | get thread ID.....                        | 1639         | getpgrp().....          | 1044      |
| 427 | get user name.....                        | 1026         | GETPID.....             | 1771-1772 |
| 428 | getaddrinfo.....                          | 921          | getpid().....           | 1045      |
| 429 | getaddrinfo().....                        | 983          | getpmsg.....            | 1028      |
| 430 | GETALL.....                               | 1771         | getpmsg().....          | 1046      |
| 431 | getc().....                               | 984          | getppid().....          | 1047      |
| 432 | getchar().....                            | 987          | getpriority().....      | 1048      |
| 433 | getchar_unlocked.....                     | 985          | getpriority().....      | 775       |
| 434 | getcontext().....                         | 988          | getpriority().....      | 1051      |
| 435 | getcwd().....                             | 990          | getpriority().....      | 775       |
| 436 | getc_unlocked().....                      | 985          | getpriority().....      | 1052      |
| 437 | getdate().....                            | 993          | getprotoent.....        | 775       |
| 438 | getdate_err.....                          | 993          | getprotoent() .....     | 1053      |
| 439 | getegid().....                            | 998          | getpwent.....           | 777       |
| 440 | getenv.....                               | 796          | getpwent().....         | 1054      |
| 441 | getenv().....                             | 999          | getpwnam().....         | 1055      |
| 442 | geteuid().....                            | 1002         | getpwnam_r.....         | 1055      |
| 443 | getgid().....                             | 1003         | getpwuid().....         | 1058      |
| 444 | getgrent.....                             | 769          | getpwuid_r.....         | 1058      |
| 445 | getgrent().....                           | 1004         | getrlimit().....        | 1061      |
| 446 | getgrgid().....                           | 1005         | getrusage().....        | 1064      |
| 447 | getgrgid_r.....                           | 1005         | gets().....             | 1066      |
| 448 | getgrnam().....                           | 1008         | getservbyname.....      | 779       |
| 449 | getgrnam_r.....                           | 1008         | getservbyname() .....   | 1068      |
| 450 | getgroups().....                          | 1011         | getservbyport.....      | 779       |
| 451 | gethostbyaddr().....                      | 1013         | getservbyport().....    | 1069      |
| 452 | gethostbyname.....                        | 1013         | getservent.....         | 779       |
| 453 | gethostbyname().....                      | 1017         | getservent() .....      | 1070      |
| 454 | gethostent.....                           | 771          | getsid().....           | 1071      |
| 455 | gethostent().....                         | 1018         | getsockname() .....     | 1072      |

|     |                       |                                    |                                  |                              |
|-----|-----------------------|------------------------------------|----------------------------------|------------------------------|
| 456 | getsockopt()          | 1073                               | iconv_close()                    | 1103                         |
| 457 | getsubopt()           | 1076                               | iconv_open()                     | 1104                         |
| 458 | gettimeofday()        | 1079                               | IF_                              | 516                          |
| 459 | getuid()              | 1080                               | if_freenameindex()               | 1106                         |
| 460 | getutxent             | 781                                | if_indextoname()                 | 1107                         |
| 461 | getutxent()           | 1082                               | if_nameindex()                   | 1108                         |
| 462 | getutxid              | 781, 1082                          | if_nametoindex()                 | 1109                         |
| 463 | getutxline            | 781, 1082                          | ILL_                             | 516                          |
| 464 | GETVAL                | 1771-1772                          | ilogb()                          | 1110                         |
| 465 | getwc()               | 1083                               | imaxabs()                        | 1111                         |
| 466 | getwchar()            | 1084                               | imaxdiv()                        | 1112                         |
| 467 | getwd                 | 991                                | implementation-defined           | 497                          |
| 468 | getwd()               | 1085                               | IMPLINK_                         | 516                          |
| 469 | GETZCNT               | 1771-1772                          | INADDR_                          | 516                          |
| 470 | glob()                | 1086                               | index()                          | 1113                         |
| 471 | globfree              | 1086                               | inet_addr()                      | 1114                         |
| 472 | GLOB_                 | 516                                | inet_lnaof                       | 1114                         |
| 473 | GLOB_constants        |                                    | inet_lnaof()                     | 1116                         |
| 474 | error returns of glob | 1087                               | inet_makeaddr                    | 1114                         |
| 475 | used in glob          | 1086                               | inet_makeaddr()                  | 1117                         |
| 476 | GLOB_ABORTED          | 1087                               | inet_netof                       | 1114                         |
| 477 | GLOB_APPEND           | 1086-1087                          | inet_netof()                     | 1118                         |
| 478 | GLOB_DOOFFS           | 1086-1087                          | inet_network                     | 1114                         |
| 479 | GLOB_ERR              | 1086-1087                          | inet_network()                   | 1119                         |
| 480 | GLOB_MARK             | 1086                               | inet_ntoa                        | 1114                         |
| 481 | GLOB_NOCHECK          | 1087                               | inet_ntoa()                      | 1120                         |
| 482 | GLOB_NOESCAPE         | 1087                               | inet_ntop()                      | 1121                         |
| 483 | GLOB_NOMATCH          | 1087                               | inet_pton                        | 1121                         |
| 484 | GLOB_NOSORT           | 1087                               | Inf                              | 603, 626, 672, 871, 876      |
| 485 | GLOB_NOSPACE          | 1087                               | INFO                             | 879                          |
| 486 | GMT                   | 2057                               | init                             | 804, 1194                    |
| 487 | gmtime()              | 1090                               | initialize a named semaphore     | 1760                         |
| 488 | gmtime_r              | 1090                               | initialize an unnamed semaphore  | 1757                         |
| 489 | grantpt()             | 1092                               | initializing a mutex             | 1584                         |
| 490 | granularity of clock  | 954                                | initializing condition variables | 1540                         |
| 491 | HALT                  | 879                                | initstate()                      | 1123                         |
| 492 | hcreate               | 1094                               | INIT_PROCESS                     | 781-782                      |
| 493 | hcreate()             | 1094                               | input and output rationale       | 1683                         |
| 494 | hdestroy              | 1094                               | insque()                         | 1125                         |
| 495 | high resolution sleep | 1336                               | international environment        | 1806                         |
| 496 | host name             | 921                                | INT_MAX                          | 1110                         |
| 497 | htonl()               | 1097                               | INT_MIN                          | 597                          |
| 498 | htons                 | 1097                               | IN_                              | 516                          |
| 499 | htons()               | 1098                               | ioctl()                          | 1128                         |
| 500 | HUGE_VAL              | 672, 724, 806, 810, 871            | IOV_                             | 516                          |
| 501 |                       | 1099, 1202, 1206, 1229, 1231, 1233 | IOV_MAX                          | 1680, 1682, 1980, 2165, 2167 |
| 502 |                       | 1235, 1339, 1477, 1736, 1896, 1994 | IP6                              | 502                          |
| 503 |                       | 2128, 2174                         | IPC                              | 541, 1319, 1321, 1324, 1326  |
| 504 | hypot()               | 1099                               |                                  | 1776, 1780, 1846, 1849       |
| 505 | h_errno               | 1093                               | IPC_                             | 516                          |
| 506 | iconv()               | 1101                               |                                  |                              |

## Index

|     |                                |                                          |
|-----|--------------------------------|------------------------------------------|
| 507 | IPC_constants                  |                                          |
| 508 | used in semctl .....           | 1771                                     |
| 509 | used in shmctl .....           | 1844                                     |
| 510 | IPC_CREAT .....                | 1320, 1774, 1848                         |
| 511 | IPC_EXCL .....                 | 1320, 1774                               |
| 512 | IPC_NOWAIT .....               | 1322-1323, 1325-1326, 1777               |
| 513 | IPC_PRIVATE .....              | 1320, 1774, 1848                         |
| 514 | IPC_RMID .....                 | 1318, 1772, 1844                         |
| 515 | IPC_SET .....                  | 1318, 1772, 1844                         |
| 516 | IPC_STAT .....                 | 1318, 1771, 1844                         |
| 517 | IPPORT .....                   | 516                                      |
| 518 | IPPROTO .....                  | 516                                      |
| 519 | IPv4 .....                     | 569                                      |
| 520 | IPv4-compatible address .....  | 570                                      |
| 521 | IPv4-mapped address .....      | 570                                      |
| 522 | IPv6 .....                     | 570                                      |
| 523 | compatibility with IPv4 .....  | 571                                      |
| 524 | interface identification ..... | 571                                      |
| 525 | options .....                  | 571                                      |
| 526 | IPv6 address                   |                                          |
| 527 | anycast .....                  | 570                                      |
| 528 | loopback .....                 | 570                                      |
| 529 | multicast .....                | 570                                      |
| 530 | unicast .....                  | 570                                      |
| 531 | unspecified .....              | 570                                      |
| 532 | IP .....                       | 516                                      |
| 533 | isalnum() .....                | <b>1140</b>                              |
| 534 | isalpha() .....                | <b>1141</b>                              |
| 535 | isascii() .....                | <b>1142</b>                              |
| 536 | isastream() .....              | <b>1143</b>                              |
| 537 | isatty() .....                 | <b>1144</b>                              |
| 538 | isblank() .....                | <b>1145</b>                              |
| 539 | iscntrl() .....                | <b>1146</b>                              |
| 540 | isdigit() .....                | <b>1147</b>                              |
| 541 | isfdtype() .....               | <b>1148</b>                              |
| 542 | isfinite() .....               | <b>1149</b>                              |
| 543 | isgraph() .....                | <b>1150</b>                              |
| 544 | isgreater() .....              | <b>1151</b>                              |
| 545 | isgreaterequal() .....         | <b>1152</b>                              |
| 546 | isinf() .....                  | <b>1153</b>                              |
| 547 | isless() .....                 | <b>1154</b>                              |
| 548 | islessequal() .....            | <b>1155</b>                              |
| 549 | islessgreater() .....          | <b>1156</b>                              |
| 550 | islower() .....                | <b>1157</b>                              |
| 551 | isnan() .....                  | <b>1159</b>                              |
| 552 | isnormal() .....               | <b>1160</b>                              |
| 553 | ISO C standard .....           | 500, 621, 796, 826, 986                  |
| 554 | .....                          | 1241, 1676, 1723, 1806, 1856, 1883, 2026 |
| 555 | isprint() .....                | <b>1161</b>                              |
| 556 | ispunct() .....                | <b>1162</b>                              |
| 557 | isspace() .....                | <b>1163</b>                              |
|     | Issue 4                        |                                          |
|     | changes from .....             | 491                                      |
|     | isunordered() .....            | <b>1164</b>                              |
|     | isupper() .....                | <b>1165</b>                              |
|     | iswalnum() .....               | <b>1166</b>                              |
|     | iswalpha() .....               | <b>1168</b>                              |
|     | iswblank() .....               | <b>1170</b>                              |
|     | iswcntrl() .....               | <b>1171</b>                              |
|     | iswctype() .....               | <b>1173</b>                              |
|     | iswdigit() .....               | <b>1175</b>                              |
|     | iswgraph() .....               | <b>1176</b>                              |
|     | iswlower() .....               | <b>1178</b>                              |
|     | iswprint() .....               | <b>1180</b>                              |
|     | iswpunct() .....               | <b>1182</b>                              |
|     | iswspace() .....               | <b>1184</b>                              |
|     | iswupper() .....               | <b>1186</b>                              |
|     | iswxdigit() .....              | <b>1188</b>                              |
|     | isxdigit() .....               | <b>1189</b>                              |
|     | ITIMER_PROF .....              | 1023                                     |
|     | ITIMER_REAL .....              | 1023                                     |
|     | ITIMER_VIRTUAL .....           | 1023                                     |
|     | I .....                        | 516                                      |
|     | I_ATMARK .....                 | 1134-1135                                |
|     | I_CANPUT .....                 | 1135                                     |
|     | I_CKBAND .....                 | 1135                                     |
|     | I_FDINSERT .....               | 1131                                     |
|     | I_FIND .....                   | 1130                                     |
|     | I_FLUSH .....                  | 1128                                     |
|     | I_FLUSHBAND .....              | 1129                                     |
|     | I_GETBAND .....                | 1135                                     |
|     | I_GETCLTIME .....              | 1135                                     |
|     | I_GETSIG .....                 | 1130                                     |
|     | I_GRDOPT .....                 | 1131, 1680                               |
|     | I_GWROPT .....                 | 1133                                     |
|     | I_LINK .....                   | 1136                                     |
|     | I_LIST .....                   | 1134                                     |
|     | I_LOOK .....                   | 1128                                     |
|     | I_NREAD .....                  | 1131                                     |
|     | I_PEEK .....                   | 1130                                     |
|     | I_PLINK .....                  | 1137                                     |
|     | I_POP .....                    | 1128                                     |
|     | I_PUNLINK .....                | 1137                                     |
|     | I_PUSH .....                   | 1128                                     |
|     | I_RECVFD .....                 | 522, 1134                                |
|     | I_SENDFD .....                 | 1133-1134                                |
|     | I_SETCLTIME .....              | 704, 1135                                |
|     | I_SETSIG .....                 | 1129-1130                                |
|     | I_SRDOPT .....                 | 1130-1131, 1680                          |
|     | I_STR .....                    | 1132                                     |
|     | I_SWROPT .....                 | 1133, 2165                               |
|     | I_UNLINK .....                 | 1136                                     |

|     |                   |                                         |                               |            |
|-----|-------------------|-----------------------------------------|-------------------------------|------------|
| 558 | j0()              | 1190                                    | llrint()                      | 1217       |
| 559 | j1                | 1190                                    | llround()                     | 1218       |
| 560 | jn                | 1190                                    | load ordering                 | 755        |
| 561 | job control       | 804, 1044, 1194, 1811, 1824, 1984, 2100 | localeconv()                  | 1219       |
| 562 | rand48            | 759                                     | localtime()                   | 1223       |
| 563 | rand48()          | 1192                                    | localtime_r                   | 1223       |
| 564 | JST               | 2057                                    | lockf()                       | 1226       |
| 565 | kill()            | 1193                                    | locking                       | 826        |
| 566 | killpg()          | 1196                                    | advisory                      | 827        |
| 567 | l64a              | 593                                     | mandatory                     | 827        |
| 568 | l64a()            | 1197                                    | locking and unlocking a mutex | 1592       |
| 569 | labs()            | 1198                                    | log()                         | 1229       |
| 570 | LANG              | 667                                     | log10()                       | 1231       |
| 571 | last close        | 1840                                    | log1p()                       | 1233       |
| 572 | LASTMARK          | 1135                                    | log2()                        | 1234       |
| 573 | lchown()          | 1199                                    | logb()                        | 1235       |
| 574 | lcong48           | 759                                     | login shell                   | 796        |
| 575 | lcong48()         | 1201                                    | LOGIN_NAME_MAX                | 1025, 1980 |
| 576 | LC_ALL            | 791, 1221, 1346, 1805-1806              | LOGIN_PROCESS                 | 781-782    |
| 577 | LC_COLLATE        | 1086-1087, 1805-1806                    | LOG_                          | 516        |
| 578 |                   | 1924, 1970, 2112, 2143                  | LOG_constants in syslog       | 710        |
| 579 | LC_CTYPE          | 652, 1173, 1249, 1251, 1253             | LOG_ALERT                     | 710        |
| 580 |                   | 1255-1256, 1258, 1260, 1805-1806        | LOG_CONS                      | 711        |
| 581 |                   | 2044-2048, 2107, 2123, 2136, 2145-2146  | LOG_CRIT                      | 710        |
| 582 |                   | 2148-2149                               | LOG_DEBUG                     | 710        |
| 583 | LC_MESSAGES       | 667, 1805-1806, 1930                    | LOG_EMERG                     | 710        |
| 584 | LC_MONETARY       | 1221, 1805-1806, 1933                   | LOG_ERR                       | 710        |
| 585 | LC_NUMERIC        | 765, 898, 932, 966, 975                 | LOG_INFO                      | 710        |
| 586 |                   | 1221, 1805-1806, 1933, 1957, 2128       | LOG_LOCAL                     | 710        |
| 587 | LC_TIME           | 994, 1346, 1805-1806                    | LOG_NDELAY                    | 711        |
| 588 | ldexp()           | 1202                                    | LOG_NOTICE                    | 710        |
| 589 | ldiv()            | 1204                                    | LOG_NOWAIT                    | 711        |
| 590 | legacy            | 497                                     | LOG_ODELAY                    | 711        |
| 591 | LEGACY            | 2083                                    | LOG_PID                       | 711        |
| 592 | lfind             | 1239                                    | LOG_USER                      | 710-711    |
| 593 | lfind()           | 1205                                    | LOG_WARNING                   | 710        |
| 594 | lgamma()          | 1206                                    | longjmp()                     | 1236       |
| 595 | LIFO              | 560                                     | LONG_BIT                      | 760        |
| 596 | LINE_MAX          | 1980                                    | LONG_MAX                      | 1965, 2134 |
| 597 | link to a file    | 1210                                    | LONG_MIN                      | 1965, 2134 |
| 598 | link()            | 1208                                    | rand48                        | 759        |
| 599 | LINK_MAX          | 524, 892, 1208, 1722                    | rand48()                      | 1238       |
| 600 | LIO_              | 514                                     | lsearch()                     | 1239       |
| 601 | lio_listio()      | 1212                                    | lseek()                       | 1241       |
| 602 | LIO_NOP           | 1212                                    | lstat()                       | 1243       |
| 603 | LIO_NOWAIT        | 1212                                    | L_ctermid                     | 738        |
| 604 | LIO_READ          | 1212                                    | l_sysid                       | 827        |
| 605 | LIO_WAIT          | 1212                                    | makecontext()                 | 1245       |
| 606 | LIO_WRITE         | 1212                                    | malloc()                      | 1247       |
| 607 | list directed I/O | 1214                                    | MAN                           | 502        |
| 608 | listen()          | 1215                                    | manipulate signal sets        | 1863       |



## Index

|     |                                 |                             |                        |                 |
|-----|---------------------------------|-----------------------------|------------------------|-----------------|
| 609 | mappings.....                   | 1292                        | MLR.....               | 503             |
| 610 | MAP.....                        | 514, 516                    | mmap().....            | 1287            |
| 611 | MAP_FAILED.....                 | 1292                        | MM_APPL.....           | 878             |
| 612 | MAP_FIXED.....                  | 1288, 1291                  | MM_CONSOLE.....        | 878             |
| 613 | MAP_PRIVATE.....                | 887, 1288, 1292, 1296, 1328 | MM_ERROR.....          | 879-880         |
| 614 | MAP_SHARED.....                 | 890, 1288-1289              | mm_FIRM.....           | 878             |
| 615 | MAXPATHLEN.....                 | 747                         | MM_HALT.....           | 879             |
| 616 | MAX_CANON.....                  | 892                         | MM_HARD.....           | 878             |
| 617 | MAX_INPUT.....                  | 892                         | MM_INFO.....           | 879             |
| 618 | may.....                        | 497                         | MM_NOCON.....          | 879             |
| 619 | mblen().....                    | 1249                        | MM_NOMSG.....          | 879             |
| 620 | mbrlen().....                   | 1251                        | MM_NOSEV.....          | 879             |
| 621 | mbrtowc().....                  | 1253                        | MM_NOTOK.....          | 879             |
| 622 | mbsinit().....                  | 1255                        | MM_NRECOV.....         | 878             |
| 623 | mbsrtowcs().....                | 1256                        | MM_NULLMC.....         | 878             |
| 624 | mbstowcs().....                 | 1258                        | MM_OK.....             | 879             |
| 625 | mbtowc().....                   | 1260                        | MM_OPSYS.....          | 878             |
| 626 | MB_CUR_MAX.....                 | 1249, 1251, 1253, 1260      | MM_PRINT.....          | 878, 880        |
| 627 | .....                           | 2107, 2146                  | MM_RECOVER.....        | 878             |
| 628 | MCL.....                        | 514                         | MM_SOFT.....           | 878             |
| 629 | MCL_CURRENT.....                | 1285                        | MM_UTIL.....           | 878             |
| 630 | MCL_FUTURE.....                 | 1285                        | MM_WARNING.....        | 879             |
| 631 | memccpy().....                  | 1262                        | modf().....            | 1294            |
| 632 | memchr().....                   | 1263                        | MON.....               | 503             |
| 633 | memcmp().....                   | 1264                        | MORECTL.....           | 1029            |
| 634 | memcpy().....                   | 1265                        | MOREDATA.....          | 1029            |
| 635 | MEMLOCK_FUTURE.....             | 1292                        | MPR.....               | 503             |
| 636 | memmove().....                  | 1266                        | mprotect().....        | 1296            |
| 637 | memory management.....          | 545                         | mq_close().....        | 1298            |
| 638 | memory protection option.....   | 1291                        | mq_getattr().....      | 1299            |
| 639 | memset().....                   | 1267                        | mq_notify().....       | 1301            |
| 640 | message catalog descriptor..... | 791, 796, 801               | mq_open().....         | 1303            |
| 641 | message parts.....              | 540                         | MQ_OPEN_MAX.....       | 1980            |
| 642 | message priority.....           | 539                         | MQ_PRIO_MAX.....       | 1309-1310, 1980 |
| 643 | high-priority.....              | 539                         | mq_receive().....      | 1306            |
| 644 | normal.....                     | 539                         | mq_send().....         | 1309            |
| 645 | priority.....                   | 539                         | mq_setattr().....      | 1311            |
| 646 | message-discard mode.....       | 1680                        | mq_timedreceive.....   | 1306            |
| 647 | message-nondiscard mode.....    | 1680                        | mq_timedreceive()..... | 1313            |
| 648 | MET.....                        | 2057                        | mq_timedsend.....      | 1309            |
| 649 | MF.....                         | 502                         | mq_timedsend().....    | 1314            |
| 650 | MINSIGSTKSZ.....                | 1860                        | mq_unlink().....       | 1315            |
| 651 | mkdir().....                    | 1268                        | mrnd48.....            | 759             |
| 652 | mkfifo().....                   | 1271                        | mrnd48().....          | 1317            |
| 653 | mknod().....                    | 1274                        | MSE.....               | 492             |
| 654 | mkstemp().....                  | 1277                        | MSG.....               | 503, 516        |
| 655 | mktemp().....                   | 1279                        | msgctl().....          | 1318            |
| 656 | mktime().....                   | 1281                        | msgget().....          | 1320            |
| 657 | ML.....                         | 502                         | msgrcv().....          | 1322            |
| 658 | mlock().....                    | 1283                        | msgsnd().....          | 1325            |
| 659 | mlockall().....                 | 1285                        | MSGVERB.....           | 879-880         |

|     |                                      |                                          |                                  |                                          |
|-----|--------------------------------------|------------------------------------------|----------------------------------|------------------------------------------|
| 660 | MSG_.....                            | 516                                      | nohup utility.....               | 797                                      |
| 661 | MSG_ANY.....                         | 1028                                     | non-local jumps.....             | 1883                                     |
| 662 | MSG_BAND.....                        | 1028, 1662                               | non-volatile storage.....        | 950                                      |
| 663 | MSG_HIPRI.....                       | 1028, 1662                               | nrand48.....                     | 759                                      |
| 664 | MSG_NOERROR.....                     | 1322-1323                                | nrand48().....                   | <b>1348</b>                              |
| 665 | msg_perm.....                        | 541                                      | ntohl.....                       | 1097                                     |
| 666 | msqid.....                           | 541                                      | ntohl().....                     | <b>1349</b>                              |
| 667 | MST.....                             | 2057                                     | ntohs.....                       | 1097                                     |
| 668 | msync().....                         | <b>1328</b>                              | ntohs().....                     | <b>1350</b>                              |
| 669 | MS.....                              | 514, 516                                 | NULL.....                        | 683, 715, 738, 745, 752, 757, 1292       |
| 670 | MS_ASYNC.....                        | 1289, 1328                               | NUM_EMPL.....                    | 1095                                     |
| 671 | MS_INVALIDATE.....                   | 1328-1329                                | NZERO.....                       | 1048, 1344                               |
| 672 | MS_SYNC.....                         | 1289, 1328                               | OB.....                          | <b>503</b>                               |
| 673 | multicast.....                       | 570                                      | obsolescent.....                 | 675                                      |
| 674 | munlock.....                         | 1283                                     | OF.....                          | <b>503</b>                               |
| 675 | munlock().....                       | <b>1331</b>                              | OH.....                          | <b>503</b>                               |
| 676 | munlockall.....                      | 1285                                     | OLD_TIME.....                    | 781-782                                  |
| 677 | munlockall().....                    | <b>1332</b>                              | open a file.....                 | 1355                                     |
| 678 | munmap().....                        | <b>1333</b>                              | open a named semaphore.....      | 1760                                     |
| 679 | mutex attributes.....                | 1599                                     | open a shared memory object..... | 1837                                     |
| 680 | mutex initialization attributes..... | 1598                                     | open().....                      | <b>1351</b>                              |
| 681 | mutex performance.....               | 1599                                     | opendir().....                   | <b>1359</b>                              |
| 682 | MUXID_ALL.....                       | 1136-1137                                | openlog.....                     | 710                                      |
| 683 | MUXID_R.....                         | 516                                      | openlog().....                   | <b>1362</b>                              |
| 684 | M.....                               | 516                                      | OPEN_MAX.....                    | 524, 667, 762, 777, 826, 884             |
| 685 | name information.....                | 1032                                     | .....                            | 926, 961, 1005, 1008, 1025, 1058, 1104   |
| 686 | name space.....                      | <b>513</b>                               | .....                            | 1342, 1353, 1359, 1371, 1397, 1400, 1980 |
| 687 | NAME_MAX.....                        | 524, 600, 667, 682, 684, 688             | .....                            | 2039                                     |
| 688 | .....                                | 794, 814, 832, 884, 892, 926, 947, 956   | optarg.....                      | 1037, 1363                               |
| 689 | .....                                | 1199, 1208, 1243, 1268, 1275, 1304, 1315 | opterr.....                      | 1037, 1363                               |
| 690 | .....                                | 1342, 1353, 1359, 1475, 1687, 1691, 1697 | optind.....                      | 1037, 1363                               |
| 691 | .....                                | 1722, 1731, 1760, 1768, 1837, 1840, 1912 | option                           |                                          |
| 692 | .....                                | 1977, 2050, 2071, 2080, 2083             | ADV.....                         | 501                                      |
| 693 | NaN.....                             | 603, 626, 630, 632, 672, 722, 724        | AIO.....                         | 501                                      |
| 694 | .....                                | 786, 806, 811, 871, 876, 930, 1099, 1190 | BAR.....                         | 501                                      |
| 695 | .....                                | 1202, 1206, 1229, 1231, 1294, 1339, 1477 | BE.....                          | 501                                      |
| 696 | .....                                | 1894, 1896, 1906, 1994, 1996, 2174       | CD.....                          | 501                                      |
| 697 | nan().....                           | <b>1335</b>                              | CPT.....                         | 501                                      |
| 698 | nanosleep().....                     | <b>1336</b>                              | CS.....                          | 502                                      |
| 699 | NDEBUG.....                          | 520, 629                                 | FD.....                          | 502                                      |
| 700 | nearbyint().....                     | <b>1338</b>                              | FR.....                          | 502                                      |
| 701 | network interfaces.....              | 563                                      | FSC.....                         | 502                                      |
| 702 | NEW_TIME.....                        | 781-782                                  | IP6.....                         | 502                                      |
| 703 | nextafter().....                     | <b>1339</b>                              | MF.....                          | 502                                      |
| 704 | nftw().....                          | <b>1341</b>                              | ML.....                          | 502                                      |
| 705 | NGROUPS_MAX.....                     | 1012, 1980                               | MLR.....                         | 503                                      |
| 706 | nice().....                          | <b>1344</b>                              | MON.....                         | 503                                      |
| 707 | NLSPATH.....                         | 667-668                                  | MPR.....                         | 503                                      |
| 708 | NL_ARGMAX.....                       | 898, 932, 966, 975                       | MSG.....                         | 503                                      |
| 709 | NL_CAT_LOCALE.....                   | 667                                      | PIO.....                         | 504                                      |
| 710 | nl_langinfo().....                   | <b>1346</b>                              | PS.....                          | 504                                      |

## Index

|     |                        |                                         |                                 |                                          |
|-----|------------------------|-----------------------------------------|---------------------------------|------------------------------------------|
| 711 | RTS                    | 504                                     | PAGESIZE                        | 545, 1283, 1329, 1333, 1493, 1982        |
| 712 | SD                     | 504                                     | PAGE_SIZE                       | 1982                                     |
| 713 | SEM                    | 504                                     | PATH                            | 715                                      |
| 714 | SHM                    | 504                                     | PATH environment variable       | 798                                      |
| 715 | SIO                    | 504                                     | pathconf                        | 892                                      |
| 716 | SPI                    | 504                                     | pathconf()                      | 1364                                     |
| 717 | SPN                    | 505                                     | PATH_MAX                        | 524, 600, 682, 684, 688, 794             |
| 718 | SS                     | 505                                     |                                 | 814, 832, 884, 892, 926, 947, 956, 991   |
| 719 | TCT                    | 505                                     |                                 | 1085, 1199, 1208, 1243, 1268, 1275, 1304 |
| 720 | TEF                    | 506                                     |                                 | 1342, 1353, 1359, 1475, 1691, 1722, 1731 |
| 721 | THR                    | 505                                     |                                 | 1760, 1837, 1912, 1977, 1985, 2050, 2071 |
| 722 | TMO                    | 505                                     |                                 | 2080, 2083                               |
| 723 | TMR                    | 505                                     | pause()                         | 1365                                     |
| 724 | TPI                    | 505                                     | pclose()                        | 1367                                     |
| 725 | TPP                    | 506                                     | per-process timers              | 2033                                     |
| 726 | TPS                    | 506                                     | perror()                        | 1369                                     |
| 727 | TRC                    | 506                                     | persistent connection (I_PLINK) | 1137                                     |
| 728 | TRI                    | 506                                     | PF_                             | 516                                      |
| 729 | TRL                    | 506                                     | physical write                  | 950                                      |
| 730 | TSA                    | 506                                     | PIO                             | 504                                      |
| 731 | TSF                    | 506                                     | pipe                            | 889, 1355, 2168                          |
| 732 | TSH                    | 507                                     | pipe()                          | 1371                                     |
| 733 | TSP                    | 507                                     | PIPE_BUF                        | 892, 2164, 2168                          |
| 734 | TSS                    | 507                                     | PIPE_MAX                        | 2170                                     |
| 735 | TYM                    | 507                                     | plain characters                | 1932                                     |
| 736 | UP                     | 507                                     | POLL                            | 516                                      |
| 737 | XSR                    | 508                                     | poll()                          | 1373                                     |
| 738 | optopt                 | 1037, 1040, 1363                        | POLLERR                         | 1373                                     |
| 739 | optstring              | 1040                                    | POLLHUP                         | 1373                                     |
| 740 | orphaned process group | 804                                     | POLLIN                          | 1373                                     |
| 741 | O_ constants           |                                         | POLLNVAL                        | 1373                                     |
| 742 | used in open()         | 1351                                    | POLLOUT                         | 1373                                     |
| 743 | O_ACCMODE              | 823                                     | POLLPRI                         | 1373                                     |
| 744 | O_APPEND               | 543, 618, 743, 836, 1351, 2163          | POLLRDBAND                      | 1373                                     |
| 745 | O_CREAT                | 729, 1303-1304, 1315, 1351-1353         | POLLRDNORM                      | 1373                                     |
| 746 |                        | 1754, 1759, 1836-1838                   | POLLWRBAND                      | 1373                                     |
| 747 | O_DSYNC                | 610, 1351-1352, 1681, 2165              | POLLWRNORM                      | 1373                                     |
| 748 | O_EXCL                 | 1304, 1351, 1759, 1836-1837             | POLL_                           | 516                                      |
| 749 | O_NDELAY               | 1356, 2169                              | popen()                         | 1377                                     |
| 750 | O_NOCTTY               | 1352, 1356                              | portability                     | 501                                      |
| 751 | O_NONBLOCK             | 523, 704, 821, 853, 857, 864            | POSIX                           | 765                                      |
| 752 |                        | 910, 914, 940, 1029, 1132, 1134, 1304   | POSIX.1 symbols                 | 512                                      |
| 753 |                        | 1306, 1309, 1311, 1352-1353, 1371, 1374 | POSIX_ALLOC_SIZE_MIN            | 892                                      |
| 754 |                        | 1663, 1679, 2164, 2168                  | posix_fadvise()                 | 1380                                     |
| 755 | O_RDONLY               | 747, 1303, 1351-1352, 1356, 1836        | POSIX_FADV_DONTNEED             | 1380                                     |
| 756 |                        | 1838                                    | POSIX_FADV_NOREUSE              | 1380                                     |
| 757 | O_RDWR                 | 1226, 1303, 1351-1355, 1836, 1838       | POSIX_FADV_NORMAL               | 1380                                     |
| 758 | O_RSYNC                | 1352, 1681                              | POSIX_FADV_RANDOM               | 1380                                     |
| 759 | O_SYNC                 | 610, 1352, 1681, 2165                   | POSIX_FADV_SEQUENTIAL           | 1380                                     |
| 760 | O_TRUNC                | 729, 1352, 1354-1355, 1837-1838         | POSIX_FADV_WILLNEED             | 1380                                     |
| 761 | O_WRONLY               | 729, 1226, 1303, 1351-1354, 1356        | posix_fallocate()               | 1382                                     |

|     |                                     |                  |                                        |                             |
|-----|-------------------------------------|------------------|----------------------------------------|-----------------------------|
| 762 | posix_madvise()                     | 1384             | posix_trace_attr_getinherited()        | 1431                        |
| 763 | POSIX_MADV_DONTNEED                 | 1384             | posix_trace_attr_getlogsize()          | 1434                        |
| 764 | POSIX_MADV_NORMAL                   | 1384             | posix_trace_attr_init()                | 1437                        |
| 765 | POSIX_MADV_RANDOM                   | 1384             | posix_trace_clear()                    | 1438                        |
| 766 | POSIX_MADV_SEQUENTIAL               | 1384             | posix_trace_close()                    | 1440                        |
| 767 | POSIX_MADV_WILLNEED                 | 1384             | posix_trace_create()                   | 1442                        |
| 768 | posix_memalign()                    | 1388             | posix_trace_event()                    | 1446                        |
| 769 | posix_mem_offset()                  | 1386             | posix_trace_eventid_equal()            | 1448                        |
| 770 | POSIX_REC_INCR_XFER_SIZE            | 892              | posix_trace_eventset_add()             | 1450                        |
| 771 | POSIX_REC_MAX_XFER_SIZE             | 892              | posix_trace_eventtypelist_getnext_id() | 1452                        |
| 772 | POSIX_REC_MIN_XFER_SIZE             | 892              | posix_trace_flush()                    | 1454                        |
| 773 | POSIX_REC_XFER_ALIGN                | 892              | posix_trace_getnext_event()            | 1460                        |
| 774 | posix_spawn()                       | 1389             | posix_trace_get_attr()                 | 1455                        |
| 775 | posix_spawnattr_destroy()           | 1405             | posix_trace_get_filter()               | 1457                        |
| 776 | posix_spawnattr_getflags()          | 1407             | posix_trace_get_status()               | 1459                        |
| 777 | posix_spawnattr_getpgroup()         | 1409             | posix_trace_open()                     | 1463                        |
| 778 | posix_spawnattr_getschedparam()     | 1411             | posix_trace_rewind()                   | 1464                        |
| 779 | posix_spawnattr_getschedpolicy()    | 1413             | posix_trace_set_filter()               | 1465                        |
| 780 | posix_spawnattr_getsigdefault()     | 1415             | posix_trace_shutdown()                 | 1466                        |
| 781 | posix_spawnattr_getsigmask()        | 1417             | posix_trace_start()                    | 1467                        |
| 782 | posix_spawnattr_init                | 1405             | posix_trace_timedgetnext_event()       | 1469                        |
| 783 | posix_spawnattr_init()              | 1419             | posix_trace_trid_eventid_open()        | 1470                        |
| 784 | posix_spawnattr_setflags            | 1407             | posix_trace_trygetnext_event()         | 1471                        |
| 785 | posix_spawnattr_setflags()          | 1420             | POSIX_TYPED_MEM_ALLOCATE               | 1287-1288                   |
| 786 | posix_spawnattr_setpgroup           | 1409             | .....                                  | 1386, 1472, 1474            |
| 787 | posix_spawnattr_setpgroup()         | 1421             | POSIX_TYPED_MEM_ALLOCATE_CONTIG        | .....                       |
| 788 | posix_spawnattr_setschedparam       | 1411             | .....                                  | 1287-1288, 1386, 1472, 1474 |
| 789 | posix_spawnattr_setschedparam()     | 1422             | posix_typed_mem_get_info()             | 1472                        |
| 790 | posix_spawnattr_setschedpolicy      | 1413             | POSIX_TYPED_MEM_MAP_ALLOCATABLE        | 1333                        |
| 791 | posix_spawnattr_setschedpolicy()    | 1423             | .....                                  | 1474                        |
| 792 | posix_spawnattr_setsigdefault()     | 1424             | posix_typed_mem_open()                 | 1474                        |
| 793 | posix_spawnattr_setsigmask          | 1417             | pow()                                  | 1477                        |
| 794 | posix_spawnattr_setsigmask()        | 1425             | pread                                  | 1679                        |
| 795 | posix_spawnnp                       | 1389             | pread()                                | 1479                        |
| 796 | posix_spawnnp()                     | 1426             | predefined stream                      |                             |
| 797 | posix_spawn_file_actions_addclose() | 1397             | standard error                         | 537                         |
| 798 | posix_spawn_file_actions_adddup2()  | 1400             | standard input                         | 537                         |
| 799 | posix_spawn_file_actions_addopen    | 1397             | standard output                        | 537                         |
| 800 | posix_spawn_file_actions_addopen()  | 1402             | preempted thread                       | 1545                        |
| 801 | posix_spawn_file_actions_destroy()  | 1403             | printf                                 | 898                         |
| 802 | posix_spawn_file_actions_init       | 1403             | printf()                               | 1480                        |
| 803 | posix_spawn_file_actions_init()     | 1404             | priority                               | 539                         |
| 804 | POSIX_SPAWN_RESETEIDS               | 1390, 1407       | PRIO_                                  | 516                         |
| 805 | POSIX_SPAWN_SETPGROUP               | 1390, 1407, 1409 | PRIO_INHERIT                           | 1595                        |
| 806 | POSIX_SPAWN_SETSCHEDPARAM           | 1407, 1411       | PRIO_PGRP                              | 1048                        |
| 807 | POSIX_SPAWN_SETSCHEDULER            | 1390, 1407       | PRIO_PROCESS                           | 1048                        |
| 808 | .....                               | 1411, 1413       | PRIO_USER                              | 1048                        |
| 809 | POSIX_SPAWN_SETSIGDEF               | 1407             | process                                |                             |
| 810 | POSIX_SPAWN_SETSIGMASK              | 1407, 1417       | concurrent execution                   | 889                         |
| 811 | posix_trace_attr_destroy()          | 1427             | setting real and effective user IDs    | 1820                        |
| 812 | posix_trace_attr_getclockres()      | 1429             | single-threaded                        | 889                         |

## Index

|     |                                                 |                        |                                        |                 |
|-----|-------------------------------------------------|------------------------|----------------------------------------|-----------------|
| 813 | process creation .....                          | 889                    | pthread_barrierattr_destroy() .....    | 1522            |
| 814 | process group                                   |                        | pthread_barrierattr_getpshared() ..... | 1524            |
| 815 | orphaned .....                                  | 804                    | pthread_barrierattr_init .....         | 1522            |
| 816 | process group ID.....                           | 1044, 1812, 1824       | pthread_barrierattr_init().....        | 1526            |
| 817 | process ID, 1.....                              | 804                    | pthread_barrierattr_setpshared .....   | 1524            |
| 818 | process lifetime .....                          | 1195                   | pthread_barrierattr_setpshared().....  | 1527            |
| 819 | process scheduling.....                         | 546                    | pthread_barrier_destroy().....         | 1517            |
| 820 | process shared memory .....                     | 1599                   | pthread_barrier_init.....              | 1517            |
| 821 | process synchronization .....                   | 1599                   | pthread_barrier_init() .....           | 1519            |
| 822 | process termination.....                        | 803                    | PTHREAD_BARRIER_SERIAL_THREAD .....    | 1520            |
| 823 | PROT_.....                                      | 514, 516               | pthread_barrier_wait() .....           | 1520            |
| 824 | PROT_EXEC.....                                  | 1287, 1296             | pthread_cancel() .....                 | 1528            |
| 825 | PROT_NONE.....                                  | 1287, 1296             | PTHREAD_CANCELED.....                  | 560, 1565       |
| 826 | PROT_READ.....                                  | 1287, 1296             | PTHREAD_CANCEL_ASYNCHRONOUS .....      | 557             |
| 827 | PROT_WRITE.....                                 | 1287, 1289, 1291, 1296 | .....                                  | 1640            |
| 828 | PS .....                                        | 504                    | PTHREAD_CANCEL_DEFERRED .....          | 557             |
| 829 | pselect().....                                  | 1481                   | .....                                  | 1543, 1640      |
| 830 | pseudo-random sequence generation functions.... |                        | PTHREAD_CANCEL_DISABLE.....            | 557, 1640       |
| 831 | .....                                           | 1676                   | PTHREAD_CANCEL_ENABLE.....             | 557, 1640       |
| 832 | PST.....                                        | 2057                   | pthread_cleanup_pop() .....            | 1530            |
| 833 | PTHREAD.....                                    | 514                    | pthread_cleanup_push.....              | 1530            |
| 834 | pthread_atfork().....                           | 1486                   | pthread_condattr_destroy() .....       | 1549            |
| 835 | pthread_attr_destroy() .....                    | 1488                   | pthread_condattr_getclock().....       | 1551            |
| 836 | pthread_attr_getdetachstate().....              | 1491                   | pthread_condattr_getpshared().....     | 1553            |
| 837 | pthread_attr_getguardsize().....                | 1493                   | pthread_condattr_init.....             | 1549            |
| 838 | pthread_attr_getinheritsched().....             | 1495                   | pthread_condattr_init() .....          | 1555            |
| 839 | pthread_attr_getschedparam() .....              | 1497                   | pthread_condattr_setclock.....         | 1551            |
| 840 | pthread_attr_getschedpolicy().....              | 1499                   | pthread_condattr_setclock() .....      | 1556            |
| 841 | pthread_attr_getscope() .....                   | 1501                   | pthread_condattr_setpshared.....       | 1553            |
| 842 | pthread_attr_getstack() .....                   | 1503                   | pthread_condattr_setpshared() .....    | 1557            |
| 843 | pthread_attr_getstackaddr() .....               | 1505                   | pthread_cond_broadcast().....          | 1535            |
| 844 | pthread_attr_getstacksize() .....               | 1506                   | pthread_cond_destroy().....            | 1538            |
| 845 | pthread_attr_init.....                          | 1488                   | pthread_cond_init.....                 | 1538            |
| 846 | pthread_attr_init() .....                       | 1507                   | pthread_cond_init().....               | 1541            |
| 847 | pthread_attr_setdetachstate.....                | 1491                   | PTHREAD_COND_INITIALIZER.....          | 1538, 1541      |
| 848 | pthread_attr_setdetachstate() .....             | 1508                   | pthread_cond_signal .....              | 1535            |
| 849 | pthread_attr_setguardsize.....                  | 1493                   | pthread_cond_signal().....             | 1542            |
| 850 | pthread_attr_setguardsize() .....               | 1509                   | pthread_cond_timedwait().....          | 1543            |
| 851 | pthread_attr_setinheritsched.....               | 1495                   | pthread_cond_wait .....                | 1543            |
| 852 | pthread_attr_setinheritsched() .....            | 1510                   | pthread_cond_wait().....               | 1548            |
| 853 | pthread_attr_setschedparam .....                | 1497                   | pthread_create().....                  | 1558            |
| 854 | pthread_attr_setschedparam().....               | 1511                   | PTHREAD_CREATE_DETACHED.....           | 530, 1491       |
| 855 | pthread_attr_setschedpolicy.....                | 1499                   | PTHREAD_CREATE_JOINABLE ..             | 530, 1491, 1574 |
| 856 | pthread_attr_setschedpolicy() .....             | 1512                   | PTHREAD_DESTRUCTOR_ITERATIONS.....     |                 |
| 857 | pthread_attr_setscope .....                     | 1501                   | .....                                  | 1576, 1983      |
| 858 | pthread_attr_setscope().....                    | 1513                   | pthread_detach().....                  | 1561            |
| 859 | pthread_attr_setstack().....                    | 1514                   | pthread_equal() .....                  | 1563            |
| 860 | pthread_attr_setstackaddr .....                 | 1505                   | pthread_exit() .....                   | 1564            |
| 861 | pthread_attr_setstackaddr() .....               | 1515                   | PTHREAD_EXPLICIT_SCHED .....           | 1495            |
| 862 | pthread_attr_setstacksize .....                 | 1506                   | pthread_getconcurrency() .....         | 1566            |
| 863 | pthread_attr_setstacksize().....                | 1516                   | pthread_getcpuclid() .....             | 1568            |

|     |                                    |                        |                                 |                 |
|-----|------------------------------------|------------------------|---------------------------------|-----------------|
| 864 | pthread_getschedparam()            | 1569                   | pthread_rwlockattr_init         | 1633            |
| 865 | pthread_getspecific()              | 1571                   | pthread_rwlockattr_init()       | 1637            |
| 866 | PTHREAD_INHERIT_SCHED              | 1495                   | pthread_rwlockattr_setpshared   | 1635            |
| 867 | pthread_join()                     | 1573                   | pthread_rwlockattr_setpshared() | 1638            |
| 868 | PTHREAD_KEYS_MAX                   | 1576, 1983             | pthread_rwlock_destroy()        | 1618            |
| 869 | pthread_key_create()               | 1576                   | pthread_rwlock_init             | 1618            |
| 870 | pthread_key_delete()               | 1580                   | pthread_rwlock_init()           | 1620            |
| 871 | pthread_kill()                     | 1582                   | pthread_rwlock_rdlock()         | 1621            |
| 872 | pthread_mutexattr_destroy()        | 1598                   | pthread_rwlock_timedrdlock()    | 1623            |
| 873 | pthread_mutexattr_getprioceiling() | 1603                   | pthread_rwlock_timedwrlock()    | 1625            |
| 874 | pthread_mutexattr_getprotocol()    | 1605                   | pthread_rwlock_tryrdlock        | 1621            |
| 875 | pthread_mutexattr_getpshared()     | 1607                   | pthread_rwlock_tryrdlock()      | 1627            |
| 876 | pthread_mutexattr_gettype()        | 1609                   | pthread_rwlock_trywrlock()      | 1628            |
| 877 | pthread_mutexattr_init             | 1598                   | pthread_rwlock_unlock()         | 1630            |
| 878 | pthread_mutexattr_init()           | 1611                   | pthread_rwlock_wrlock           | 1628            |
| 879 | pthread_mutexattr_setprioceiling   | 1603                   | pthread_rwlock_wrlock()         | 1632            |
| 880 | pthread_mutexattr_setprioceiling() | 1612                   | PTHREAD_SCOPE_PROCESS           | 555, 1501       |
| 881 | pthread_mutexattr_setprotocol      | 1605                   | PTHREAD_SCOPE_SYSTEM            | 555, 1501       |
| 882 | pthread_mutexattr_setprotocol()    | 1613                   | pthread_self()                  | 1639            |
| 883 | pthread_mutexattr_setpshared       | 1607                   | pthread_setcancelstate()        | 1640            |
| 884 | pthread_mutexattr_setpshared()     | 1614                   | pthread_setcanceltype           | 1640            |
| 885 | pthread_mutexattr_settype          | 1609                   | pthread_setconcurrency          | 1566            |
| 886 | pthread_mutexattr_settype()        | 1615                   | pthread_setconcurrency()        | 1642            |
| 887 | PTHREAD_MUTEX_DEFAULT              | 1591, 1609             | pthread_setschedparam           | 1569            |
| 888 | pthread_mutex_destroy()            | 1583                   | pthread_setschedparam()         | 1643            |
| 889 | PTHREAD_MUTEX_ERRORCHECK           | 1591, 1609             | pthread_setspecific             | 1571            |
| 890 | pthread_mutex_getprioceiling()     | 1588                   | pthread_setspecific()           | 1644            |
| 891 | pthread_mutex_init                 | 1583                   | pthread_sigmask()               | 1645            |
| 892 | pthread_mutex_init()               | 1590                   | pthread_spin_destroy()          | 1647            |
| 893 | PTHREAD_MUTEX_INITIALIZER          | 1583, 1590             | pthread_spin_init               | 1647            |
| 894 | pthread_mutex_lock()               | 1591                   | pthread_spin_init()             | 1649            |
| 895 | PTHREAD_MUTEX_NORMAL               | 1591, 1609             | pthread_spin_lock()             | 1650            |
| 896 | PTHREAD_MUTEX_RECURSIVE            | 1591                   | pthread_spin_trylock            | 1650            |
| 897 |                                    | 1609-1610              | pthread_spin_trylock()          | 1652            |
| 898 | pthread_mutex_setprioceiling       | 1588                   | pthread_spin_unlock()           | 1653            |
| 899 | pthread_mutex_setprioceiling()     | 1594                   | PTHREAD_STACK_MIN               | 1505-1506, 1983 |
| 900 | pthread_mutex_timedlock()          | 1595                   | pthread_testcancel              | 1640            |
| 901 | pthread_mutex_trylock              | 1591                   | pthread_testcancel()            | 1654            |
| 902 | pthread_mutex_trylock()            | 1597                   | PTHREAD_THREADS_MAX             | 1558, 1983      |
| 903 | pthread_mutex_unlock               | 1591, 1597             | ptsname()                       | 1655            |
| 904 | pthread_once()                     | 1616                   | putc()                          | 1656            |
| 905 | PTHREAD_ONCE_INIT                  | 1616                   | putchar()                       | 1658            |
| 906 | PTHREAD_PRIO_INHERIT               | 1605                   | putchar_unlocked                | 985             |
| 907 | PTHREAD_PRIO_NONE                  | 1605                   | putchar_unlocked()              | 1659            |
| 908 | PTHREAD_PRIO_PROTECT               | 1592, 1605             | putc_unlocked                   | 985             |
| 909 | PTHREAD_PROCESS_PRIVATE            | 1524, 1553             | putc_unlocked()                 | 1657            |
| 910 |                                    | 1599, 1607, 1635, 1647 | putenv()                        | 1660            |
| 911 | PTHREAD_PROCESS_SHARED             | 1524, 1553             | putmsg()                        | 1662            |
| 912 |                                    | 1599, 1607, 1635, 1647 | putpmsg                         | 1662            |
| 913 | pthread_rwlockattr_destroy()       | 1633                   | puts()                          | 1666            |
| 914 | pthread_rwlockattr_getpshared()    | 1635                   | pututxline                      | 781             |

## Index

|     |                                |                                          |                          |           |
|-----|--------------------------------|------------------------------------------|--------------------------|-----------|
| 915 | pututxline()                   | 1668                                     | REG_BADPAT               | 1709      |
| 916 | putwc()                        | 1669                                     | REG_BADRPT               | 1710      |
| 917 | putwchar()                     | 1670                                     | REG_EBRACE               | 1709      |
| 918 | pwrite                         | 2163                                     | REG_EBRACK               | 1709      |
| 919 | pwrite()                       | 1671                                     | REG_ECOLLATE             | 1709      |
| 920 | P_ALL                          | 2104                                     | REG_ECTYPE               | 1709      |
| 921 | P_PGID                         | 2104                                     | REG_EESCAPE              | 1709      |
| 922 | P_PID                          | 2104                                     | REG_EPAREN               | 1709      |
| 923 | qsort()                        | 1672                                     | REG_ERANGE               | 1709      |
| 924 | queue a signal to a process    | 1881                                     | REG_ESPACE               | 1709      |
| 925 | raise()                        | 1673                                     | REG_ESUBREG              | 1709      |
| 926 | rand()                         | 1675                                     | REG_EXTENDED             | 1707      |
| 927 | random                         | 1123                                     | REG_ICASE                | 1707      |
| 928 | random()                       | 1678                                     | REG_NEWLINE              | 1707      |
| 929 | RAND_MAX                       | 1675                                     | REG_NOMATCH              | 1709      |
| 930 | rand_r                         | 1675                                     | REG_NOSUB                | 1707      |
| 931 | read from a file               | 1683                                     | REG_NOTBOL               | 1708      |
| 932 | read()                         | 1679                                     | REG_NOTEOL               | 1708      |
| 933 | readdir()                      | 1687                                     | remainder()              | 1715      |
| 934 | readdir_r                      | 1687                                     | remove a directory       | 1732      |
| 935 | readlink()                     | 1691                                     | remove directory entries | 2073      |
| 936 | readv                          | 1679                                     | remove()                 | 1716      |
| 937 | readv()                        | 1694                                     | remque                   | 1125      |
| 938 | real user ID                   | 601, 1194                                | remque()                 | 1718      |
| 939 | realloc()                      | 1695                                     | remquo()                 | 1719      |
| 940 | realpath()                     | 1697                                     | rename a file            | 1723      |
| 941 | REALTIME                       | 607, 609-610, 612, 615-616               | rename()                 | 1721      |
| 942 |                                | 618, 696-697, 831, 1212, 1283, 1285      | rewind()                 | 1725      |
| 943 |                                | 1298-1299, 1301, 1303, 1306, 1309, 1311  | rewinddir()              | 1726      |
| 944 |                                | 1313-1315, 1336, 1389, 1397, 1400        | RE_DUP_MAX               | 1983      |
| 945 |                                | 1402-1404, 1407, 1409, 1411, 1413, 1417  | rindex()                 | 1728      |
| 946 |                                | 1419-1423, 1425, 1568, 1740-1744, 1747   | rint()                   | 1729      |
| 947 |                                | 1754-1757, 1759, 1762, 1764, 1766, 1768  | RLIMIT_                  | 516       |
| 948 |                                | 1770, 1836, 1840, 1880, 1887, 1893, 2028 | RLIMIT_AS                | 1062      |
| 949 |                                | 2031-2032                                | RLIMIT_CORE              | 1061      |
| 950 | REALTIME THREADS               | 1495, 1499, 1501, 1510                   | RLIMIT_CPU               | 1061      |
| 951 |                                | 1512-1513, 1569, 1588, 1594, 1603, 1605  | RLIMIT_DATA              | 1061      |
| 952 |                                | 1612-1613, 1643                          | RLIMIT_FSIZE             | 1061      |
| 953 | recv()                         | 1699                                     | RLIMIT_NOFILE            | 1061-1062 |
| 954 | recvfrom()                     | 1701                                     | RLIMIT_STACK             | 1061      |
| 955 | recvmsg()                      | 1704                                     | RLIM_                    | 516       |
| 956 | regcomp()                      | 1707                                     | RLIM_INFINITY            | 1061-1062 |
| 957 | regerror                       | 1707                                     | RLIM_SAVED_CUR           | 1062      |
| 958 | regexec                        | 1707                                     | RLIM_SAVED_MAX           | 1062      |
| 959 | regfree                        | 1707                                     | rmdir()                  | 1731      |
| 960 | register fork handlers         | 1486                                     | RMSGD                    | 1131      |
| 961 | REG_                           | 516                                      | RMSGN                    | 1131      |
| 962 | REG_constants                  |                                          | RNORM                    | 1131      |
| 963 | error return values of regcomp | 1709                                     | round robin              | 548       |
| 964 | used in regcomp                | 1707                                     | round()                  | 1735      |
| 965 | REG_BADBR                      | 1709                                     | routing                  | 562       |

|      |                          |                                          |                                      |                               |
|------|--------------------------|------------------------------------------|--------------------------------------|-------------------------------|
| 966  | RPROTDAT                 | 1131                                     | seed48()                             | 1751                          |
| 967  | RPROTDIS                 | 1131                                     | seekdir()                            | 1752                          |
| 968  | RPROTNORM                | 1131                                     | SEEK_CUR                             | 824, 939, 1241                |
| 969  | RS_HIPRI                 | 1028, 1130, 1662                         | SEEK_END                             | 824, 939, 1241                |
| 970  | RTLD_GLOBAL              | 750, 754-755, 757                        | SEEK_GET                             | 1725                          |
| 971  | RTLD_LAZY                | 754, 757                                 | SEEK_SET                             | 544, 612, 618, 824, 939, 1241 |
| 972  | RTLD_LOCAL               | 755                                      | SEGV                                 | 516                           |
| 973  | RTLD_NEXT                | 757-758                                  | select()                             | 1753                          |
| 974  | RTLD_NOW                 | 754-755                                  | SEM                                  | 504                           |
| 975  | RTS                      | 504                                      | semctl()                             | 1771                          |
| 976  | RTSIG_MAX                | 1983                                     | semget()                             | 1774                          |
| 977  | RUSAGE                   | 516                                      | semid                                | 541                           |
| 978  | RUSAGE_CHILDREN          | 1064                                     | semop()                              | 1777                          |
| 979  | RUSAGE_SELF              | 1064                                     | SEM                                  | 514, 516                      |
| 980  | SA                       | 516                                      | sem_close()                          | 1754                          |
| 981  | SA_NOCLDSTOP             | 530, 1852, 1856                          | sem_destroy()                        | 1755                          |
| 982  | SA_NOCLDWAIT             | 801-802, 805, 1064, 1854, 2097           | SEM_FAILED                           | 1760                          |
| 983  | SA_NODEFER               | 1854                                     | sem_getvalue()                       | 1756                          |
| 984  | SA_ONSTACK               | 791, 799, 1853                           | sem_init()                           | 1757                          |
| 985  | SA_RESETHAND             | 647, 1853-1854                           | SEM_NSEMS_MAX                        | 1757, 1983                    |
| 986  | SA_RESTART               | 647, 1853, 1867                          | sem_open()                           | 1759                          |
| 987  | SA_SIGINFO               | 1852-1853, 1856, 1880                    | sem_perm                             | 541                           |
| 988  | scalb()                  | 1736                                     | sem_post()                           | 1762                          |
| 989  | scalbln()                | 1737                                     | sem_timedwait()                      | 1764                          |
| 990  | scanf                    | 932                                      | sem_trywait()                        | 1766                          |
| 991  | scanf()                  | 1739                                     | SEM_UNDO                             | 1777                          |
| 992  | schedule alarm           | 621                                      | sem_unlink()                         | 1768                          |
| 993  | scheduling documentation | 557                                      | SEM_VALUE_MAX                        | 1759, 1983                    |
| 994  | scheduling policy        |                                          |                                      | 1757                          |
| 995  | round robin              | 548                                      | sem_wait                             | 1766                          |
| 996  | SCHED                    | 514                                      | sem_wait()                           | 1770                          |
| 997  | SCHED_FIFO               | 544, 547, 556, 792, 887                  | send()                               | 1782                          |
| 998  |                          | 1048, 1344, 1497, 1499, 1569, 1603       | sendmsg()                            | 1784                          |
| 999  |                          | 1621, 1745, 1762                         | sendto()                             | 1787                          |
| 1000 | sched_getparam()         | 1741                                     | service name                         | 921                           |
| 1001 | sched_getscheduler()     | 1742                                     | session                              | 804, 1194, 1811, 1824         |
| 1002 | sched_get_priority_max() | 1740                                     | set cancelability state              | 1640                          |
| 1003 | sched_get_priority_min   | 1740                                     | set file creation mask               | 2063                          |
| 1004 | SCHED_OTHER              | 547, 550, 1499, 1569, 1745               | set process group ID for job control | 1811                          |
| 1005 | SCHED_RR                 | 544, 547-548, 556, 792, 887, 1048        | set-group-ID                         | 686, 797, 803, 828            |
| 1006 |                          | 1344, 1497, 1499, 1569, 1621, 1745, 1762 | set-user-ID                          | 797, 803, 991, 1194           |
| 1007 | sched_rr_get_interval()  | 1743                                     | SETALL                               | 1771, 1774                    |
| 1008 | sched_setparam()         | 1744                                     | setbuf()                             | 1790                          |
| 1009 | sched_setscheduler()     | 1747                                     | setcontext                           | 988                           |
| 1010 | SCHED_SPORADIC           | 544, 547-548, 1621                       | setcontext()                         | 1791                          |
| 1011 |                          | 1745, 1762                               | setgid()                             | 1792                          |
| 1012 | sched_yield()            | 1750                                     | setenv()                             | 1793                          |
| 1013 | SCM                      | 516                                      | seteuid()                            | 1795                          |
| 1014 | SD                       | 504                                      | setgid()                             | 1796                          |
| 1015 | security considerations  | 689, 803, 1194, 1811, 1914               | setgrent                             | 769                           |
| 1016 | seed48                   | 759                                      | setgrent()                           | 1798                          |



## Index

|      |               |                                        |                                |                                                               |
|------|---------------|----------------------------------------|--------------------------------|---------------------------------------------------------------|
| 1017 | sethostent    | 771                                    | SHM_RDONLY                     | 1842                                                          |
| 1018 | sethostent()  | <b>1799</b>                            | SHM_RND                        | 1842                                                          |
| 1019 | setitimer     | 1023                                   | shm_unlink()                   | <b>1840</b>                                                   |
| 1020 | setitimer()   | <b>1800</b>                            | should                         | 497                                                           |
| 1021 | setjmp()      | <b>1801</b>                            | shutdown()                     | <b>1850</b>                                                   |
| 1022 | setkey()      | <b>1803</b>                            | SHUT_                          | 516                                                           |
| 1023 | setlocale()   | <b>1805</b>                            | SIGABRT                        | 595                                                           |
| 1024 | setlogmask    | 710                                    | sigaction()                    | <b>1852</b>                                                   |
| 1025 | setlogmask()  | <b>1809</b>                            | sigaddset()                    | <b>1859</b>                                                   |
| 1026 | setnetent     | 773                                    | SIGALRM                        | 621, 1023, 1898, 2059, 2078                                   |
| 1027 | setnetent()   | <b>1810</b>                            | sigaltstack()                  | <b>1860</b>                                                   |
| 1028 | setpgid()     | <b>1811</b>                            | SIGBUS                         | 1289, 1292, 1645                                              |
| 1029 | setpgrp()     | <b>1814</b>                            | SIGCANCEL                      | 1528                                                          |
| 1030 | setpriority   | 1048                                   | SIGCHLD                        | 711, 801-802, 805, 1064, 1092<br>1856, 1873, 1989, 2097, 2104 |
| 1031 | setpriority() | <b>1815</b>                            | SIGCLD                         | 1856                                                          |
| 1032 | setprotoent   | 775                                    | SIGCONT                        | 534, 802, 804, 1193-1194                                      |
| 1033 | setprotoent() | <b>1816</b>                            | sigdelset()                    | <b>1862</b>                                                   |
| 1034 | setpwent      | 777                                    | sigemptyset()                  | <b>1863</b>                                                   |
| 1035 | setpwent()    | <b>1817</b>                            | SIGEV_                         | 514                                                           |
| 1036 | setregid()    | <b>1818</b>                            | SIGEV_NONE                     | 529, 544                                                      |
| 1037 | setreuid()    | <b>1820</b>                            | SIGEV_SIGNAL                   | 529, 2028                                                     |
| 1038 | setrlimit     | 1061                                   | SIGEV_THREAD                   | 529-530                                                       |
| 1039 | setrlimit()   | <b>1822</b>                            | sigfillset()                   | <b>1865</b>                                                   |
| 1040 | setservent    | 779                                    | SIGFPE                         | 1645, 1872                                                    |
| 1041 | setservent()  | <b>1823</b>                            | sighold                        | 1872                                                          |
| 1042 | setsid()      | <b>1824</b>                            | sighold()                      | <b>1866</b>                                                   |
| 1043 | setsockopt()  | <b>1826</b>                            | SIGHUP                         | 704, 801-802, 804                                             |
| 1044 | setstate      | 1123                                   | sigignore                      | 1866, 1872                                                    |
| 1045 | setstate()    | <b>1829</b>                            | SIGILL                         | 1645, 1872                                                    |
| 1046 | setuid()      | <b>1830</b>                            | SIGINT                         | 889, 1989                                                     |
| 1047 | setutxent     | 781                                    | siginterrupt()                 | <b>1867</b>                                                   |
| 1048 | setutxent()   | <b>1833</b>                            | sigismember()                  | <b>1869</b>                                                   |
| 1049 | SETVAL        | 1771, 1774                             | SIGKILL                        | 1194, 1852, 1856, 1873                                        |
| 1050 | setvbuf()     | <b>1834</b>                            | siglongjmp()                   | <b>1870</b>                                                   |
| 1051 | shall         | 497                                    | Signal Generation and Delivery | 528                                                           |
| 1052 | shell         | 796, 804, 1026, 1044, 1194, 1812, 2100 | signal handler                 | 1872                                                          |
| 1053 | job           | <b>1194</b>                            | signal()                       | <b>1872</b>                                                   |
| 1054 | login         | 1026                                   | signaling a condition          | 1536                                                          |
| 1055 | shell scripts |                                        | signals                        | 528                                                           |
| 1056 | exec          | 796                                    | signbit()                      | <b>1876</b>                                                   |
| 1057 | shell, login  | 796                                    | sigpause                       | 1872                                                          |
| 1058 | SHM           | <b>504</b> , 516                       | sigpause()                     | <b>1877</b>                                                   |
| 1059 | shmat()       | <b>1842</b>                            | sigpending()                   | <b>1878</b>                                                   |
| 1060 | shmctl()      | <b>1844</b>                            | SIGPIPE                        | 821, 853, 910, 914, 940, 1663, 2166                           |
| 1061 | shmdt()       | <b>1846</b>                            | SIGPOLL                        | 704, 1129-1130                                                |
| 1062 | shmget()      | <b>1848</b>                            | sigprocmask                    | 1645                                                          |
| 1063 | shmid         | 541                                    | sigprocmask()                  | <b>1879</b>                                                   |
| 1064 | SHMLBA        | 1842                                   | SIGPROF                        | 1023                                                          |
| 1065 | SHM_          | 516                                    | sigqueue()                     | <b>1880</b>                                                   |
| 1066 | shm_open()    | <b>1836</b>                            | SIGQUEUE_MAX                   | 1880, 1983                                                    |
| 1067 | shm_perm      | 541                                    |                                |                                                               |

|      |                         |                                       |                                    |                              |
|------|-------------------------|---------------------------------------|------------------------------------|------------------------------|
| 1068 | SIGQUIT                 | 1989                                  | socket receive queue               | 564                          |
| 1069 | sigrelse                | 1872                                  | socket types                       | 563                          |
| 1070 | sigrelse()              | <b>1882</b>                           | socket()                           | <b>1901</b>                  |
| 1071 | SIGRTMAX                | 528, 530, 1880, 1887, 1891            | socketpair()                       | <b>1903</b>                  |
| 1072 | SIGRTMIN                | 528, 530, 1880, 1887, 1891            | sockets                            | 562                          |
| 1073 | SIGSEGV                 | 1062, 1333, 1493, 1645                | addressing                         | 562                          |
| 1074 | sigset                  | 1872, 1882                            | asynchronous errors                | 565                          |
| 1075 | sigsetjmp()             | <b>1883</b>                           | connection indication queue        | 565                          |
| 1076 | SIGSTKSZ                | 1860                                  | IPv4                               | 569                          |
| 1077 | SIGSTOP                 | 528, 1852, 1873                       | IPv6                               | 570                          |
| 1078 | )                       | 1856                                  | local UNIX connections             | 569                          |
| 1079 | sigsuspend()            | <b>1885</b>                           | options                            | 566                          |
| 1080 | sigtimedwait()          | <b>1887</b>                           | pending error                      | 564                          |
| 1081 | SIGTSTP                 | 528                                   | protocol families                  | 562                          |
| 1082 | SIGTTIN                 | 528, 857, 864, 1681                   | protocols                          | 562                          |
| 1083 | SIGTTOU                 | 528, 821, 853, 910, 914, 940          | signals                            | 565                          |
| 1084 |                         | 1998, 2000, 2002, 2009, 2012, 2166    | SOCK_                              | 516                          |
| 1085 | SIGURG                  | 1130                                  | SPI                                | <b>504</b>                   |
| 1086 | SIGVTALRM               | 1023                                  | SPN                                | <b>505</b>                   |
| 1087 | sigwait()               | <b>1891</b>                           | sporadic server policy             |                              |
| 1088 | sigwaitinfo             | 1887                                  | execution capacity                 | 548                          |
| 1089 | sigwaitinfo()           | <b>1893</b>                           | replenishment period               | 548                          |
| 1090 | SIGXCPU                 | 1061                                  | sprintf                            | 898                          |
| 1091 | SIGXFSZ                 | 1061, 2050, 2163                      | sprintf()                          | <b>1905</b>                  |
| 1092 | SIG_                    | 514, 516                              | spurious wakeup                    | 1536                         |
| 1093 | SIG_BLOCK               | 1645                                  | sqrt()                             | <b>1906</b>                  |
| 1094 | SIG_DFL                 | 530, 791, 1062, 1852, 1854, 1872-1873 | srand                              | 1675                         |
| 1095 | SIG_ERR                 | 647, 1873                             | srand()                            | <b>1908</b>                  |
| 1096 | SIG_HOLD                | 1873                                  | srand48                            | 759                          |
| 1097 | SIG_IGN                 | 530-531, 791, 797, 801-802            | srand48()                          | <b>1909</b>                  |
| 1098 |                         | 1064, 1852, 1872, 2097                | srandom                            | 1123                         |
| 1099 | SIG_SETMASK             | 1645                                  | random()                           | <b>1910</b>                  |
| 1100 | SIG_UNBLOCK             | 1645                                  | SS                                 | <b>505</b>                   |
| 1101 | sin()                   | <b>1894</b>                           | sscanf                             | 932                          |
| 1102 | sinh()                  | <b>1896</b>                           | sscanf()                           | <b>1911</b>                  |
| 1103 | SIO                     | <b>504</b>                            | SSIZE_MAX                          | 1322, 1679, 1691, 1933, 2164 |
| 1104 | SI_                     | 516                                   | SSIZE_MAX                          | 1306                         |
| 1105 | SI_ASYNCIO              | 532                                   | SS_                                | 516                          |
| 1106 | SI_MESGQ                | 532                                   | SS_DISABLE                         | 1860-1861                    |
| 1107 | SI_QUEUE                | 532                                   | SS_ONSTACK                         | 1860                         |
| 1108 | SI_TIMER                | 532                                   | stack size                         | 1488                         |
| 1109 | SI_USER                 | 532                                   | stat()                             | <b>1912</b>                  |
| 1110 | sleep()                 | <b>1898</b>                           | statvfs                            | 947                          |
| 1111 | SND                     | 516                                   | statvfs()                          | <b>1916</b>                  |
| 1112 | SNDZERO                 | 1133                                  | stderr                             | 1917                         |
| 1113 | snprintf                | 898, 1905                             | STDERR_FILENO                      | 1917                         |
| 1114 | SO                      | 516                                   | stdin                              | <b>1917</b>                  |
| 1115 | socket I/O mode         | 563                                   | STDIN_FILENO                       | 1377, 1917                   |
| 1116 | socket out-of-band data | 565                                   | stdio locking functions            | 869                          |
| 1117 | socket owner            | 564                                   | stdio with explicit client locking | 985                          |
| 1118 | socket queue limits     | 564                                   | stdout                             | 1917                         |

## Index

- 1119 `STDOUT_FILENO` .....1377, 1917
- 1120 `STR`.....516
- 1121 `strcasecmp()`.....**1919**
- 1122 `strcat()` .....**1920**
- 1123 `strchr()`.....**1921**
- 1124 `strcmp()`.....**1922**
- 1125 `strcoll()` .....**1924**
- 1126 `strcpy()`.....**1926**
- 1127 `strcspn()` .....**1928**
- 1128 `strdup()` .....**1929**
- 1129 `STREAM`.....1132, 1134, 1662, 1680, 2164
- 1130 `stream`
  - 1131 `byte-oriented`.....537
  - 1132 `wide-oriented`.....537
- 1133 `STREAM head/tail`.....**539**
- 1134 `STREAMS` .....491, 522
- 1135 `streams`.....535
- 1136 `STREAMS` .....704, 813, 832, 1028, 1128
- 1137 .....1143, 1352-1354, 1373
- 1138 `access`.....540
- 1139 `streams`
  - 1140 `interaction with file descriptors`.....535
- 1141 `STREAMS`
  - 1142 `multiplexed`.....1136
  - 1143 `overview`.....539
- 1144 `streams`
  - 1145 `stream orientation`.....537
- 1146 `STREAM_MAX`.....835, 884, 1377, 1983
- 1147 `strerror()`.....**1930**
- 1148 `strfmon()`.....**1932**
- 1149 `strftime()`.....**1936**
- 1150 `strlen()` .....**1941**
- 1151 `strncasecmp`.....1919
- 1152 `strncasecmp()` .....**1943**
- 1153 `strncat()` .....**1944**
- 1154 `strncmp()` .....**1945**
- 1155 `strncpy()`.....**1946**
- 1156 `strpbrk()`.....**1947**
- 1157 `strptime()`.....**1948**
- 1158 `strrchr()` .....**1952**
- 1159 `strspn()`.....**1954**
- 1160 `strstr()`.....**1955**
- 1161 `strtod()`.....**1956**
- 1162 `strtoimax()`.....**1960**
- 1163 `strtok()`.....**1961**
- 1164 `strtok_r`.....1961
- 1165 `strtol()`.....**1964**
- 1166 `strtoul()` .....**1967**
- 1167 `strxfrm()`.....**1970**
- 1168 `ST_NOSUID`.....791, 799, 947
- 1169 `ST_RDONLY` .....947
- `superuser`.....601, 689, 1210, 2073
- `supplementary groups`.....689, 1011
- `SVID` .....1884
- `SVR4`.....1291, 1336
- `SV_`.....516
- `swab()`.....**1972**
- `swapcontext`.....1245
- `swapcontext()` .....**1974**
- `swprintf` .....966
- `swprintf()`.....**1975**
- `swscanf` .....975
- `swscanf()`.....**1976**
- `symbols`
  - `POSIX.1`.....512
- `symlink()`.....**1977**
- `SYMLINK_MAX` .....892, 1977
- `SYMLOOP_MAX`.....1983
- `sync()` .....**1979**
- `synchronously accept a signal`.....1888
- `sysconf()`.....**1980**
- `syslog` .....710
- `syslog()`.....**1988**
- `system crash`.....950
- `System III`.....689, 2065
- `system interfaces` .....585
- `system name`.....2065
- `System V`.....621, 689, 798, 804, 826-827
  - .....894, 1044, 1194, 1269, 1732, 1824, 1856
  - .....1884, 2004, 2065
- `system()`.....**1989**
- `S_`.....516
- `S_BANDURG` .....1130
- `S_ERROR`.....1129
- `S_HANGUP`.....1130
- `S_HIPRI`.....1129
- `S_IFBLK`.....1274
- `S_IFCHR`.....1274
- `S_IFDIR`.....1274
- `S_IFIFO` .....1274
- `S_IFREG`.....1274
- `S_INPUT`.....1129
- `S_IRGRP` .....817, 944, 1274
- `S_IROTH`.....817, 944, 1274
- `S_IRUSR`.....817, 944, 1274
- `S_IRWXG` .....1274
- `S_IRWXO` .....1274
- `S_IRWXU` .....1274
- `S_ISGID`.....684, 686, 1274, 2050, 2164
- `S_ISUID`.....684, 686, 1274, 2050, 2164
- `S_ISVTX`.....684, 1274, 1722, 1732, 2071
- `S_IWGRP` .....817, 944, 1274

|      |                            |                |                                   |                               |
|------|----------------------------|----------------|-----------------------------------|-------------------------------|
| 1170 | S_IWOTH                    | 817, 944, 1274 | thread ID                         | 1563                          |
| 1171 | S_IWUSR                    | 817, 944, 1274 | thread IDs                        | 554                           |
| 1172 | S_IXGRP                    | 1274           | thread mutexes                    | 554                           |
| 1173 | S_IXOTH                    | 1274           | thread scheduling                 | 555                           |
| 1174 | S_IXUSR                    | 1274           | thread termination                | 1564                          |
| 1175 | S_MSG                      | 1129           | thread-safety                     | 553, 869                      |
| 1176 | S_OUTPUT                   | 1129           | thread-specific data key creation | 1578                          |
| 1177 | S_RDBAND                   | 1129-1130      | thread-specific data key deletion | 1580                          |
| 1178 | S_RDNORM                   | 1129           | thread-specific data management   | 1572                          |
| 1179 | S_WRBAND                   | 1129           | threads                           | 553                           |
| 1180 | S_WRNORM                   | 1129           | regular file operations           | 561                           |
| 1181 | TABSIZE                    | 649, 1239      | time()                            | 2025                          |
| 1182 | tan()                      | 1994           | timer ID                          | 2030                          |
| 1183 | tanh()                     | 1996           | TIMER_                            | 515-516                       |
| 1184 | tcdrain()                  | 1998           | TIMER_ABSTIME                     | 551, 700, 2032                |
| 1185 | tcflow()                   | 2000           | timer_create()                    | 2028                          |
| 1186 | tcflush()                  | 2002           | timer_delete()                    | 2031                          |
| 1187 | tcgetattr()                | 2004           | timer_getoverrun()                | 2032                          |
| 1188 | tcgetpgrp()                | 2006           | timer_gettime                     | 2032                          |
| 1189 | tcgetsid()                 | 2008           | TIMER_MAX                         | 1983                          |
| 1190 | TCIFLUSH                   | 2002           | timer_settime                     | 2032                          |
| 1191 | TCIOFF                     | 2000           | times()                           | 2035                          |
| 1192 | TCIOFLUSH                  | 2002           | timezone()                        | 2038                          |
| 1193 | TCION                      | 2000           | TMO                               | 505                           |
| 1194 | TCOFLUSH                   | 2002           | tmpfile()                         | 2039                          |
| 1195 | TCOOFF                     | 2000           | tmpnam()                          | 2041                          |
| 1196 | TCOON                      | 2000           | TMP_MAX                           | 2021, 2041                    |
| 1197 | TCP_                       | 516            | TMR                               | 505                           |
| 1198 | TCSADRAIN                  | 2011           | toascii()                         | 2043                          |
| 1199 | TCSAFLUSH                  | 2011           | tolower()                         | 2044                          |
| 1200 | TCSANOW                    | 2011           | TOSTOP                            | 821, 853, 910, 914, 940, 2166 |
| 1201 | tcsendbreak()              | 2009           | toupper()                         | 2045                          |
| 1202 | tcsetattr()                | 2011           | towctrans()                       | 2046                          |
| 1203 | tcsetpgrp()                | 2014           | towlower()                        | 2047                          |
| 1204 | TCT                        | 505            | toupper()                         | 2048                          |
| 1205 | tdelete()                  | 2016           | TPI                               | 505                           |
| 1206 | TEF                        | 506            | TPP                               | 506                           |
| 1207 | telldir()                  | 2020           | TPS                               | 506                           |
| 1208 | tempnam()                  | 2021           | TRAP_                             | 516                           |
| 1209 | terminal access control    | 2004, 2012     | TRC                               | 506                           |
| 1210 | terminal device name       | 2054           | TRI                               | 506                           |
| 1211 | terminate a process        | 803            | TRL                               | 506                           |
| 1212 | terminology                | 497            | trunc()                           | 2049                          |
| 1213 | termios structure          | 2004           | truncate()                        | 2050                          |
| 1214 | tfind()                    | 2023           | TSA                               | 506                           |
| 1215 | tgamma()                   | 2024           | tsearch()                         | 2052                          |
| 1216 | THR                        | 505            | TSF                               | 506                           |
| 1217 | thread cancelation         |                | TSH                               | 507                           |
| 1218 | cleanup handlers           | 560            | TSP                               | 507                           |
| 1219 | thread creation            | 1559           | TSS                               | 507                           |
| 1220 | thread creation attributes | 1488           | ttyname()                         | 2053                          |

## Index

|      |                            |                      |                              |            |
|------|----------------------------|----------------------|------------------------------|------------|
| 1221 | ttyname_r                  | 2053                 | vsnprintf                    | 2088, 2093 |
| 1222 | TTY_NAME_MAX               | 1983, 2053           | vsprintf                     | 2088, 2093 |
| 1223 | twalk()                    | 2055                 | vswprintf                    | 2091       |
| 1224 | TYM                        | 507                  | vswprintf()                  | 2095       |
| 1225 | tzname                     | 2056-2057            | vswscanf()                   | 2096       |
| 1226 | TZNAME_MAX                 | 1983                 | vwprintf                     | 2091, 2095 |
| 1227 | tzset                      | 2057                 | wait for process termination | 2100       |
| 1228 | tzset()                    | 2057                 | wait for thread termination  | 1574       |
| 1229 | t_uscalar_t                | 1131                 | wait()                       | 2097       |
| 1230 | ualarm()                   | 2059                 | waitid()                     | 2104       |
| 1231 | UINT_MAX                   | 621, 1899            | waiting on a condition       | 1544       |
| 1232 | ulimit()                   | 2061                 | waitpid                      | 2097       |
| 1233 | ULONG_MAX                  | 1968, 2139           | waitpid()                    | 2106       |
| 1234 | UL_GETFSIZE                | 2061-2062            | WARNING                      | 879        |
| 1235 | UL_SETFSIZE                | 2061                 | warning                      |            |
| 1236 | umask()                    | 2063                 | MAN                          | 502        |
| 1237 | UN                         | 507                  | OB                           | 503        |
| 1238 | uname()                    | 2065                 | OF                           | 503        |
| 1239 | undefined                  | 497                  | UN                           | 507        |
| 1240 | underlying function        | 537                  | WCONTINUED                   | 2097, 2104 |
| 1241 | ungetc()                   | 2067                 | wcrtomb()                    | 2107       |
| 1242 | ungetwc()                  | 2069                 | wcscat()                     | 2109       |
| 1243 | unicast                    | 570                  | wcschr()                     | 2110       |
| 1244 | UNIX extension             | 491                  | wcscmp()                     | 2111       |
| 1245 | unlink()                   | 2071                 | wcscoll()                    | 2112       |
| 1246 | unlockpt()                 | 2076                 | wcscpy()                     | 2113       |
| 1247 | unsetenv()                 | 2077                 | wcscspn()                    | 2114       |
| 1248 | unspecified                | 498                  | wcsftime()                   | 2115       |
| 1249 | UP                         | 507                  | wcslen()                     | 2117       |
| 1250 | US-ASCII                   | 1142                 | wcsncat()                    | 2118       |
| 1251 | user ID                    |                      | wcsncmp()                    | 2119       |
| 1252 | real and effective         | 1820                 | wcsncpy()                    | 2120       |
| 1253 | setting real and effective | 1820                 | wcspbrk()                    | 2121       |
| 1254 | USER_PROCESS               | 781-782              | wcsrchr()                    | 2122       |
| 1255 | usleep()                   | 2078                 | wcrtombs()                   | 2123       |
| 1256 | UTC                        | 2057                 | wcsspn()                     | 2125       |
| 1257 | utime()                    | 2080                 | wcsstr()                     | 2126       |
| 1258 | utimes()                   | 2083                 | wcstod()                     | 2127       |
| 1259 | va_arg()                   | 2085                 | wcstoimax()                  | 2130       |
| 1260 | va_end                     | 2085                 | wcstok()                     | 2131       |
| 1261 | va_start                   | 2085                 | wcstol()                     | 2133       |
| 1262 | Version 7                  | 621, 689, 1194, 2065 | wcstombs()                   | 2136       |
| 1263 | vfork()                    | 2086                 | wcstoul()                    | 2138       |
| 1264 | vfprintf()                 | 2088                 | wcswcs()                     | 2141       |
| 1265 | vfprintf()                 | 2090                 | wcswidth()                   | 2142       |
| 1266 | vfwprintf()                | 2091                 | wcsxfrm()                    | 2143       |
| 1267 | vfwscanf()                 | 2092                 | wctob()                      | 2145       |
| 1268 | VISIT                      | 2055                 | wctomb()                     | 2146       |
| 1269 | vprintf                    | 2088                 | wctrans()                    | 2148       |
| 1270 | vprintf()                  | 2093                 | wctype()                     | 2149       |
| 1271 | vscanf()                   | 2094                 | wcwidth()                    | 2151       |

|      |                                      |                                   |
|------|--------------------------------------|-----------------------------------|
| 1272 | WEOF .....                           | 584, 1166, 1168, 1171, 1173       |
| 1273 | .....                                | 1175-1176, 1178, 1180, 1182, 1184 |
| 1274 | .....                                | 1186, 1188, 2047-2048, 2069       |
| 1275 | WEXITED.....                         | 2104                              |
| 1276 | WEXITSTATUS .....                    | 2098                              |
| 1277 | wide-oriented stream.....            | 537                               |
| 1278 | WIFCONTINUED.....                    | 2098                              |
| 1279 | WIFEXITED .....                      | 2098                              |
| 1280 | WIFSIGNALED .....                    | 2098                              |
| 1281 | WIFSTOPPED.....                      | 2098, 2101                        |
| 1282 | wmemchr().....                       | <b>2152</b>                       |
| 1283 | wmemcmp().....                       | <b>2153</b>                       |
| 1284 | wmemcpy() .....                      | <b>2154</b>                       |
| 1285 | wmemmove() .....                     | <b>2155</b>                       |
| 1286 | wmemset() .....                      | <b>2156</b>                       |
| 1287 | WNOHANG.....                         | 1856, 2097, 2104                  |
| 1288 | WNOWAIT.....                         | 2104                              |
| 1289 | wordexp().....                       | <b>2157</b>                       |
| 1290 | wordfree .....                       | 2157                              |
| 1291 | wprintf .....                        | 966                               |
| 1292 | wprintf().....                       | <b>2162</b>                       |
| 1293 | WRDE_.....                           | 516                               |
| 1294 | WRDE_APPEND.....                     | 2158                              |
| 1295 | WRDE_BADCHAR.....                    | 2159                              |
| 1296 | WRDE_BADVAL.....                     | 2159                              |
| 1297 | WRDE_CMDSUB .....                    | 2159                              |
| 1298 | WRDE_DOOFFS.....                     | 2158                              |
| 1299 | WRDE_NOCMD .....                     | 2158                              |
| 1300 | WRDE_NOSPACE .....                   | 2159                              |
| 1301 | WRDE_REUSE.....                      | 2158                              |
| 1302 | WRDE_SHOWERR.....                    | 2158                              |
| 1303 | WRDE_SYNTAX .....                    | 2159                              |
| 1304 | WRDE_UNDEF.....                      | 2158                              |
| 1305 | write to a file.....                 | 2168                              |
| 1306 | write() .....                        | <b>2163</b>                       |
| 1307 | writev .....                         | 2163                              |
| 1308 | wscanf .....                         | 975                               |
| 1309 | wscanf() .....                       | <b>2173</b>                       |
| 1310 | WSTOPPED .....                       | 2104                              |
| 1311 | WSTOPSIG .....                       | 2098                              |
| 1312 | WTERMSIG .....                       | 2098                              |
| 1313 | WUNTRACED .....                      | 2097, 2100                        |
| 1314 | XSI.....                             | <b>507</b>                        |
| 1315 | XSI interprocess communication ..... | <b>541</b>                        |
| 1316 | XSR .....                            | <b>508</b>                        |
| 1317 | X_OK.....                            | 601                               |
| 1318 | y0() .....                           | <b>2174</b>                       |
| 1319 | y1.....                              | 2174                              |
| 1320 | yn .....                             | 2174                              |
| 1321 | zombie process.....                  | 801                               |

# Introduction

## 1.1 Scope

The scope of IEEE Std. 1003.1-200x is described in the Base Definitions volume of IEEE Std. 1003.1-200x.

## 1.2 Conformance

Conformance requirements for IEEE Std. 1003.1-200x are defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 2, Conformance.

## 1.3 Normative References

Normative references for IEEE Std. 1003.1-200x are defined in the Base Definitions volume of IEEE Std. 1003.1-200x.

## 1.4 Changes from Issue 4

### Notes to Reviewers

*This section with side shading will not appear in the final copy. - Ed.*

The change history is subject to revision. The intention is to keep change history from Issue 4, and in the Issue 5 to Issue 6 change history to note changes from POSIX.2-1992 as well as Issue 5.

The following sections describe changes made to this volume of IEEE Std. 1003.1-200x since Issue 4. The CHANGE HISTORY section for each utility describes technical changes made to that utility since Issue 4. Changes made between Issue 2 and Issue 4 are not included.

### 1.4.1 Changes from Issue 4 to Issue 4, Version 2

The following list summarizes the major changes that were made in this volume of IEEE Std. 1003.1-200x from Issue 4 to Issue 4, Version 2:

- The X/Open UNIX extension was added, which specifies the common core utilities of 4.3 Berkeley Software Distribution (4.3 BSD), the OSF AES, and SVID Issue 3.

### 24 1.4.2 Changes from Issue 4, Version 2 to Issue 5

25 The following list summarizes the major changes that were made in this volume of  
26 IEEE Std. 1003.1-200x from Issue 4, Version 2 to Issue 5:

- 27 • Large File Summit (LFS) Extensions were added.
- 28 • Some utilities were updated to reflect changes for the POSIX Realtime Extension.
- 29 • Some utilities were updated to reflect changes for the POSIX Threads Extension.
- 30 • The LEGACY category of utilities was introduced as a replacement for the TO BE  
31 WITHDRAWN, WITHDRAWN, and Possibly Unsupportable categories.
- 32 • The following utilities were added:

33 *fuser*  
34 *ipcrm*  
35 *ipcs*  
36 *link*  
37 *unlink*

### 38 1.4.3 Changes from Issue 5 to Issue 6

39 The following list summarizes the major changes that were made in this volume of  
40 IEEE Std. 1003.1-200x from Issue 5 to Issue 6:

- 41 • This volume of IEEE Std. 1003.1-200x is extensively revised so it can be both an IEEE POSIX  
42 Standard and an Open Group Technical Standard.
- 43 • this volume of IEEE Std. 1003.1-200x is updated to mandate support of FIPS 151-2. The  
44 following changes were made:
  - 45 — Support is mandated for the capabilities associated with the following symbolic  
46 constants:  
47 *\_POSIX\_CHOWN\_RESTRICTED*  
48 *\_POSIX\_JOB\_CONTROL*  
49 *\_POSIX\_SAVED\_IDS*
  - 50 — In the environment for the login shell, the environment variables *LOGNAME* and *HOME*  
51 shall be defined and have the properties described in the Base Definitions volume of  
52 IEEE Std. 1003.1-200x, Chapter 7, Locale.
- 53 • this volume of IEEE Std. 1003.1-200x is updated to align with some features of the Single  
54 UNIX Specification.
- 55 • A RATIONALE section is added to each reference page.



## 56 1.5 Terminology

57 This section appears in the Base Definitions volume of IEEE Std. 1003.1-200x, but is repeated  
58 here for convenience:

59 For the purposes of IEEE Std. 1003.1-200x, the following terminology definitions apply:

### 60 **can**

61 Describes a permissible optional feature or behavior available to the user or application. The  
62 feature or behavior is mandatory for an implementation that conforms to  
63 IEEE Std. 1003.1-200x. An application can rely on the existence of the feature or behavior.

### 64 **implementation-defined**

65 Describes a value or behavior that is not defined by IEEE Std. 1003.1-200x but is selected by  
66 an implementor. The value or behavior may vary among implementations that conform to  
67 IEEE Std. 1003.1-200x. An application should not rely on the existence of the value or  
68 behavior. An application that relies on such a value or behavior cannot be assured to be  
69 portable across conforming implementations.

70 The implementor shall document such a value or behavior so that it can be used correctly  
71 by an application.

### 72 **legacy**

73 Describes a feature or behavior that is being retained for compatibility with older  
74 applications, but which has limitations which make it inappropriate for developing portable  
75 applications. New applications should use alternative means of obtaining equivalent  
76 functionality.

### 77 **may**

78 Describes a feature or behavior that is optional for an implementation that conforms to  
79 IEEE Std. 1003.1-200x. An application should not rely on the existence of the feature or  
80 behavior. An application that relies on such a feature or behavior cannot be assured to be  
81 portable across conforming implementations.

82 To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

### 83 **shall**

84 For an implementation that conforms to IEEE Std. 1003.1-200x, describes a feature or  
85 behavior that is mandatory. An application can rely on the existence of the feature or  
86 behavior.

87 For an application or user, describes a behavior that is mandatory.

### 88 **should**

89 For an implementation that conforms to IEEE Std. 1003.1-200x, describes a feature or  
90 behavior that is recommended but not mandatory. An application should not rely on the  
91 existence of the feature or behavior. An application that relies on such a feature or behavior  
92 cannot be assured to be portable across conforming implementations.

93 For an application, describes a feature or behavior that is recommended programming  
94 practice for optimum portability.

### 95 **undefined**

96 Describes the nature of a value or behavior not defined by IEEE Std. 1003.1-200x which  
97 results from use of an invalid program construct or invalid data input.

98 The value or behavior may vary among implementations that conform to  
99 IEEE Std. 1003.1-200x. An application should not rely on the existence or validity of the  
100 value or behavior. An application that relies on any particular value or behavior cannot be

101                   assured to be portable across conforming implementations.

102                   **unspecified**

103                   Describes the nature of a value or behavior not specified by IEEE Std. 1003.1-200x which  
104                   results from use of a valid program construct or valid data input.

105                   The value or behavior may vary among implementations that conform to  
106                   IEEE Std. 1003.1-200x. An application should not rely on the existence or validity of the  
107                   value or behavior. An application that relies on any particular value or behavior cannot be  
108                   assured to be portable across conforming implementations.

109 **1.6 Definitions**

110 Concepts and definitions are defined in the Base Definitions volume of IEEE Std. 1003.1-200x. |

## 111 1.7 Relationship to Other Documents

### 112 1.7.1 The System Interfaces volume of IEEE Std. 1003.1-200x

113 This subsection describes some of the features provided by the System Interfaces volume of  
 114 IEEE Std. 1003.1-200x that are assumed to be globally available by all systems conforming to this  
 115 volume of IEEE Std. 1003.1-200x. This subsection does not attempt to detail all of the features  
 116 defined in the System Interfaces volume of IEEE Std. 1003.1-200x that are required by all of the  
 117 utilities defined in this volume of IEEE Std. 1003.1-200x; the utility and function descriptions  
 118 point out additional functionality required to provide the corresponding specific features  
 119 needed by each.

120 The following subsections describe frequently used concepts. Many of these concepts are  
 121 described in the Base Definitions volume of IEEE Std. 1003.1-200x. Utility and function  
 122 description statements override these defaults when appropriate.

#### 123 1.7.1.1 Process Attributes

124 The following process attributes, as described in the System Interfaces volume of  
 125 IEEE Std. 1003.1-200x, are assumed to be supported for all processes in this volume of  
 126 IEEE Std. 1003.1-200x:

|     |                           |                         |
|-----|---------------------------|-------------------------|
| 127 | Controlling Terminal      | Real Group ID           |
| 128 | Current Working Directory | Real User ID            |
| 129 | Effective Group ID        | Root Directory          |
| 130 | Effective User ID         | Saved Set-Group-ID      |
| 131 | File Descriptors          | Saved Set-User-ID       |
| 132 | File Mode Creation Mask   | Session Membership      |
| 133 | Process Group ID          | Supplementary Group IDs |
| 134 | Process ID                |                         |

135 A conforming implementation may include additional process attributes.

#### 136 1.7.1.2 Concurrent Execution of Processes

137 The following functionality of the *fork()* function defined in the System Interfaces volume of  
 138 IEEE Std. 1003.1-200x shall be available on all systems conforming to this volume of  
 139 IEEE Std. 1003.1-200x:

- 140 1. Independent processes shall be capable of executing independently without either process  
 141 terminating.
- 142 2. A process shall be able to create a new process with all of the attributes referenced in  
 143 Section 1.7.1.1, determined according to the semantics of a call to the *fork()* function  
 144 defined in the System Interfaces volume of IEEE Std. 1003.1-200x followed by a call in the  
 145 child process to one of the *exec* functions defined in the System Interfaces volume of  
 146 IEEE Std. 1003.1-200x.

#### 147 1.7.1.3 File Access Permissions

148 The file access control mechanism described by the Base Definitions volume of  
 149 IEEE Std. 1003.1-200x, Section 4.1, File Access Permissions applies to all files on an  
 150 implementation conforming to this volume of IEEE Std. 1003.1-200x.

## 151 1.7.1.4 File Read, Write, and Creation

152 If a file that does not exist is to be written, it shall be created as described below, unless the  
 153 utility description states otherwise.

154 When a file that does not exist is created, the following features defined in the System Interfaces  
 155 volume of IEEE Std. 1003.1-200x shall apply unless the utility or function description states  
 156 otherwise:

- 157 1. The user ID of the file is set to the effective user ID of the calling process.
- 158 2. The group ID of the file is set to the effective group ID of the calling process or the group  
 159 ID of the directory in which the file is being created.
- 160 3. If the file is a regular file, the permission bits of the file are set to:  
 161 S\_IROTH | S\_IWOTH | S\_IRGRP | S\_IWGRP | S\_IRUSR | S\_IWUSR  
 162 (see the description of *File Modes* in the Base Definitions volume of IEEE Std. 1003.1-200x,  
 163 Chapter 13, Headers, <sys/stat.h>) except that the bits specified by the file mode creation  
 164 mask of the process are cleared. If the file is a directory, the permission bits are set to:  
 165 S\_IRWXU | S\_IRWXG | S\_IRWXO  
 166 except that the bits specified by the file mode creation mask of the process are cleared.
- 167 4. The *st\_atime*, *st\_ctime*, and *st\_mtime* fields of the file shall be updated as specified in the  
 168 System Interfaces volume of IEEE Std. 1003.1-200x, Section 2.5, Standard I/O Streams.
- 169 5. If the file is a directory, it shall be an empty directory; otherwise, the file shall have length  
 170 zero.
- 171 6. If the file is a symbolic link, the effect shall be undefined unless the {POSIX2\_SYMLINKS}  
 172 variable is in effect for the directory in which the symbolic link would be created.
- 173 7. Unless otherwise specified, the file created shall be a regular file.

174 When an attempt is made to create a file that already exists, the action shall depend on the file  
 175 type:

- 176 1. For directories and FIFO special files, the attempt shall fail and the utility shall either  
 177 continue with its operation or exit immediately with a non-zero status, depending on the  
 178 description of the utility.
- 179 2. For regular files:
  - 180 a. The user ID, group ID, and permission bits of the file shall not be changed.
  - 181 b. The file shall be truncated to zero length.
  - 182 c. The *st\_ctime* and *st\_mtime* fields shall be marked for update.
- 183 3. For other file types, the effect is implementation-defined.

184 When a file is to be appended, the file shall be opened in a manner equivalent to using the  
 185 O\_APPEND flag, without the O\_TRUNC flag, in the *open()* function defined in the System  
 186 Interfaces volume of IEEE Std. 1003.1-200x.

187 When a file is to be read or written, the file shall be opened with an access mode corresponding  
 188 to the operation to be performed. If file access permissions deny access, the requested operation  
 189 shall fail.

190 1.7.1.5 *File Removal*

191 When a directory that is the root directory or current working directory of any process is  
192 removed, the effect is implementation-defined. If file access permissions deny access, the  
193 requested operation fails. Otherwise, when a file is removed:

- 194 1. Its directory entry is removed from the file system.
- 195 2. The link count of the file is decremented.
- 196 3. If the file is an empty directory (see the Base Definitions volume of IEEE Std. 1003.1-200x,  
197 Section 3.145, Empty Directory):
  - 198 a. If no process has the directory open, the space occupied by the directory is freed and  
199 the directory is no longer accessible.
  - 200 b. If one or more processes have the directory open, the directory contents are  
201 preserved until all references to the file have been closed.
- 202 4. If the file is a directory that is not empty, the *st\_ctime* field is marked for update.
- 203 5. If the file is not a directory:
  - 204 a. If the link count becomes zero:
    - 205 i. If no process has the file open, the space occupied by the file is freed and the  
206 file is no longer accessible.
    - 207 ii. If one or more processes have the file open, the file contents are preserved until  
208 all references to the file have been closed.
  - 209 b. If the link count is not reduced to zero, the *st\_ctime* field is marked for update.
- 210 6. The *st\_ctime* and *st\_mtime* fields of the containing directory are marked for update.

211 1.7.1.6 *File Time Values*

212 All files shall have the three time values described by the Base Definitions volume of  
213 IEEE Std. 1003.1-200x, Section 4.3, File Times Update.

214 1.7.1.7 *File Contents*

215 When a reference is made to the contents of a file, *pathname*, this means the equivalent of all of  
216 the data placed in the space pointed to by *buf* when performing the *read()* function calls in the  
217 following operations defined in the System Interfaces volume of IEEE Std. 1003.1-200x:

```
218 while (read (fildes, buf, nbytes) > 0)
219 ;
```

220 If the file is indicated by a path name *pathname*, the file descriptor shall be determined by the  
221 equivalent of the following operation defined in the System Interfaces volume of  
222 IEEE Std. 1003.1-200x:

```
223 fildes = open (pathname, O_RDONLY);
```

224 The value of *nbytes* in the above sequence is unspecified; if the file is of a type where the data  
225 returned by *read()* would vary with different values, the value is one that results in the most  
226 data being returned.

227 If the *read()* function calls would return an error, it is unspecified whether the contents of the file  
228 are considered to include any data from offsets in the file beyond where the error would be  
229 returned.

230 **1.7.1.8 Path Name Resolution**

231 The path name resolution algorithm, described by the Base Definitions volume of  
232 IEEE Std. 1003.1-200x, Section 4.5, Path Name Resolution, is used by implementations  
233 conforming to this volume of IEEE Std. 1003.1-200x; see also the Base Definitions volume of  
234 IEEE Std. 1003.1-200x, Section 4.4, File Hierarchy.

235 **1.7.1.9 Changing the Current Working Directory**

236 When the current working directory (see the Base Definitions volume of IEEE Std. 1003.1-200x,  
237 Section 3.438, Working Directory) is to be changed, unless the utility or function description  
238 states otherwise, the operation shall succeed unless a call to the *chdir()* function defined in the  
239 System Interfaces volume of IEEE Std. 1003.1-200x would fail when invoked with the new  
240 working directory path name as its argument.

241 **1.7.1.10 Establish the Locale**

242 The functionality of the *setlocale()* function defined in the System Interfaces volume of  
243 IEEE Std. 1003.1-200x is assumed to be available on all systems conforming to this volume of  
244 IEEE Std. 1003.1-200x; that is, utilities that require the capability of establishing an international  
245 operating environment shall be permitted to set the specified category of the international  
246 environment.

247 **1.7.1.11 Actions Equivalent to Functions**

248 Some utility descriptions specify that a utility performs actions equivalent to a function defined  
249 in the System Interfaces volume of IEEE Std. 1003.1-200x. Such specifications require only that  
250 the external effects be equivalent, not that any effect within the utility and visible only to the  
251 utility be equivalent.

## 252 1.8 Portability

253 Some of the utilities in the Shell and Utilities volume of IEEE Std. 1003.1-200x and functions in  
254 the System Interfaces volume of IEEE Std. 1003.1-200x describe functionality that might not be  
255 fully portable to systems meeting the requirements for POSIX conformance (see the Base  
256 Definitions volume of IEEE Std. 1003.1-200x, Chapter 2, Conformance).

257 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in  
258 the margin identifies the nature of the option, extension, or warning (see Section 1.8.1). For  
259 maximum portability, an application should avoid such functionality.

260 Unless the primary task of a utility is to produce textual material on its standard output,  
261 application developers should not rely on the format or content of any such material that may be  
262 produced. Where the primary task *is* to provide such material, but the output format is  
263 incompletely specified, the description is marked with the OF margin code and shading.  
264 Application developers are warned not to expect that the output of such an interface on one  
265 system is any guide to its behavior on another system.

### 266 1.8.1 Codes

267 Codes and their meanings are listed in the Base Definitions volume of IEEE Std. 1003.1-200x, but  
268 are repeated here for convenience:

#### 269 ADV **Advisory Information**

270 The functionality described is optional. The functionality described is also an extension to the  
271 ISO C standard.

272 Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section.  
273 Where additional semantics apply to a function, the material is identified by use of the ADV  
274 margin legend.

#### 275 AIO **Asynchronous Input and Output**

276 The functionality described is optional. The functionality described is also an extension to the  
277 ISO C standard.

278 Where applicable, functions are marked with the AIO margin legend in the SYNOPSIS section.  
279 Where additional semantics apply to a function, the material is identified by use of the AIO  
280 margin legend.

#### 281 BAR **Barriers**

282 The functionality described is optional. The functionality described is also an extension to the  
283 ISO C standard.

284 Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.  
285 Where additional semantics apply to a function, the material is identified by use of the BAR  
286 margin legend.

#### 287 BE **Batch Environment Services and Utilities**

288 The functionality described is optional.

289 Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.  
290 Where additional semantics apply to a utility, the material is identified by use of the BE margin  
291 legend.

#### 292 CD **C-Language Development Utilities**

293 The functionality described is optional.

294 Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.  
295 Where additional semantics apply to a utility, the material is identified by use of the CD margin



296 legend.

297 CPT **Process CPU-Time Clocks**  
298 The functionality described is optional. The functionality described is also an extension to the  
299 ISO C standard.

300 Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.  
301 Where additional semantics apply to a function, the material is identified by use of the CPT  
302 margin legend.

303 CS **Clock Selection**  
304 The functionality described is optional. The functionality described is also an extension to the  
305 ISO C standard.

306 Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section.  
307 Where additional semantics apply to a function, the material is identified by use of the CS  
308 margin legend.

309 CX **Extension to the ISO C standard**  
310 The functionality described is an extension to the ISO C standard. Application writers may  
311 make use of an extension as it is supported on all IEEE Std. 1003.1-200x-conforming systems.

312 FD **FORTRAN Development Utilities**  
313 The functionality described is optional.

314 Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.  
315 Where additional semantics apply to a utility, the material is identified by use of the FD margin  
316 legend.

317 FR **FORTRAN Runtime Utilities**  
318 The functionality described is optional.

319 Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.  
320 Where additional semantics apply to a utility, the material is identified by use of the FR margin  
321 legend.

322 FSC **File Synchronization**  
323 The functionality described is optional. The functionality described is also an extension to the  
324 ISO C standard.

325 Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.  
326 Where additional semantics apply to a function, the material is identified by use of the FSC  
327 margin legend.

328 IP6 **IPV6**  
329 The functionality described is optional. The functionality described is also an extension to the  
330 ISO C standard.

331 Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.  
332 Where additional semantics apply to a function, the material is identified by use of the IP6  
333 margin legend.

334 MAN **Mandatory in the Next Draft**  
335 This is an interim draft code used to aid reviewers during the development of  
336 IEEE Std. 1003.1-200x. It denotes a feature that was previously an option or extension that is  
337 being brought into the mandatory base functionality. This margin code will be removed from the  
338 final draft.

339 MF **Memory Mapped Files**  
340 The functionality described is optional. The functionality described is also an extension to the

341 ISO C standard.

342 Where applicable, functions are marked with the MF margin legend in the SYNOPSIS section.  
343 Where additional semantics apply to a function, the material is identified by use of the MF  
344 margin legend.

345 ML **Process Memory Locking**  
346 The functionality described is optional. The functionality described is also an extension to the  
347 ISO C standard.

348 Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.  
349 Where additional semantics apply to a function, the material is identified by use of the ML  
350 margin legend.

351 MLR **Range Memory Locking**  
352 The functionality described is optional. The functionality described is also an extension to the  
353 ISO C standard.

354 Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.  
355 Where additional semantics apply to a function, the material is identified by use of the MLR  
356 margin legend.

357 MON **Monotonic Clock**  
358 The functionality described is optional. The functionality described is also an extension to the  
359 ISO C standard.

360 Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.  
361 Where additional semantics apply to a function, the material is identified by use of the MON  
362 margin legend.

363 MPR **Memory Protection**  
364 The functionality described is optional. The functionality described is also an extension to the  
365 ISO C standard.

366 Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section.  
367 Where additional semantics apply to a function, the material is identified by use of the MPR  
368 margin legend.

369 MSG **Message Passing**  
370 The functionality described is optional. The functionality described is also an extension to the  
371 ISO C standard.

372 Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.  
373 Where additional semantics apply to a function, the material is identified by use of the MSG  
374 margin legend.

375 OB **Obsolescent**  
376 The functionality described may be withdrawn in a future version of this volume of  
377 IEEE Std. 1003.1-200x. Strictly Conforming POSIX Applications and Strictly Conforming XSI  
378 Applications shall not use obsolescent features.

379 OF **Output Format Incompletely Specified**  
380 The functionality described is an XSI extension. The format of the output produced by the utility  
381 is not fully specified. It is therefore not possible to post-process this output in a consistent  
382 fashion. Typical problems include unknown length of strings and unspecified field delimiters.

383 OH **Optional Header**  
384 In the SYNOPSIS section of some interfaces in the System Interfaces volume of  
385 IEEE Std. 1003.1-200x an included header is marked as in the following example:

```
386 OH #include <sys/types.h>
387 #include <grp.h>
388 struct group *getgrnam(const char *name);
```

389 This indicates that the marked header is not required on XSI-conformant systems.

#### 390 PIO **Prioritized Input and Output**

391 The functionality described is optional. The functionality described is also an extension to the  
392 ISO C standard.

393 Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.  
394 Where additional semantics apply to a function, the material is identified by use of the PIO  
395 margin legend.

#### 396 PS **Process Scheduling**

397 The functionality described is optional. The functionality described is also an extension to the  
398 ISO C standard.

399 Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.  
400 Where additional semantics apply to a function, the material is identified by use of the PS  
401 margin legend.

#### 402 RTS **Realtime Signals Extension**

403 The functionality described is optional. The functionality described is also an extension to the  
404 ISO C standard.

405 Where applicable, functions are marked with the RTS margin legend in the SYNOPSIS section.  
406 Where additional semantics apply to a function, the material is identified by use of the RTS  
407 margin legend.

#### 408 SD **Software Development Utilities**

409 The functionality described is optional.

410 Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.  
411 Where additional semantics apply to a utility, the material is identified by use of the SD margin  
412 legend.

#### 413 SEM **Semaphores**

414 The functionality described is optional. The functionality described is also an extension to the  
415 ISO C standard.

416 Where applicable, functions are marked with the SEM margin legend in the SYNOPSIS section.  
417 Where additional semantics apply to a function, the material is identified by use of the SEM  
418 margin legend.

#### 419 SHM **Shared Memory Objects**

420 The functionality described is optional. The functionality described is also an extension to the  
421 ISO C standard.

422 Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.  
423 Where additional semantics apply to a function, the material is identified by use of the SHM  
424 margin legend.

#### 425 SIO **Synchronized Input and Output**

426 The functionality described is optional. The functionality described is also an extension to the  
427 ISO C standard.

428 Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.  
429 Where additional semantics apply to a function, the material is identified by use of the SIO  
430 margin legend.

|     |     |                                                                                                  |
|-----|-----|--------------------------------------------------------------------------------------------------|
| 431 | SPI | <b>Spin Locks</b>                                                                                |
| 432 |     | The functionality described is optional. The functionality described is also an extension to the |
| 433 |     | ISO C standard.                                                                                  |
| 434 |     | Where applicable, functions are marked with the SPI margin legend in the SYNOPSIS section.       |
| 435 |     | Where additional semantics apply to a function, the material is identified by use of the SPI     |
| 436 |     | margin legend.                                                                                   |
| 437 | SPN | <b>Spawn</b>                                                                                     |
| 438 |     | The functionality described is optional. The functionality described is also an extension to the |
| 439 |     | ISO C standard.                                                                                  |
| 440 |     | Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.       |
| 441 |     | Where additional semantics apply to a function, the material is identified by use of the SPN     |
| 442 |     | margin legend.                                                                                   |
| 443 | SS  | <b>Process Sporadic Server</b>                                                                   |
| 444 |     | The functionality described is optional. The functionality described is also an extension to the |
| 445 |     | ISO C standard.                                                                                  |
| 446 |     | Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.        |
| 447 |     | Where additional semantics apply to a function, the material is identified by use of the SS      |
| 448 |     | margin legend.                                                                                   |
| 449 | TCT | <b>Thread CPU-Time Clocks</b>                                                                    |
| 450 |     | The functionality described is optional. The functionality described is also an extension to the |
| 451 |     | ISO C standard.                                                                                  |
| 452 |     | Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.       |
| 453 |     | Where additional semantics apply to a function, the material is identified by use of the TCT     |
| 454 |     | margin legend.                                                                                   |
| 455 | THR | <b>Threads</b>                                                                                   |
| 456 |     | The functionality described is optional. The functionality described is also an extension to the |
| 457 |     | ISO C standard.                                                                                  |
| 458 |     | Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section.       |
| 459 |     | Where additional semantics apply to a function, the material is identified by use of the THR     |
| 460 |     | margin legend.                                                                                   |
| 461 | TMO | <b>Timeouts</b>                                                                                  |
| 462 |     | The functionality described is optional. The functionality described is also an extension to the |
| 463 |     | ISO C standard.                                                                                  |
| 464 |     | Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section.       |
| 465 |     | Where additional semantics apply to a function, the material is identified by use of the TMO     |
| 466 |     | margin legend.                                                                                   |
| 467 | TMR | <b>Timers</b>                                                                                    |
| 468 |     | The functionality described is optional. The functionality described is also an extension to the |
| 469 |     | ISO C standard.                                                                                  |
| 470 |     | Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section.       |
| 471 |     | Where additional semantics apply to a function, the material is identified by use of the TMR     |
| 472 |     | margin legend.                                                                                   |
| 473 | TPI | <b>Threads Priority Inheritance</b>                                                              |
| 474 |     | The functionality described is optional. The functionality described is also an extension to the |
| 475 |     | ISO C standard.                                                                                  |

476 Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.  
477 Where additional semantics apply to a function, the material is identified by use of the TPI  
478 margin legend.

479 TPP **Thread Priority Protection**  
480 The functionality described is optional. The functionality described is also an extension to the  
481 ISO C standard.

482 Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.  
483 Where additional semantics apply to a function, the material is identified by use of the TPP  
484 margin legend.

485 TPS **Thread Execution Scheduling**  
486 The functionality described is optional. The functionality described is also an extension to the  
487 ISO C standard.

488 Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.  
489 Where additional semantics apply to a function, the material is identified by use of the TPS  
490 margin legend.

491 TRC **Trace**  
492 The functionality described is optional. The functionality described is also an extension to the  
493 ISO C standard.

494 Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.  
495 Where additional semantics apply to a function, the material is identified by use of the TRC  
496 margin legend.

497 TEF **Trace Event Filter**  
498 The functionality described is optional. The functionality described is also an extension to the  
499 ISO C standard.

500 Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.  
501 Where additional semantics apply to a function, the material is identified by use of the TEF  
502 margin legend.

503 TRL **Trace Log**  
504 The functionality described is optional. The functionality described is also an extension to the  
505 ISO C standard.

506 Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.  
507 Where additional semantics apply to a function, the material is identified by use of the TRL  
508 margin legend.

509 TRI **Trace Inherit**  
510 The functionality described is optional. The functionality described is also an extension to the  
511 ISO C standard.

512 Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.  
513 Where additional semantics apply to a function, the material is identified by use of the TRI  
514 margin legend.

515 TSA **Thread Stack Address Attribute**  
516 The functionality described is optional. The functionality described is also an extension to the  
517 ISO C standard.

518 Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.  
519 Where additional semantics apply to a function, the material is identified by use of the TSA  
520 margin legend.

- 521 TSF **Thread-Safe Functions**  
522 The functionality described is optional. The functionality described is also an extension to the  
523 ISO C standard.
- 524 Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section.  
525 Where additional semantics apply to a function, the material is identified by use of the TSF  
526 margin legend.
- 527 TSH **Thread Process-Shared Synchronization**  
528 The functionality described is optional. The functionality described is also an extension to the  
529 ISO C standard.
- 530 Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.  
531 Where additional semantics apply to a function, the material is identified by use of the TSH  
532 margin legend.
- 533 TSP **Thread Sporadic Server**  
534 The functionality described is optional. The functionality described is also an extension to the  
535 ISO C standard.
- 536 Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.  
537 Where additional semantics apply to a function, the material is identified by use of the TSP  
538 margin legend.
- 539 TSS **Thread Stack Address Size**  
540 The functionality described is optional. The functionality described is also an extension to the  
541 ISO C standard.
- 542 Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.  
543 Where additional semantics apply to a function, the material is identified by use of the TSS  
544 margin legend.
- 545 TYM **Typed Memory Objects**  
546 The functionality described is optional. The functionality described is also an extension to the  
547 ISO C standard.
- 548 Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.  
549 Where additional semantics apply to a function, the material is identified by use of the TYM  
550 margin legend.
- 551 UN **Possibly Unsupportable Feature**  
552 The functionality described is an XSI extension. It need not be possible to implement the  
553 required functionality (as defined) on all conformant systems and the functionality need not be  
554 present. This may, for example, be the case where the conformant system is hosted and the  
555 underlying system provides the service in an alternative way.
- 556 UP **User Portability Utilities**  
557 The functionality described is optional.
- 558 Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.  
559 Where additional semantics apply to a utility, the material is identified by use of the UP margin  
560 legend.
- 561 XSI **Extension**  
562 The functionality described is an XSI extension. Functionality marked XSI is also an extension to  
563 the ISO C standard. Application writers may confidently make use of an extension on all  
564 systems supporting the X/Open System Interfaces Extension.

565 If an entire SYNOPSIS section is shaded and marked with one XSI, all the functionality described  
566 in that reference page is an extension. See the Base Definitions volume of IEEE Std. 1003.1-200x,  
567 Section 3.441, XSI.

568 XSR **XSI STREAMS**

569 The functionality described is optional. The functionality described is also an extension to the  
570 ISO C standard.

571 Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section.  
572 Where additional semantics apply to a function, the material is identified by use of the XSR  
573 margin legend.

574 **1.9 Utility Limits**

575 This section lists magnitude limitations imposed by a specific implementation. The braces  
576 notation, {LIMIT}, is used in this volume of IEEE Std. 1003.1-200x to indicate these values, but  
577 the braces are not part of the name.

578 **Table 1-1 Utility Limit Minimum Values**

| Name                      | Description                                                                                                                                                                                                                                                                 | Value  |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| {POSIX2_BC_BASE_MAX}      | The maximum <i>obase</i> value allowed by the <i>bc</i> utility.                                                                                                                                                                                                            | 99     |
| {POSIX2_BC_DIM_MAX}       | The maximum number of elements permitted in an array by the <i>bc</i> utility.                                                                                                                                                                                              | 2048   |
| {POSIX2_BC_SCALE_MAX}     | The maximum <i>scale</i> value allowed by the <i>bc</i> utility.                                                                                                                                                                                                            | 99     |
| {POSIX2_BC_STRING_MAX}    | The maximum length of a string constant accepted by the <i>bc</i> utility.                                                                                                                                                                                                  | 1000   |
| {POSIX2_COLL_WEIGHTS_MAX} | The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> <b>order</b> keyword in the locale definition file; see the <b>border_start</b> keyword in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 7.3.2, <i>LC_COLLATE</i> . | 2      |
| {POSIX2_EXPR_NEST_MAX}    | The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.                                                                                                                                                                         | 32     |
| {POSIX2_LINE_MAX}         | Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing newline.                                    | 2048   |
| {POSIX2_RE_DUP_MAX}       | The maximum number of repeated occurrences of a BRE permitted when using the interval notation $\{m,n\}$ ; see the Base Definitions volume of IEEE Std. 1003.1-200x, Section 9.3.6, BREs Matching Multiple Characters.                                                      | 255    |
| {POSIX2_VERSION}          | This value indicates the version of the utilities in this volume of IEEE Std. 1003.1-200x that are provided by the implementation. It changes with each published version.                                                                                                  | 199209 |

610 The values specified in Table 1-1 represent the lowest values conforming implementations shall  
611 provide and, consequently, the largest values on which an application can rely without further  
612 enquiries, as described below. These values shall be accessible to applications via the *getconf*  
613 utility (see *getconf* (on page 2692)) and through the *sysconf()* function defined in the System  
614 Interfaces volume of IEEE Std. 1003.1-200x. The literal names shown in Table 1-1 apply only to  
615 the *getconf* utility; the high-level language binding describes the exact form of each name to be  
616 used by the interfaces in that binding.

617 Implementations may provide more liberal, or less restrictive, values than shown in Table 1-1.  
618 These possibly more liberal values are accessible using the symbols in Table 1-2 (on page 2221).

619 The *sysconf()* function defined in the System Interfaces volume of IEEE Std. 1003.1-200x or the  
620 *getconf* utility return the value of each symbol on each specific implementation. The value so



621 retrieved is the largest, or most liberal, value that is available throughout the session lifetime, as  
 622 determined at session creation. The literal names shown in the table apply only to the *getconf*  
 623 utility; the high-level language binding describes the exact form of each name to be used by the  
 624 interfaces in that binding.

625 All numeric limits defined by the System Interfaces volume of IEEE Std. 1003.1-200x, such as  
 626 {PATH\_MAX}, also apply to this volume of IEEE Std. 1003.1-200x. All the utilities defined by this  
 627 volume of IEEE Std. 1003.1-200x are implicitly limited by these values, unless otherwise noted in  
 628 the utility descriptions.

629 It is not guaranteed that the application can actually reach the specified limit of an  
 630 implementation in any given case, or at all, as a lack of virtual memory or other resources may  
 631 prevent this. The limit value indicates only that the implementation does not specifically impose  
 632 any arbitrary, more restrictive limit.

633 **Table 1-2** Symbolic Utility Limits

| Name               | Description                                                                                                                                                                                                                                                                | Minimum Value             |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|
| {BC_BASE_MAX}      | The maximum <i>obase</i> value allowed by the <i>bc</i> utility.                                                                                                                                                                                                           | {POSIX2_BC_BASE_MAX}      |
| {BC_DIM_MAX}       | The maximum number of elements permitted in an array by the <i>bc</i> utility.                                                                                                                                                                                             | {POSIX2_BC_DIM_MAX}       |
| {BC_SCALE_MAX}     | The maximum <i>scale</i> value allowed by the <i>bc</i> utility.                                                                                                                                                                                                           | {POSIX2_BC_SCALE_MAX}     |
| {BC_STRING_MAX}    | The maximum length of a string constant accepted by the <i>bc</i> utility.                                                                                                                                                                                                 | {POSIX2_BC_STRING_MAX}    |
| {COLL_WEIGHTS_MAX} | The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> <b>order</b> keyword in the locale definition file; see the <b>order_start</b> keyword in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 7.3.2, <i>LC_COLLATE</i> . | {POSIX2_COLL_WEIGHTS_MAX} |
| {EXPR_NEST_MAX}    | The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.                                                                                                                                                                        | {POSIX2_EXPR_NEST_MAX}    |
| {LINE_MAX}         | Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the                                                     | {POSIX2_LINE_MAX}         |

669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681

| Name         | Description                                                                                                                                                                                                                                 | Minimum Value       |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| {RE_DUP_MAX} | trailing newline.<br>The maximum number of repeated occurrences of a BRE permitted when using the interval notation $\{m,n\}$ ; see the Base Definitions volume of IEEE Std. 1003.1-200x, Section 9.3.6, BREs Matching Multiple Characters. | {POSIX2_RE_DUP_MAX} |

682  
683

The following value may be a constant within an implementation or may vary from one path name to another.

684  
685  
686

{POSIX2\_SYMLINKS}

When referring to a directory, the system supports the creation of symbolic links within that directory; for non-directory files, the meaning of {POSIX2\_SYMLINKS} is undefined.

**687 1.10 Grammar Conventions**

688 Portions of this volume of IEEE Std. 1003.1-200x are expressed in terms of a special grammar  
689 notation. It is used to portray the complex syntax of certain program input. The grammar is  
690 based on the syntax used by the *yacc* utility. However, it does not represent fully functional *yacc*  
691 input, suitable for program use; the lexical processing and all semantic requirements are  
692 described only in textual form. The grammar is not based on source used in any traditional  
693 implementation and has not been tested with the semantic code that would normally be  
694 required to accompany it. Furthermore, there is no implication that the partial *yacc* code  
695 presented represents the most efficient, or only, means of supporting the complex syntax within  
696 the utility. Implementations may use other programming languages or algorithms, as long as the  
697 syntax supported is the same as that represented by the grammar.

698 The following typographical conventions are used in the grammar; they have no significance  
699 except to aid in reading.

- 700 • The identifiers for the reserved words of the language are shown with a leading capital letter.  
701 (These are terminals in the grammar; for example, **While**, **Case**.)
- 702 • The identifiers for terminals in the grammar are all named with uppercase letters and  
703 underscores; for example, **NEWLINE**, **ASSIGN\_OP**, **NAME**.
- 704 • The identifiers for non-terminals are all lowercase.

## 705 1.11 Utility Description Defaults

706 This section describes all of the subsections used within the utility descriptions, including:

- 707 • Intended usage of the section
- 708 • Global defaults that affect all the standard utilities
- 709 • The meanings of notations used in this volume of IEEE Std. 1003.1-200x that are specific to
- 710 individual utility sections

711 Integer variables and constants, including the values of operands and option-arguments, used  
712 by the utilities listed in this volume of IEEE Std. 1003.1-200x shall be implemented as equivalent  
713 to the ISO C standard **signed long** data type. Conversion between types shall be as described in  
714 the ISO C standard. The evaluation of arithmetic expressions shall be equivalent to that  
715 described in Section 6.3 of the ISO C standard.

### 716 NAME

717 This section gives the name or names of the utility and briefly states its purpose.

### 718 SYNOPSIS

719 The SYNOPSIS section summarizes the syntax of the calling sequence for the utility,  
720 including options, option-arguments, and operands. Standards for utility naming are  
721 described in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2, Utility  
722 Syntax Guidelines; for describing the utility's arguments in the Base Definitions volume  
723 of IEEE Std. 1003.1-200x, Section 12.1, Utility Argument Syntax.

### 724 DESCRIPTION

725 The DESCRIPTION section describes the actions of the utility. If the utility has a very  
726 complex set of subcommands or its own procedural language, an EXTENDED  
727 DESCRIPTION section is also provided. Most explanations of optional functionality are  
728 omitted here, as they are usually explained in the OPTIONS section.

729 Some utilities in this volume of IEEE Std. 1003.1-200x are described in terms of  
730 functionality equivalent to the System Interfaces volume of IEEE Std. 1003.1-200x.  
731 When specific functions are cited, the underlying operating system provides equivalent  
732 functionality and all side effects associated with successful execution of the function.  
733 The treatment of errors and intermediate results from the individual functions cited is  
734 generally not specified by this volume of IEEE Std. 1003.1-200x. See the utility's EXIT  
735 STATUS and CONSEQUENCES OF ERRORS sections for all actions associated with  
736 errors encountered by the utility.

### 737 OPTIONS

738 The OPTIONS section describes the utility options and option-arguments, and how  
739 they modify the actions of the utility. Standard utilities that have options either fully  
740 comply with the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2, Utility  
741 Syntax Guidelines or describe all deviations. Apparent disagreements between  
742 functionality descriptions in the OPTIONS and DESCRIPTION (or EXTENDED  
743 DESCRIPTION) sections are always resolved in favor of the OPTIONS section.

744 Each OPTIONS section that uses the phrase "The ... utility shall conform to the Utility  
745 Syntax Guidelines ..." refers only to the use of the utility as specified by this volume of  
746 IEEE Std. 1003.1-200x; implementation extensions should also conform to the  
747 guidelines, but may allow exceptions for historical practice.

748 Unless otherwise stated in the utility description, when given an option unrecognized  
749 by the implementation, or when a required option-argument is not provided, standard  
750 utilities shall issue a diagnostic message to standard error and exit with a non-zero exit

751 status.

752 XSI All utilities in this volume of IEEE Std. 1003.1-200x shall be capable of processing  
753 arguments using 8-bit transparency.

754 **Default Behavior:** When this section is listed as “None.”, it means that the  
755 implementation need not support any options. Standard utilities that do not accept  
756 options, but that do accept operands, shall recognize “—” as a first argument to be  
757 discarded.

758 The requirement for recognizing “—” is because portable applications need a way to  
759 shield their operands from any arbitrary options that the implementation may provide  
760 as an extension. For example, if the standard utility *foo* is listed as taking no options,  
761 and the application needed to give it a path name with a leading hyphen, it could safely  
762 do it as:

```
763 foo -- -myfile
```

764 and avoid any problems with `-m` used as an extension.

## 765 OPERANDS

766 The OPERANDS section describes the utility operands, and how they affect the actions  
767 of the utility. Apparent disagreements between functionality descriptions in the  
768 OPERANDS and DESCRIPTION (or EXTENDED DESCRIPTION) sections shall be  
769 resolved in favor of the OPERANDS section.

770 If an operand naming a file can be specified as ‘-’, which means to use the standard  
771 input instead of a named file, this is explicitly stated in this section. Unless otherwise  
772 stated, the use of multiple instances of ‘-’ to mean standard input in a single  
773 command produces unspecified results.

774 Unless otherwise stated, the standard utilities that accept operands shall process those  
775 operands in the order specified in the command line.

776 **Default Behavior:** When this section is listed as “None.”, it means that the  
777 implementation need not support any operands.

## 778 STDIN

779 The STDIN section describes the standard input of the utility. This section is frequently  
780 merely a reference to the following section, as many utilities treat standard input and  
781 input files in the same manner. Unless otherwise stated, all restrictions described in the  
782 INPUT FILES section shall apply to this section as well.

783 Use of a terminal for standard input can cause any of the standard utilities that read  
784 standard input to stop when used in the background. For this reason, applications  
785 should not use interactive features in scripts to be placed in the background.

786 The specified standard input format of the standard utilities shall not depend on the  
787 existence or value of the environment variables defined in this volume of  
788 IEEE Std. 1003.1-200x, except as provided by this volume of IEEE Std. 1003.1-200x.

789 **Default Behavior:** When this section is listed as “Not used.”, it means that the  
790 standard input shall not be read when the utility is used as described by this volume of  
791 IEEE Std. 1003.1-200x.

## 792 INPUT FILES

793 The INPUT FILES section describes the files, other than the standard input, used as  
794 input by the utility. It includes files named as operands and option-arguments as well  
795 as other files that are referred to, such as start-up and initialization files, databases, and

796 so on. Commonly-used files are generally described in one place and cross-referenced  
797 by other utilities.

798 XSI All utilities in this volume of IEEE Std. 1003.1-200x shall be capable of processing input  
799 files using 8-bit transparency.

800 When a standard utility reads a seekable input file and terminates without an error  
801 before it reaches end-of-file, the utility shall ensure that the file offset in the open file  
802 description is properly positioned just past the last byte processed by the utility. For  
803 files that are not seekable, the state of the file offset in the open file description for that  
804 file is unspecified. A portable application shall not assume that the following three  
805 commands are equivalent:

```
806 tail -n +2 file
807 (sed -n 1q; cat) < file
808 cat file | (sed -n 1q; cat)
```

809 The second command is equivalent to the first only when the file is seekable. The third  
810 command leaves the file offset in the open file description in an unspecified state. Other  
811 utilities, such as *head*, *read*, and *sh*, have similar properties.

812 Some of the standard utilities, such as filters, process input files a line or a block at a  
813 time and have no restrictions on the maximum input file size. Some utilities may have  
814 size limitations that are not as obvious as file space or memory limitations. Such  
815 limitations should reflect resource limitations of some sort, not arbitrary limits set by  
816 implementors. Implementations shall document those utilities that are limited by  
817 constraints other than file system space, available memory, and other limits specifically  
818 cited by this volume of IEEE Std. 1003.1-200x, and identify what the constraint is and  
819 indicate a way of estimating when the constraint would be reached. Similarly, some  
820 utilities descend the directory tree (recursively). Implementations shall also document  
821 any limits that they may have in descending the directory tree that are beyond limits  
822 cited by this volume of IEEE Std. 1003.1-200x.

823 When an input file is described as a *text file*, the utility produces undefined results if  
824 given input that is not from a text file, unless otherwise stated. Some utilities (for  
825 example, *make*, *read*, *sh*) allow for continued input lines using an escaped <newline>  
826 convention; unless otherwise stated, the utility need not be able to accumulate more  
827 than {LINE\_MAX} bytes from a set of multiple, continued input lines. Thus, for a  
828 portable application the total of all the continued lines in a set cannot exceed  
829 {LINE\_MAX}. If a utility using the escaped <newline> convention detects an end-of-  
830 file condition immediately after an escaped <newline>, the results are unspecified.

831 Record formats are described in a notation similar to that used by the C-language  
832 function, *printf()*. See the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 5,  
833 File Format Notation for a description of this notation. The format description is  
834 intended to be sufficiently rigorous to allow other applications to generate these input  
835 files. However, since <blank> characters can legitimately be included in some of the  
836 fields described by the standard utilities, particularly in locales other than the POSIX  
837 locale, this intent is not always realized.

838 **Default Behavior:** When this section is listed as “None.”, it means that no input files  
839 are required to be supplied when the utility is used as described by this volume of  
840 IEEE Std. 1003.1-200x.

#### 841 ENVIRONMENT VARIABLES

842 The ENVIRONMENT VARIABLES section lists what variables affect the utility’s  
843 execution.

844 The entire manner in which environment variables described in this volume of  
 845 IEEE Std. 1003.1-200x affect the behavior of each utility is described in the  
 846 ENVIRONMENT VARIABLES section for that utility, in conjunction with the global  
 847 XSI effects of the *LANG*, *LC\_ALL*, and *NLSPATH* environment variables described in the  
 848 Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables.  
 849 The existence or value of environment variables described in this volume of  
 850 IEEE Std. 1003.1-200x shall not otherwise affect the specified behavior of the standard  
 851 utilities. Any effects of the existence or value of environment variables not described by  
 852 this volume of IEEE Std. 1003.1-200x upon the standard utilities are unspecified.

853 For those standard utilities that use environment variables as a means for selecting a  
 854 utility to execute (such as *CC* in *make*), the string provided to the utility is subjected to  
 855 the path search described for *PATH* in the Base Definitions volume of  
 856 IEEE Std. 1003.1-200x, Chapter 8, Environment Variables.

857 XSI All utilities in this volume of IEEE Std. 1003.1-200x shall be capable of processing  
 858 environment variable names and values using 8-bit transparency.

859 **Default Behavior:** When this section is listed as “None.”, it means that the behavior of  
 860 the utility is not directly affected by environment variables described by this volume of  
 861 IEEE Std. 1003.1-200x when the utility is used as described by this volume of  
 862 IEEE Std. 1003.1-200x.

### 863 ASYNCHRONOUS EVENTS

864 The ASYNCHRONOUS EVENTS section lists how the utility reacts to such events as  
 865 signals and what signals are caught.

866 **Default Behavior:** When this section is listed as “Default.”, or it refers to “the standard  
 867 action for all other signals; see Section 1.11 (on page 2224)” it means that the action  
 868 taken as a result of the signal shall be one of the following:

- 869 1. The action is that inherited from the parent according to the rules of inheritance  
 870 of signal actions defined in the System Interfaces volume of IEEE Std. 1003.1-200x.
- 871 2. When no action has been taken to change the default, the default action is that  
 872 specified by the System Interfaces volume of IEEE Std. 1003.1-200x.
- 873 3. The result of the utility’s execution is as if default actions had been taken.

874 A utility is permitted to catch a signal, perform some additional processing (such as  
 875 deleting temporary files), restore the default signal action (or action inherited from the  
 876 parent process), and resignal itself.

### 877 STDOUT

878 The STDOUT section describes the standard output of the utility. This section is  
 879 frequently merely a reference to the following section, OUTPUT FILES, because many  
 880 utilities treat standard output and output files in the same manner.

881 Use of a terminal for standard output may cause any of the standard utilities that write  
 882 standard output to stop when used in the background. For this reason, applications  
 883 should not use interactive features in scripts to be placed in the background.

884 Record formats are described in a notation similar to that used by the C-language  
 885 function, *printf()*. See the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 5,  
 886 File Format Notation for a description of this notation.

887 The specified standard output of the standard utilities shall not depend on the  
 888 existence or value of the environment variables defined in this volume of  
 889 IEEE Std. 1003.1-200x, except as provided by this volume of IEEE Std. 1003.1-200x.

890 Some of the standard utilities describe their output using the verb *display*, defined in  
891 the Base Definitions volume of IEEE Std. 1003.1-200x, Section 3.135, Display. Output  
892 described in the STDOUT sections of such utilities may be produced using means other  
893 than standard output. When standard output is directed to a terminal, the output  
894 described shall be written directly to the terminal. Otherwise, the results are undefined.

895 **Default Behavior:** When this section is listed as “Not used.”, it means that the  
896 standard output shall not be written when the utility is used as described by this  
897 volume of IEEE Std. 1003.1-200x.

#### 898 **STDERR**

899 The STDERR section describes the standard error output of the utility. Only those  
900 messages that are purposely sent by the utility are described.

901 Use of a terminal for standard error may cause any of the standard utilities that write  
902 standard error output to stop when used in the background. For this reason,  
903 applications should not use interactive features in scripts to be placed in the  
904 background.

905 The format of diagnostic messages for most utilities is unspecified, but the language  
906 and cultural conventions of diagnostic and informative messages whose format is  
907 unspecified by this volume of IEEE Std. 1003.1-200x should be affected by the setting of  
908 XSI *LC\_MESSAGES* and *NLSPATH*.

909 The specified standard error output of standard utilities shall not depend on the  
910 existence or value of the environment variables defined in this volume of  
911 IEEE Std. 1003.1-200x, except as provided by this volume of IEEE Std. 1003.1-200x.

912 **Default Behavior:** When this section is listed as “Used only for diagnostic messages.”,  
913 it means that, unless otherwise stated, the diagnostic messages shall be sent to the  
914 standard error only when the exit status is non-zero and the utility is used as described  
915 by this volume of IEEE Std. 1003.1-200x.

916 When this section is listed as “Not used.”, it means that the standard error shall not be  
917 used when the utility is used as described in this volume of IEEE Std. 1003.1-200x.

#### 918 **OUTPUT FILES**

919 The OUTPUT FILES section describes the files created or modified by the utility.  
920 Temporary or system files that are created for internal usage by this utility or other  
921 parts of the implementation (for example, spool, log, and audit files) are not described  
922 in this, or any, section. The utilities creating such files and the names of such files are  
923 unspecified. If applications are written to use temporary or intermediate files, they  
924 should use the *TMPDIR* environment variable, if it is set and represents an accessible  
925 directory, to select the location of temporary files.

926 Implementations shall ensure that temporary files, when used by the standard utilities,  
927 are named so that different utilities or multiple instances of the same utility can operate  
928 simultaneously without regard to their working directories, or any other process  
929 characteristic other than process ID. There are two exceptions to this rule:

- 930 1. Resources for temporary files other than the name space (for example, disk space,  
931 available directory entries, or number of processes allowed) are not guaranteed.
- 932 2. Certain standard utilities generate output files that are intended as input for other  
933 utilities (for example, *lex* generates *lex.yy.c*), and these cannot have unique  
934 names. These cases are explicitly identified in the descriptions of the respective  
935 utilities.



936 Any temporary file created by the implementation shall be removed by the  
937 implementation upon a utility's successful exit, exit because of errors, or before  
938 termination by any of the SIGHUP, SIGINT, or SIGTERM signals, unless specified  
939 otherwise by the utility description.

940 Receipt of the SIGQUIT signal should generally cause termination (unless in some  
941 debugging mode) that would bypass any attempted recovery actions.

942 Record formats are described in a notation similar to that used by the C-language  
943 function, *printf()*; see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 5,  
944 File Format Notation for a description of this notation.

945 **Default Behavior:** When this section is listed as "None.", it means that no files are  
946 created or modified as a consequence of direct action on the part of the utility when the  
947 utility is used as described by this volume of IEEE Std. 1003.1-200x. However, the  
948 utility may create or modify system files, such as log files, that are outside the utility's  
949 normal execution environment.

#### 950 EXTENDED DESCRIPTION

951 The EXTENDED DESCRIPTION section provides a place for describing the actions of  
952 very complicated utilities, such as text editors or language processors, which typically  
953 have elaborate command languages.

954 **Default Behavior:** When this section is listed as "None.", no further description is  
955 necessary.

#### 956 EXIT STATUS

957 The EXIT STATUS section describes the values the utility shall return to the calling  
958 program, or shell, and the conditions that cause these values to be returned. Usually,  
959 utilities return zero for successful completion and values greater than zero for various  
960 error conditions. If specific numeric values are listed in this section, the system shall  
961 use those values for the errors described. In some cases, status values are listed more  
962 loosely, such as >0. A portable application shall not rely on any specific value in the  
963 range shown and shall be prepared to receive any value in the range.

964 For example, a utility may list zero as a successful return, 1 as a failure for a specific  
965 reason, and >1 as "an error occurred". In this case, unspecified conditions may cause a  
966 2 or 3, or other value, to be returned. A portable application should be written so that it  
967 tests for successful exit status values (zero in this case), rather than relying upon the  
968 single specific error value listed in this volume of IEEE Std. 1003.1-200x. In that way, it  
969 has maximum portability, even on implementations with extensions.

970 Unspecified error conditions may be represented by specific values not listed in this  
971 volume of IEEE Std. 1003.1-200x.

#### 972 CONSEQUENCES OF ERRORS

973 The CONSEQUENCES OF ERRORS section describes the effects on the environment,  
974 file systems, process state, and so on, when error conditions occur. It does not describe  
975 error messages produced or exit status values used.

976 The many reasons for failure of a utility are generally not specified by the utility  
977 descriptions. Utilities may terminate prematurely if they encounter: invalid usage of  
978 options, arguments, or environment variables; invalid usage of the complex syntaxes  
979 expressed in EXTENDED DESCRIPTION sections; difficulties accessing, creating,  
980 reading, or writing files; or difficulties associated with the privileges of the process.

981 The following shall apply to each utility, unless otherwise stated:

982 • If the requested action cannot be performed on an operand representing a file,  
983 directory, user, process, and so on, the utility shall issue a diagnostic message to  
984 standard error and continue processing the next operand in sequence, but the final  
985 exit status shall be returned as non-zero.

986 For a utility that recursively traverses a file hierarchy (such as *find* or *chown -R*), if  
987 the requested action cannot be performed on a file or directory encountered in the  
988 hierarchy, the utility shall issue a diagnostic message to standard error and continue  
989 processing the remaining files in the hierarchy, but the final exit status shall be  
990 returned as non-zero.

991 • If the requested action characterized by an option or option-argument cannot be  
992 performed, the utility shall issue a diagnostic message to standard error and the exit  
993 status returned shall be non-zero.

994 • When an unrecoverable error condition is encountered, the utility shall exit with a  
995 non-zero exit status.

996 • A diagnostic message shall be written to standard error whenever an error  
997 condition occurs.

998 When a utility encounters an error condition several actions are possible, depending on  
999 the severity of the error and the state of the utility. Included in the possible actions of  
1000 various utilities are: deletion of temporary or intermediate work files; deletion of  
1001 incomplete files; validity checking of the file system or directory.

1002 **Default Behavior:** When this section is listed as “Default.”, it means that any changes  
1003 to the environment are unspecified.

#### 1004 APPLICATION USAGE

1005 This section is non-normative.

1006 The APPLICATION USAGE section gives advice to the application programmer or user  
1007 about the way the utility should be used.

#### 1008 EXAMPLES

1009 This section is non-normative.

1010 The EXAMPLES section gives one or more examples of usage, where appropriate. In  
1011 the event of conflict between an example and a normative part of the specification, the  
1012 normative material is to be taken as correct.

1013 In all examples, quoting has been used, showing how sample commands (utility names  
1014 combined with arguments) could be passed correctly to a shell (see *sh*) or as a string to  
1015 the *system()* function defined in the System Interfaces volume of IEEE Std. 1003.1-200x.  
1016 Such quoting would not be used if the utility is invoked using one of the *exec* functions  
1017 defined in the System Interfaces volume of IEEE Std. 1003.1-200x.

#### 1018 RATIONALE

1019 This section is non-normative.

1020 This section contains historical information concerning the contents of this volume of  
1021 IEEE Std. 1003.1-200x and why features were included or discarded by the standard  
1022 developers.

#### 1023 FUTURE DIRECTIONS

1024 This section is non-normative.

1025 The FUTURE DIRECTIONS section should be used as a guide to current thinking; there  
1026 is not necessarily a commitment to implement all of these future directions in their

1027                   entirety.

1028                   **SEE ALSO**

1029                   This section is non-normative.

1030                   The SEE ALSO section lists related entries.

1031                   **CHANGE HISTORY**

1032                   This section is non-normative.

1033                   The CHANGE HISTORY section shows the derivation of the description used by this  
1034                   volume of IEEE Std. 1003.1-200x and lists the functional differences between Issues 4  
1035                   and 6.

1036                   Certain of the standard utilities describe how they can invoke other utilities or applications, such  
1037                   as by passing a command string to the command interpreter. The external influences (STDIN,  
1038                   ENVIRONMENT VARIABLES, and so on) and external effects (STDOUT, CONSEQUENCES OF  
1039                   ERRORS, and so on) of such invoked utilities are not described in the section concerning the  
1040                   standard utility that invokes them.

**1.12 Considerations for Utilities in Support of Files of Arbitrary Size**

The following utilities support files of any size up to the maximum that can be created by the implementation. This support includes correct writing of file size-related values (such as file sizes and offsets, line numbers, and block counts) and correct interpretation of command line arguments that contain such values.

|      |                 |                                                   |
|------|-----------------|---------------------------------------------------|
| 1046 | <i>basename</i> | Return non-directory portion of path name.        |
| 1047 | <i>cat</i>      | Concatenate and print files.                      |
| 1048 | <i>cd</i>       | Change working directory.                         |
| 1049 | <i>chgrp</i>    | Change file group ownership.                      |
| 1050 | <i>chmod</i>    | Change file modes.                                |
| 1051 | <i>chown</i>    | Change file ownership.                            |
| 1052 | <i>cksum</i>    | Write file checksums and sizes.                   |
| 1053 | <i>cmp</i>      | Compare two files.                                |
| 1054 | <i>cp</i>       | Copy files.                                       |
| 1055 | <i>dd</i>       | Convert and copy a file.                          |
| 1056 | <i>df</i>       | Report free disk space.                           |
| 1057 | <i>dirname</i>  | Return directory portion of path name.            |
| 1058 | <i>du</i>       | Estimate file space usage.                        |
| 1059 | <i>find</i>     | Find files.                                       |
| 1060 | <i>ln</i>       | Link files.                                       |
| 1061 | <i>ls</i>       | List directory contents.                          |
| 1062 | <i>mkdir</i>    | Make directories.                                 |
| 1063 | <i>mv</i>       | Move files.                                       |
| 1064 | <i>pathchk</i>  | Check path names.                                 |
| 1065 | <i>pwd</i>      | Return working directory name.                    |
| 1066 | <i>rm</i>       | Remove directory entries.                         |
| 1067 | <i>rmdir</i>    | Remove directories.                               |
| 1068 | <i>sh</i>       | Shell, the standard command language interpreter. |
| 1069 | <i>sum</i>      | Print checksum and block or byte count of a file. |
| 1070 | <i>test</i>     | Evaluate expression.                              |
| 1071 | <i>touch</i>    | Change file access and modification times.        |
| 1072 | <i>ulimit</i>   | Set or report file size limit.                    |

Exceptions to the requirement that utilities support files of any size up to the maximum are as follows:

1. Uses of files as command scripts, or for configuration or control, are exempt. For example, it is not required that *sh* be able to read an arbitrarily large **.profile**.

- 1077            2. Shell input and output redirection are exempt. For example, it is not required that the  
1078            redirections *sum < file* or *echo foo > file* succeed for an arbitrarily large existing file.



# Shell Command Language

1080

1081 This chapter contains the definition of the Shell Command Language.

## 1082 2.1 Shell Introduction

1083 The shell is a command language interpreter. This chapter describes the syntax of that command  
 1084 language as it is used by the *sh* utility and the *system()* and *popen()* functions defined in the  
 1085 System Interfaces volume of IEEE Std. 1003.1-200x.

1086 The shell operates according to the following general overview of operations. The specific  
 1087 details are included in the cited sections of this chapter.

- 1088 1. The shell reads its input from a file (see *sh*), from the `-c` option or from the *system()* and  
 1089 *popen()* functions defined in the System Interfaces volume of IEEE Std. 1003.1-200x. If the  
 1090 first line of a file of shell commands starts with the characters "#!", the results are  
 1091 XSI unspecified. On XSI-conformant systems, if the first two characters of a file are "#!", it  
 1092 shall behave as described for executable scripts in Section 2.10 (on page 2265).
- 1093 2. The shell breaks the input into tokens: words and operators; see Section 2.3 (on page 2238).
- 1094 3. The shell parses the input into simple commands (see Section 2.9.1 (on page 2256)) and  
 1095 compound commands (see Section 2.9.4 (on page 2261)).
- 1096 4. The shell performs various expansions (separately) on different parts of each command,  
 1097 resulting in a list of path names and fields to be treated as a command and arguments; see  
 1098 Section 2.6 (on page 2244).
- 1099 5. The shell performs redirection (see Section 2.7 (on page 2251)) and removes redirection  
 1100 operators and their operands from the parameter list.
- 1101 6. The shell executes a function (see Section 2.9.5 (on page 2263)), built-in (see Section 2.15  
 1102 (on page 2276)), executable file, or script, giving the names of the arguments as positional  
 1103 parameters numbered 1 to *n*, and the name of the command (or in the case of a function  
 1104 within a script, the name of the script) as the positional parameter numbered 0 (see Section  
 1105 2.9.1.1 (on page 2257)).
- 1106 7. The shell optionally waits for the command to complete and collects the exit status (see  
 1107 Section 2.8.2 (on page 2255)).

## 1108 2.2 Quoting

1109 Quoting is used to remove the special meaning of certain characters or words to the shell.  
 1110 Quoting can be used to preserve the literal meaning of the special characters in the next  
 1111 paragraph, prevent reserved words from being recognized as such, and prevent parameter  
 1112 expansion and command substitution within here-document processing (see Section 2.7.4 (on  
 1113 page 2252)).

1114 The application shall quote the following characters if they are to represent themselves:

1115 | & ; < > ( ) \$ ' \ " ' <space> <tab> <newline>

1116 and the following may need to be quoted under certain circumstances. That is, these characters  
 1117 may be special depending on conditions described elsewhere in this volume of  
 1118 IEEE Std. 1003.1-200x:

1119 \* ? [ # ~ = %

1120 The various quoting mechanisms are the escape character, single-quotes, and double-quotes.  
 1121 The here-document represents another form of quoting; see Section 2.7.4 (on page 2252).

### 1122 2.2.1 Escape Character (Backslash)

1123 A backslash that is not quoted shall preserve the literal value of the following character, with the  
 1124 exception of a <newline> character. If a <newline> character follows the backslash, the shell  
 1125 shall interpret this as line continuation. The backslash and <newline> characters shall be  
 1126 removed before splitting the input into tokens. Since the escaped <newline> character is  
 1127 removed entirely from the input and is not replaced by any white space, it cannot serve as a  
 1128 token separator.

### 1129 2.2.2 Single-Quotes

1130 Enclosing characters in single-quotes ( ' ' ) shall preserve the literal value of each character  
 1131 within the single-quotes. A single-quote cannot occur within single-quotes.

### 1132 2.2.3 Double-Quotes

1133 Enclosing characters in double-quotes ( " " ) shall preserve the literal value of all characters  
 1134 within the double-quotes, with the exception of the characters dollar sign, backquote, and  
 1135 backslash, as follows:

1136 \$ The dollar sign shall retain its special meaning introducing parameter expansion (see  
 1137 Section 2.6.2 (on page 2245)), a form of command substitution (see Section 2.6.3 (on page  
 1138 2247)), and arithmetic expansion (see Section 2.6.4 (on page 2248)).

1139 The input characters within the quoted string that are also enclosed between "\$(" and the  
 1140 matching ')' is not affected by the double-quotes, but rather shall define that command  
 1141 whose output replaces the "\$(...)" when the word is expanded. The tokenizing rules in  
 1142 Section 2.3 (on page 2238) shall be applied recursively to find the matching ')'.  
 1143

1144 Within the string of characters from an enclosed "\${" to the matching '}', an even number  
 1145 of unescaped double-quotes or single-quotes, if any, shall occur. A preceding backslash  
 1146 character shall be used to escape a literal '{' or '}'. The rule in Section 2.6.2 (on page  
 2245) shall be used to determine the matching '}'.

1147 ` The backquote shall retain its special meaning introducing the other form of command  
 1148 substitution (see Section 2.6.3 (on page 2247)). The portion of the quoted string from the  
 1149 initial backquote and the characters up to the next backquote that is not preceded by a



1150           backslash, having escape characters removed, defines that command whose output replaces  
1151           " ` . . . ` " when the word is expanded. Either of the following cases produces undefined  
1152           results:

1153           • A single-quoted or double-quoted string that begins, but does not end, within the  
1154           " ` . . . ` " sequence

1155           • A " ` . . . ` " sequence that begins, but does not end, within the same double-quoted  
1156           string

1157       \   The backslash shall retain its special meaning as an escape character (see Section 2.2.1 (on  
1158       page 2236)) only when followed by one of the following characters when considered special:

1159           \$   `   "   \   <newline>

1160       The application shall ensure that a double-quote is preceded by a backslash to be included  
1161       within double-quotes. The parameter '@' has special meaning inside double-quotes and is  
1162       described in Section 2.5.2 (on page 2241).

## 1163 2.3 Token Recognition

1164 The shell reads its input in terms of lines from a file, from a terminal in the case of an interactive  
 1165 shell, or from a string in the case of *sh -c* or *system()*. The input lines can be of unlimited length.  
 1166 These lines are parsed using two major modes: ordinary token recognition and processing of  
 1167 here-documents.

1168 When an **io\_here** token has been recognized by the grammar (see Section 2.11 (on page 2266)),  
 1169 one or more of the subsequent lines immediately following the next **NEWLINE** token form the  
 1170 body of one or more here-documents and shall be parsed according to the rules of Section 2.7.4  
 1171 (on page 2252).

1172 When it is not processing an **io\_here**, the shell shall break its input into tokens by applying the  
 1173 first applicable rule below to the next character in its input. The token shall be from the current  
 1174 position in the input until a token is delimited according to one of the rules below; the characters  
 1175 forming the token are exactly those in the input, including any quoting characters. If it is  
 1176 indicated that a token is delimited, and no characters have been included in a token, processing  
 1177 shall continue until an actual token is delimited.

- 1178 1. If the end of input is recognized, the current token shall be delimited. If there is no current  
 1179 token, the end-of-input indicator shall be returned as the token.
- 1180 2. If the previous character was used as part of an operator and the current character is not  
 1181 quoted and can be used with the current characters to form an operator, it shall be used as  
 1182 part of that (operator) token.

1183 On some systems, the symbol "`( (`" is a control operator; its use produces unspecified  
 1184 results. Applications that wish to have nested subshells, such as:

```
1185 ((echo Hello);(echo World))
```

1186 shall separate the "`( (`" characters into two tokens by including white space between them.  
 1187 Some systems may treat these as invalid arithmetic expressions instead of subshells.

1188 Certain combinations of characters are invalid in portable scripts, as shown in the  
 1189 grammar, and that some systems have assigned these combinations (such as "`|&`") as  
 1190 valid control operators. Portable scripts cannot rely on receiving errors in all cases where  
 1191 this volume of IEEE Std. 1003.1-200x indicates that a syntax is invalid.

- 1192 3. If the previous character was used as part of an operator and the current character cannot  
 1193 be used with the current characters to form an operator, the operator containing the  
 1194 previous character shall be delimited.
- 1195 4. If the current character is backslash, single-quote, or double-quote (`'\'`, `'\''`, or `'\"'`)  
 1196 and it is not quoted, it shall affect quoting for subsequent characters up to the end of the  
 1197 quoted text. The rules for quoting are as described in Section 2.2 (on page 2236). During  
 1198 token recognition no substitutions shall be actually performed, and the result token shall  
 1199 contain exactly the characters that appear in the input (except for `<newline>` character  
 1200 joining), unmodified, including any embedded or enclosing quotes or substitution  
 1201 operators, between the quote mark and the end of the quoted text. The token shall not be  
 1202 delimited by the end of the quoted field.
- 1203 5. If the current character is an unquoted `'$'` or `'\''`, the shell shall identify the start of any  
 1204 candidates for parameter expansion (Section 2.6.2 (on page 2245)), command substitution  
 1205 (Section 2.6.3 (on page 2247)), or arithmetic expansion (Section 2.6.4 (on page 2248)) from  
 1206 their introductory unquoted character sequences: `'$'` or `"${"`, `"$(` or `'\''`, and `"$(` (`"`,  
 1207 respectively. The shell shall read sufficient input to determine the end of the unit to be  
 1208 expanded (as explained in the cited sections). While processing the characters, if instances

- 1209 of expansions or quoting are found nested within the substitution, the shell shall  
 1210 recursively process them in the manner specified for the construct that is found. The  
 1211 characters found from the beginning of the substitution to its end, allowing for any  
 1212 recursion necessary to recognize embedded constructs, shall be included unmodified in the  
 1213 result token, including any embedded or enclosing substitution operators or quotes. The  
 1214 token shall not be delimited by the end of the substitution.
- 1215 6. If the current character is not quoted and can be used as the first character of a new  
 1216 operator, the current token (if any) shall be delimited. The current character shall be used  
 1217 as the beginning of the next (operator) token.
  - 1218 7. If the current character is an unquoted <newline> character, the current token shall be  
 1219 delimited.
  - 1220 8. If the current character is an unquoted <blank> character, any token containing the  
 1221 previous character is delimited and the current character shall be discarded.
  - 1222 9. If the previous character was part of a word, the current character shall be appended to  
 1223 that word.
  - 1224 10. If the current character is a '#', it and all subsequent characters up to, but excluding, the  
 1225 next <newline> character shall be discarded as a comment. The <newline> character that  
 1226 ends the line is not considered part of the comment.
  - 1227 11. The current character is used as the start of a new word.
- 1228 Once a token is delimited, it is categorized as required by the grammar in Section 2.11 (on page  
 1229 2266).

### 1230 2.3.1 Alias Substitution

1231 UP XSI The processing of aliases shall be supported on all XSI-conformant systems or if the system  
 1232 supports the User Portability Utilities option (and the rest of this section is not further shaded for  
 1233 these options).

1234 After a token has been delimited, but before applying the grammatical rules in Section 2.11 (on  
 1235 page 2266), a resulting word that is identified to be the command name word of a simple  
 1236 command shall be examined to determine whether it is an unquoted, valid alias name. However,  
 1237 reserved words in correct grammatical context shall not be candidates for alias substitution. A  
 1238 valid alias name (see the Base Definitions volume of IEEE Std. 1003.1-200x, Section 3.10, Alias  
 1239 Name) shall be one that has been defined by the *alias* utility and not subsequently undefined  
 1240 using *unalias*. Implementations also may provide predefined valid aliases that are in effect when  
 1241 the shell is invoked. To prevent infinite loops in recursive aliasing, if the shell is not currently  
 1242 processing an alias of the same name, the word shall be replaced by the value of the alias;  
 1243 otherwise, it shall not be replaced.

1244 If the value of the alias replacing the word ends in a <blank> character, the shell shall check the  
 1245 next command word for alias substitution; this process shall continue until a word is found that  
 1246 is not a valid alias or an alias value does not end in a <blank> character.

1247 When used as specified by this volume of IEEE Std. 1003.1-200x, alias definitions shall not be  
 1248 inherited by separate invocations of the shell or by the utility execution environments invoked  
 1249 by the shell; see Section 2.13 (on page 2273).

1250 **2.4 Reserved Words**

1251 Reserved words are words that have special meaning to the shell; see Section 2.9 (on page 2256).  
 1252 The following words shall be recognized as reserved words:

|      |             |             |             |              |
|------|-------------|-------------|-------------|--------------|
| 1253 | <b>!</b>    | <b>do</b>   | <b>esac</b> | <b>in</b>    |
| 1254 | <b>{</b>    | <b>done</b> | <b>fi</b>   | <b>then</b>  |
| 1255 | <b>}</b>    | <b>elif</b> | <b>for</b>  | <b>until</b> |
| 1256 | <b>case</b> | <b>else</b> | <b>if</b>   | <b>while</b> |

1257 This recognition shall only occur when none of the characters is quoted and when the word is  
 1258 used as:

- 1259 • The first word of a command
- 1260 • The first word following one of the reserved words other than **case**, **for**, or **in**
- 1261 • The third word in a **case** or **for** command (only **in** is valid in this case)

1262 See the grammar in Section 2.11 (on page 2266).

1263 The following words may be recognized as reserved words on some systems (when none of the  
 1264 characters are quoted), causing unspecified results:

|      |           |           |                 |               |
|------|-----------|-----------|-----------------|---------------|
| 1265 | <b>[[</b> | <b>]]</b> | <b>function</b> | <b>select</b> |
|------|-----------|-----------|-----------------|---------------|

1266 Words that are the concatenation of a name and a colon (':') are reserved; their use produces  
 1267 unspecified results. This reservation is to allow future implementations that support named  
 1268 labels for flow control.

1269 **2.5 Parameters and Variables**

1270 A parameter can be denoted by a name, a number, or one of the special characters listed in  
1271 Section 2.5.2. A variable is a parameter denoted by a name.

1272 A parameter is set if it has an assigned value (null is a valid value). Once a variable is set, it can  
1273 only be unset by using the *unset* special built-in command.

1274 **2.5.1 Positional Parameters**

1275 A positional parameter is a parameter denoted by the decimal value represented by one or more  
1276 digits, other than the single digit 0. The digits denoting the positional parameters shall always be  
1277 interpreted as a decimal value, even if there is a leading zero. When a positional parameter with  
1278 more than one digit is specified, the application shall enclose the digits in braces (see Section  
1279 2.6.2 (on page 2245)). Positional parameters are initially assigned when the shell is invoked (see  
1280 *sh*), temporarily replaced when a shell function is invoked (see Section 2.9.5 (on page 2263)), and  
1281 can be reassigned with the *set* special built-in command.

1282 **2.5.2 Special Parameters**

1283 Listed below are the special parameters and the values to which they shall expand. Only the  
1284 values of the special parameters are listed; see Section 2.6 (on page 2244) for a detailed summary  
1285 of all the stages involved in expanding words.

1286 @ Expands to the positional parameters, starting from one. When the expansion occurs within  
1287 double-quotes, and where field splitting (see Section 2.6.5 (on page 2249)) is performed,  
1288 each positional parameter expands as a separate field, with the provision that the expansion  
1289 of the first parameter is still joined with the beginning part of the original word (assuming  
1290 that the expanded parameter was embedded within a word), and the expansion of the last  
1291 parameter is still joined with the last part of the original word. If there are no positional  
1292 parameters, the expansion of '@' generates zero fields, even when '@' is double-quoted.

1293 \* Expands to the positional parameters, starting from one. When the expansion occurs within  
1294 a double-quoted string (see Section 2.2.3 (on page 2236)), it expands to a single field with the  
1295 value of each parameter separated by the first character of the *IFS* variable, or by a <space>  
1296 character if *IFS* is unset. If *IFS* is set to a null string, this is not equivalent to unsetting it; its  
1297 first character does not exist, so the parameter values are concatenated.

1298 # Expands to the decimal number of positional parameters. The command name (parameter  
1299 0) is not counted in the number given by '#' because it is a special parameter, not a  
1300 positional parameter.

1301 ? Expands to the decimal exit status of the most recent pipeline (see Section 2.9.2 (on page  
1302 2258)).

1303 – (Hyphen.) Expands to the current option flags (the single-letter option names concatenated  
1304 into a string) as specified on invocation by the *set* special built-in command or implicitly by  
1305 the shell.

1306 \$ Expands to the decimal process ID of the invoked shell. In a subshell (see Section 2.13 (on  
1307 page 2273)), '\$' shall expand to the same value as that of the current shell.

1308 ! Expands to the decimal process ID of the most recent background command (see Section  
1309 2.9.3 (on page 2259)) executed from the current shell. (For example, background commands  
1310 executed from subshells do not affect the value of "\$!" in the current shell environment.)  
1311 For a pipeline, the process ID is that of the last command in the pipeline.

1312 0 (Zero.) Expands to the name of the shell or shell script. See *sh* (on page 3060) for a detailed  
1313 description of how this name is derived.

1314 See the description of the *IFS* variable in Section 2.5.3.

### 1315 2.5.3 Shell Variables

1316 Variables shall be initialized from the environment (as defined by the Base Definitions volume of  
1317 IEEE Std. 1003.1-200x, Chapter 8, Environment Variables and the *exec* function in the System  
1318 Interfaces volume of IEEE Std. 1003.1-200x) and can be given new values with variable  
1319 assignment commands. If a variable is initialized from the environment, it shall be marked for  
1320 export immediately; see the *export* special built-in. New variables can be defined and initialized  
1321 with variable assignments, with the *read* or *getopts* utilities, with the *name* parameter in a *for*  
1322 loop, with the  $\${name=word}$  expansion, or with other mechanisms provided as implementation  
1323 extensions.

1324 The following variables shall affect the execution of the shell.

1325 *ENV* This variable, when and only when an interactive shell is invoked, shall be  
1326 subjected to parameter expansion (see Section 2.6.2 (on page 2245)) by the  
1327 shell and the resulting value shall be used as a path name of a file containing  
1328 shell commands to execute in the current environment. The file need not be  
1329 executable. If the expanded value of *ENV* is not an absolute path name, the  
1330 results are unspecified. *ENV* shall be ignored if the user's real and effective  
1331 user IDs or real and effective group IDs are different.

1332 UP XSI The processing of the *ENV* shell variable shall be supported on all XSI-  
1333 conformant systems or if the system supports the User Portability Utilities  
1334 option.

1335 *HOME* This variable shall be interpreted as the path name of the user's home  
1336 directory. The contents of *HOME* are used in tilde expansion (see Section 2.6.1  
1337 (on page 2244)).

1338 *IFS* (Input Field Separators.) A string treated as a list of characters that is used for  
1339 field splitting and to split lines into fields with the *read* command. If *IFS* is not  
1340 set, the shell shall behave as if the value of *IFS* were the <space>, <tab>, and  
1341 <newline> characters; see Section 2.6.5 (on page 2249).

1342 *LANG* This variable shall provide a default value for the internationalization  
1343 variables that are unset or null. If *LANG* is unset or null, the corresponding  
1344 value from the implementation-defined default locale is used. If any of the  
1345 internationalization variables contains an invalid setting, the utility behaves as  
1346 if none of the variables had been defined.

1347 *LC\_ALL* This variable shall provide a default value for the *LC\_\** variables, as described  
1348 in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 8,  
1349 Environment Variables.

1350 *LC\_COLLATE* This variable shall determine the behavior of range expressions, equivalence  
1351 classes, and multi-character collating elements within pattern matching.

1352 *LC\_CTYPE* This variable shall determine the interpretation of sequences of bytes of text  
1353 data as characters (for example, single-byte as opposed to multi-byte  
1354 characters), which characters are defined as letters (character class **alpha**) and  
1355 <blank> characters (character class **blank**), and the behavior of character  
1356 classes within pattern matching. Changing the value of *LC\_CTYPE* after the  
1357 shell has started shall not affect the lexical processing of shell commands in

|      |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1358 |                    | the current shell execution environment or its subshells. Invoking a shell script or performing <i>exec sh</i> subjects the new shell to the changes in <i>LC_CTYPE</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 1359 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1360 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1361 | <i>LC_MESSAGES</i> | This variable shall determine the language in which messages should be written.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1362 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1363 | <i>LINENO</i>      | This variable shall be set by the shell to a decimal number representing the current sequential line number (numbered starting with 1) within a script or function before it executes each command. If the user unsets or resets <i>LINENO</i> , the variable may lose its special meaning for the life of the shell. If the shell is not currently executing a script or function, the value of <i>LINENO</i> is unspecified. This volume of IEEE Std. 1003.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.                                                                                                                                                                                                            |
| 1364 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1365 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1366 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1367 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1368 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1369 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1370 | XSI <i>NLSPATH</i> | This variable shall determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 1371 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1372 | <i>PATH</i>        | This variable represents a string formatted as described in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables, used to effect command interpretation; see Section 2.9.1.1 (on page 2257).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 1373 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1374 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1375 | <i>PPID</i>        | This variable shall be set by the shell to the decimal process ID of the process that invoked this shell. In a subshell (see Section 2.13 (on page 2273)), <i>PPID</i> shall be set to the same value as that of the parent of the current shell. For example, <i>echo\$PPID</i> and ( <i>echo\$PPID</i> ) would produce the same value. This volume of IEEE Std. 1003.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.                                                                                                                                                                                                                                                                                                  |
| 1376 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1377 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1378 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1379 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1380 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1381 | <i>PS1</i>         | Each time an interactive shell is ready to read a command, the value of this variable shall be subjected to parameter expansion and written to standard error. The default value shall be "\$ ". For users who have specific additional implementation-defined privileges, the default may be another, implementation-defined value. (Historically, the superuser has had a prompt of '#'.) The shell shall replace each instance of the character '!' in <i>PS1</i> with the history file number of the next command to be typed. Escaping the '!' with another '!' (that is, "!!") shall place the literal character '!' in the prompt. This volume of IEEE Std. 1003.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option. |
| 1382 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1383 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1384 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1385 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1386 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1387 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1388 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1389 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1390 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1391 | <i>PS2</i>         | Each time the user enters a <newline> character prior to completing a command line in an interactive shell, the value of this variable shall be subjected to parameter expansion and written to standard error. The default value is "> ". This volume of IEEE Std. 1003.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.                                                                                                                                                                                                                                                                                                                                                                                                |
| 1392 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1393 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1394 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1395 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1396 | <i>PS4</i>         | When an execution trace ( <i>set -x</i> ) is being performed in an interactive shell, before each line in the execution trace, the value of this variable shall be subjected to parameter expansion and written to standard error. The default value is "+ ". This volume of IEEE Std. 1003.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.                                                                                                                                                                                                                                                                                                                                                                             |
| 1397 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1398 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1399 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1400 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1401 | <i>PWD</i>         | This variable shall be set by the shell to be an absolute path name of the current working directory, containing no components of type symbolic link, no components that are dot, and no components that are dot-dot when the shell is initialized. If an application sets or unsets the value of <i>PWD</i> , the behaviors of the <i>cd</i> and <i>pwd</i> utilities are unspecified.                                                                                                                                                                                                                                                                                                                                                                                                 |
| 1402 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1403 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1404 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1405 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

## 1406 2.6 Word Expansions

1407 This section describes the various expansions that are performed on words. Not all expansions  
1408 are performed on every word, as explained in the following sections.

1409 Tilde expansions, parameter expansions, command substitutions, arithmetic expansions, and  
1410 quote removals that occur within a single word expand to a single field. It is only field splitting  
1411 or path name expansion that can create multiple fields from a single word. The single exception  
1412 to this rule is the expansion of the special parameter '@' within double-quotes, as described in  
1413 Section 2.5.2 (on page 2241).

1414 The order of word expansion shall be as follows:

- 1415 1. Tilde expansion (see Section 2.6.1), parameter expansion (see Section 2.6.2 (on page 2245)),  
1416 command substitution (see Section 2.6.3 (on page 2247)), and arithmetic expansion (see  
1417 Section 2.6.4 (on page 2248)) shall be performed, beginning to end. See item 5 in Section 2.3  
1418 (on page 2238).
- 1419 2. Field splitting (see Section 2.6.5 (on page 2249)) shall be performed on the portions of the  
1420 fields generated by step 1, unless *IFS* is null.
- 1421 3. Path name expansion (see Section 2.6.6 (on page 2249)) shall be performed, unless *set -f* is  
1422 in effect.
- 1423 4. Quote removal (see Section 2.6.7 (on page 2250)) shall always be performed last.

1424 The expansions described in this section shall occur in the same shell environment as that in  
1425 which the command is executed.

1426 If the complete expansion appropriate for a word results in an empty field, that empty field shall  
1427 be deleted from the list of fields that form the completely expanded command, unless the  
1428 original word contained single-quote or double-quote characters.

1429 The '\$' character is used to introduce parameter expansion, command substitution, or  
1430 arithmetic evaluation. If an unquoted '\$' is followed by a character that is either not numeric,  
1431 the name of one of the special parameters (see Section 2.5.2 (on page 2241)), a valid first  
1432 character of a variable name, a left curly brace ('{') or a left parenthesis, the result is  
1433 unspecified.

### 1434 2.6.1 Tilde Expansion

1435 A *tilde-prefix* consists of an unquoted tilde character at the beginning of a word, followed by all  
1436 of the characters preceding the first unquoted slash in the word, or all the characters in the word  
1437 if there is no slash. In an assignment (see the Base Definitions volume of IEEE Std. 1003.1-200x,  
1438 Section 4.16, Variable Assignment), multiple tilde-prefixes can be used: at the beginning of the  
1439 word (that is, following the equal sign of the assignment), following any unquoted colon, or  
1440 both. A tilde-prefix in an assignment is terminated by the first unquoted colon or slash. If none  
1441 of the characters in the tilde-prefix are quoted, the characters in the tilde-prefix following the  
1442 tilde are treated as a possible login name from the user database. A portable login name cannot  
1443 contain characters outside the set given in the description of the *LOGNAME* environment  
1444 variable in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 8.3, Other Environment  
1445 Variables. If the login name is null (that is, the tilde-prefix contains only the tilde), the tilde-  
1446 prefix is replaced by the value of the variable *HOME*. If *HOME* is unset, the results are  
1447 unspecified. Otherwise, the tilde-prefix is replaced by a path name of the initial working  
1448 directory associated with the login name obtained using the *getpwnam()* function as defined in  
1449 the System Interfaces volume of IEEE Std. 1003.1-200x. If the system does not recognize the login  
1450 name, the results are undefined.



1451 **2.6.2 Parameter Expansion**

1452 The format for parameter expansion is as follows:

1453  $\${expression}$ 1454 where *expression* consists of all characters until the matching '}'. Any '}' escaped by a  
1455 backslash or within a quoted string, and characters in embedded arithmetic expansions,  
1456 command substitutions, and variable expansions, shall not be examined in determining the  
1457 matching '}'.

1458 The simplest form for parameter expansion is:

1459  $\${parameter}$ 1460 The value, if any, of *parameter* shall be substituted.1461 The parameter name or symbol can be enclosed in braces, which are optional except for  
1462 positional parameters with more than one digit or when *parameter* is followed by a character that  
1463 could be interpreted as part of the name. The matching closing brace shall be determined by  
1464 counting brace levels, skipping over enclosed quoted strings, and command substitutions.1465 If the parameter name or symbol is not enclosed in braces, the expansion shall use the longest  
1466 valid name (see the Base Definitions volume of IEEE Std. 1003.1-200x, Section 3.232, Name),  
1467 whether or not the symbol represented by that name exists.

1468 If a parameter expansion occurs inside double-quotes:

- 1469
- Path name expansion shall not be performed on the results of the expansion.
  - Field splitting shall not be performed on the results of the expansion, with the exception of  
1470 '@'; see Section 2.5.2 (on page 2241).
- 1471

1472 In addition, a parameter expansion can be modified by using one of the following formats. In  
1473 each case that a value of *word* is needed (based on the state of *parameter*, as described below),  
1474 *word* shall be subjected to tilde expansion, parameter expansion, command substitution, and  
1475 arithmetic expansion. If *word* is not needed, it shall not be expanded. The '}' character that  
1476 delimits the following parameter expansion modifications shall be determined as described  
1477 previously in this section and in Section 2.2.3 (on page 2236). (For example,  $\{\mathbf{foo-bar}\mathbf{xyz}$   
1478 would result in the expansion of **foo** followed by the string **xyz** if **foo** is set, else the string  
1479 "barxyz").1480  $\${parameter}:-word$  **Use Default Values.** If *parameter* is unset or null, the expansion of *word*  
1481 shall be substituted; otherwise, the value of *parameter* shall be substituted.1482  $\${parameter}:=word$  **Assign Default Values.** If *parameter* is unset or null, the expansion of  
1483 *word* shall be assigned to *parameter*. In all cases, the final value of  
1484 *parameter* shall be substituted. Only variables, not positional parameters  
1485 or special parameters, can be assigned in this way.1486  $\${parameter}?:[word]$  **Indicate Error if Null or Unset.** If *parameter* is unset or null, the  
1487 expansion of *word* (or a message indicating it is unset if *word* is omitted)  
1488 shall be written to standard error and the shell exits with a non-zero exit  
1489 status. Otherwise, the value of *parameter* shall be substituted. An  
1490 interactive shell need not exit.1491  $\${parameter}:+word$  **Use Alternative Value.** If *parameter* is unset or null, null shall be  
1492 substituted; otherwise, the expansion of *word* shall be substituted.1493 In the parameter expansions shown previously, use of the colon in the format results in a test for  
1494 a parameter that is unset or null; omission of the colon results in a test for a parameter that is

1495 only unset. The following table summarizes the effect of the colon:

|                            | <i>parameter</i><br><b>Set and Not Null</b> | <i>parameter</i><br><b>Set But Null</b> | <i>parameter</i><br><b>Unset</b> |
|----------------------------|---------------------------------------------|-----------------------------------------|----------------------------------|
| 1496 $\${parameter:-word}$ | substitute <i>parameter</i>                 | substitute <i>word</i>                  | substitute <i>word</i>           |
| 1497 $\${parameter-word}$  | substitute <i>parameter</i>                 | substitute null                         | substitute <i>word</i>           |
| 1498 $\${parameter:=word}$ | substitute <i>parameter</i>                 | assign <i>word</i>                      | assign <i>word</i>               |
| 1499 $\${parameter=word}$  | substitute <i>parameter</i>                 | substitute <i>parameter</i>             | assign null                      |
| 1500 $\${parameter?word}$  | substitute <i>parameter</i>                 | error, exit                             | error, exit                      |
| 1501 $\${parameter?word}$  | substitute <i>parameter</i>                 | substitute null                         | error, exit                      |
| 1502 $\${parameter+word}$  | substitute <i>word</i>                      | substitute null                         | substitute null                  |
| 1503 $\${parameter+word}$  | substitute <i>word</i>                      | substitute <i>word</i>                  | substitute null                  |

1506 In all cases shown with “substitute”, the expression is replaced with the value shown. In all  
1507 cases shown with “assign”, *parameter* is assigned that value, which also replaces the expression.

1508  $\${#parameter}$  **String Length.** The length in characters of the value of *parameter* shall be  
1509 substituted. If *parameter* is '\*' or '@', the result of the expansion is  
1510 unspecified.

1511 The following four varieties of parameter expansion provide for substring processing. In each  
1512 case, pattern matching notation (see Section 2.14 (on page 2274)), rather than regular expression  
1513 notation, shall be used to evaluate the patterns. If *parameter* is '\*' or '@', the result of the  
1514 expansion is unspecified. Enclosing the full parameter expansion string in double-quotes shall  
1515 not cause the following four varieties of pattern characters to be quoted, whereas quoting  
1516 characters within the braces shall have this effect.

1517  $\${parameter%word}$  **Remove Smallest Suffix Pattern.** The *word* is expanded to produce a  
1518 pattern. The parameter expansion then results in *parameter*, with the  
1519 smallest portion of the suffix matched by the *pattern* deleted.

1520  $\${parameter%%word}$  **Remove Largest Suffix Pattern.** The *word* shall be expanded to produce a  
1521 pattern. The parameter expansion then results in *parameter*, with the  
1522 largest portion of the suffix matched by the *pattern* deleted.

1523  $\${parameter#word}$  **Remove Smallest Prefix Pattern.** The *word* shall be expanded to produce  
1524 a pattern. The parameter expansion then results in *parameter*, with the  
1525 smallest portion of the prefix matched by the *pattern* deleted.

1526  $\${parameter##word}$  **Remove Largest Prefix Pattern.** The *word* shall be expanded to produce a  
1527 pattern. The parameter expansion then results in *parameter*, with the  
1528 largest portion of the prefix matched by the *pattern* deleted.

## 1529 Examples

1530  $\${parameter:-word}$   
1531 In this example, *ls* is executed only if *x* is null or unset. (The  $\$(ls)$  command substitution  
1532 notation is explained in Section 2.6.3 (on page 2247).)

```
1533 ${x:-$(ls)}
```

```
1534 ${parameter:=word}
1535 unset X
1536 echo ${X:=abc}
1537 abc
```

```
1538 ${parameter:?word}
1539 unset posix
```

```
1540 echo ${posix:?}
1541 sh: posix: parameter null or not set
```

```
1542 ${parameter:+word}
1543 set a b c
1544 echo ${3:+posix}
1545 posix
```

```
1546 ${#parameter}
1547 HOME=/usr/posix
1548 echo ${#HOME}
1549 10
```

```
1550 ${parameter%word}
1551 x=file.c
1552 echo ${x%.c}.o
1553 file.o
```

```
1554 ${parameter%%word}
1555 x=posix/src/std
1556 echo ${x%%/*}
1557 posix
```

```
1558 ${parameter#word}
1559 x=$HOME/src/cmd
1560 echo ${x#$HOME}
1561 /src/cmd
```

```
1562 ${parameter##word}
1563 x=/one/two/three
1564 echo ${x##*/}
1565 three
```

1566 The double-quoting of patterns is different depending on where the double-quotes are placed:

1567 "\$ {x#\*} " The asterisk is a pattern character. |

1568 \${x#"\*" } The literal asterisk is quoted and not special. |

### 1569 2.6.3 Command Substitution

1570 Command substitution allows the output of a command to be substituted in place of the  
1571 command name itself. Command substitution shall occur when the command is enclosed as  
1572 follows:

```
1573 $(command)
```

1574 or (backquoted version):

```
1575 ` command `
```

1576 The shell shall expand the command substitution by executing *command* in a subshell  
1577 environment (see Section 2.13 (on page 2273)) and replacing the command substitution (the text  
1578 of *command* plus the enclosing "\$ ( ) " or backquotes) with the standard output of the command,  
1579 removing sequences of one or more <newline> characters at the end of the substitution.  
1580 Embedded <newline> characters before the end of the output shall not be removed; however,  
1581 they may be treated as field delimiters and eliminated during field splitting, depending on the  
1582 value of *IFS* and quoting that is in effect.

1583 Within the backquoted style of command substitution, backslash shall retain its literal meaning,  
 1584 except when followed by: '\$', '`', or '\\` (dollar sign, backquote, backslash). The search for  
 1585 the matching backquote shall be satisfied by the first backquote found without a preceding  
 1586 backslash; during this search, if a non-escaped backquote is encountered within a shell  
 1587 comment, a here-document, an embedded command substitution of the \$(*command*) form, or a  
 1588 quoted string, undefined results occur. A single-quoted or double-quoted string that begins, but  
 1589 does not end, within the "`...`" sequence produces undefined results.

1590 With the \$(*command*) form, all characters following the open parenthesis to the matching closing  
 1591 parenthesis constitute the *command*. Any valid shell script can be used for *command*, except:

- 1592 • A script consisting solely of redirections produces unspecified results
- 1593 • See the restriction on single subshells described below

1594 The results of command substitution shall not be processed for further tilde expansion,  
 1595 parameter expansion, command substitution, or arithmetic expansion. If a command  
 1596 substitution occurs inside double-quotes, it shall not be performed on the results of the  
 1597 substitution.

1598 Command substitution can be nested. To specify nesting within the backquoted version, the  
 1599 application shall precede the inner backquotes with backslashes, for example:

```
1600 \ `command` \ `
```

1601 If the command substitution consists of a single subshell, such as:

```
1602 $((command))
```

1603 a portable application shall separate the "\$(" and "((" into two tokens (that is, separate them  
 1604 with white space). This is required to avoid any ambiguities with arithmetic expansion.

#### 1605 2.6.4 Arithmetic Expansion

1606 Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and  
 1607 substituting its value. The format for arithmetic expansion shall be as follows:

```
1608 $((expression))
```

1609 The expression shall be treated as if it were in double-quotes, except that a double-quote inside  
 1610 the expression is not treated specially. The shell expands all tokens in the expression for  
 1611 parameter expansion, command substitution, and quote removal.

1612 Next, the shell shall treat this as an arithmetic expression and substitutes the value of the  
 1613 expression. The arithmetic expression shall be processed according to the rules of the ISO C  
 1614 standard, with the following exceptions:

- 1615 • Only integer arithmetic is required.
- 1616 • The *sizeof*() operator and the prefix and postfix "++" and "--" operators are not required.
- 1617 • Selection, iteration, and jump statements are not supported.

1618 As an extension, the shell may recognize arithmetic expressions beyond those listed. If the  
 1619 expression is invalid, the expansion fails and the shell shall write a message to standard error  
 1620 indicating the failure.

1621 **Examples**

1622 A simple example using arithmetic expansion:

```

1623 # repeat a command 100 times
1624 x=100
1625 while [$x -gt 0]
1626 do
1627 command
1628 x=$((x-1))
1629 done

```

1630 **2.6.5 Field Splitting**

1631 After parameter expansion (Section 2.6.2 (on page 2245)), command substitution (Section 2.6.3  
 1632 (on page 2247)), and arithmetic expansion (Section 2.6.4 (on page 2248)), the shell shall scan the  
 1633 results of expansions and substitutions that did not occur in double-quotes for field splitting and  
 1634 multiple fields can result.

1635 The shell shall treat each character of the *IFS* as a delimiter and uses the delimiters to split the  
 1636 results of parameter expansion and command substitution into fields.

1637 1. If the value of *IFS* is a <space>, <tab>, and <newline> character, or if it is unset, any  
 1638 sequence of <space>, <tab>, or <newline> characters at the beginning or end of the input  
 1639 shall be ignored and any sequence of those characters within the input shall delimit a field.  
 1640 For example, the input:

```
1641 <newline><space><tab>foo<tab><tab>bar<space>
```

1642 yields two fields, **foo** and **bar**.

1643 2. If the value of *IFS* is null, no field splitting shall be performed.

1644 3. Otherwise, the following rules shall be applied in sequence. The term “*IFS* white space” is  
 1645 used to mean any sequence (zero or more instances) of white space characters that are in  
 1646 the *IFS* value (for example, if *IFS* contains <space>/<comma>/<tab>, any sequence of  
 1647 <space> and <tab> characters is considered *IFS* white space).

1648 a. *IFS* white space shall be ignored at the beginning and end of the input.

1649 b. Each occurrence in the input of an *IFS* character that is not *IFS* white space, along  
 1650 with any adjacent *IFS* white space, shall delimit a field, as described previously.

1651 c. Non-zero-length *IFS* white space shall delimit a field.

1652 **2.6.6 Path Name Expansion**

1653 After field splitting, if *set -f* is not in effect, each field in the resulting command line shall be  
 1654 expanded using the algorithm described in Section 2.14 (on page 2274), qualified by the rules in  
 1655 Section 2.14.3 (on page 2275).

1656 **2.6.7 Quote Removal**

1657 The quote characters: '\', '\'', and '"' (backslash, single-quote, double-quote) that were |  
1658 present in the original word shall be removed unless they have themselves been quoted.

1659 **2.7 Redirection**

1660 Redirection is used to open and close files for the current shell execution environment (see  
1661 Section 2.13 (on page 2273)) or for any command. *Redirection operators* can be used with numbers  
1662 representing file descriptors (see the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
1663 3.167, File Descriptor) as described below.

1664 The overall format used for redirection is:

```
1665 [n]redir-op word
```

1666 The number *n* is an optional decimal number designating the file descriptor number; the  
1667 application shall ensure it is delimited from any preceding text and immediately precede the  
1668 redirection operator *redir-op*. If *n* is quoted, the number shall not be recognized as part of the  
1669 redirection expression. For example:

```
1670 echo \2>a
```

1671 writes the character 2 into file a. If any part of *redir-op* is quoted, no redirection expression is  
1672 recognized. For example:

```
1673 echo 2\>a
```

1674 writes the characters 2>a to standard output. The optional number, redirection operator, and  
1675 *word* shall not appear in the arguments provided to the command to be executed (if any).

1676 Open files are represented by decimal numbers starting with zero. The largest possible value is  
1677 implementation-defined; however, all implementations shall support at least 0 to 9, inclusive, for  
1678 use by the application. These numbers are called *file descriptors*. The values 0, 1, and 2 have  
1679 special meaning and conventional uses and are implied by certain redirection operations; they  
1680 are referred to as *standard input*, *standard output*, and *standard error*, respectively. Programs  
1681 usually take their input from standard input, and write output on standard output. Error  
1682 messages are usually written on standard error. The redirection operators can be preceded by  
1683 one or more digits (with no intervening <blank> characters allowed) to designate the file  
1684 descriptor number.

1685 If the redirection operator is "<<" or "<<-", the word that follows the redirection operator shall  
1686 be subjected to quote removal; it is unspecified whether any of the other expansions occur. For  
1687 the other redirection operators, the word that follows the redirection operator shall be subjected  
1688 to tilde expansion, parameter expansion, command substitution, arithmetic expansion, and  
1689 quote removal. Path name expansion shall not be performed on the word by a non-interactive  
1690 shell; an interactive shell may perform it, but does do so only when the expansion would result  
1691 in one word.

1692 If more than one redirection operator is specified with a command, the order of evaluation is  
1693 from beginning to end.

1694 A failure to open or create a file shall cause a redirection to fail.

### 1695 2.7.1 Redirecting Input

1696 Input redirection shall cause the file whose name results from the expansion of *word* to be  
 1697 opened for reading on the designated file descriptor, or standard input if the file descriptor is not  
 1698 specified.

1699 The general format for redirecting input is:

```
1700 [n]<word
```

1701 where the optional *n* represents the file descriptor number. If the number is omitted, the  
 1702 redirection shall refer to standard input (file descriptor 0).

### 1703 2.7.2 Redirecting Output

1704 The two general formats for redirecting output are:

```
1705 [n]>word
1706 [n]>|word
```

1707 where the optional *n* represents the file descriptor number. If the number is omitted, the  
 1708 redirection shall refer to standard output (file descriptor 1).

1709 Output redirection using the '>' format shall fail if the *noclobber* option is set (see the  
 1710 description of *set -C*) and the file named by the expansion of *word* exists and is a regular file.  
 1711 Otherwise, redirection using the '>' or '>|' formats shall cause the file whose name results  
 1712 from the expansion of *word* to be created and opened for output on the designated file  
 1713 descriptor, or standard output if none is specified. If the file does not exist, it shall be created;  
 1714 otherwise, it shall be truncated to be an empty file after being opened.

### 1715 2.7.3 Appending Redirected Output

1716 Appended output redirection shall cause the file whose name results from the expansion of  
 1717 *word* to be opened for output on the designated file descriptor. The file is opened as if the *open()*  
 1718 function as defined in the System Interfaces volume of IEEE Std. 1003.1-200x was called with the  
 1719 *O\_APPEND* flag. If the file does not exist, it shall be created.

1720 The general format for appending redirected output is as follows:

```
1721 [n]>>word
```

1722 where the optional *n* represents the file descriptor number. If the number is omitted, the  
 1723 redirection refers to standard output (file descriptor 1).

### 1724 2.7.4 Here-Document

1725 The redirection operators "<<" and "<<-" both allow redirection of lines contained in a shell  
 1726 input file, known as a *here-document*, to the input of a command.

1727 The here-document shall be treated as a single word that begins after the next <newline>  
 1728 character and continues until there is a line containing only the delimiter, with no trailing  
 1729 <blank> characters. Then the next here-document starts, if there is one. The format is as follows:

```
1730 [n]<<word
1731 here-document
1732 delimiter
```

1733 where the optional *n* represents the file descriptor number. If the number is omitted, the here-  
 1734 document refers to standard output (file descriptor 0).



1735 If any character in *word* is quoted, the delimiter shall be formed by performing quote removal on  
 1736 *word*, and the here-document lines are not expanded. Otherwise, the delimiter shall be the *word*  
 1737 itself.

1738 If no characters in *word* are quoted, all lines of the here-document shall be expanded for  
 1739 parameter expansion, command substitution, and arithmetic expansion. In this case, the  
 1740 backslash in the input behaves as the backslash inside double-quotes (see Section 2.2.3 (on page  
 1741 2236)). However, the double-quote character ( ' " ' ) shall not be treated specially within a here-  
 1742 document, except when the double-quote appears within "\$ ( ) ", "` ` ", or "\$ { } ".

1743 If the redirection symbol is "<<-", all leading tab characters shall be stripped from input lines  
 1744 and the line containing the trailing delimiter. If more than one "<<" or "<<-" operator is  
 1745 specified on a line, the here-document associated with the first operator shall be supplied first by  
 1746 the application and shall be read first by the shell.

### 1747 **Examples**

1748 An example of a here-document follows:

```
1749 cat <<eof1; cat <<eof2
1750 Hi ,
1751 eof1
1752 Helene.
1753 eof2
```

### 1754 **2.7.5 Duplicating an Input File Descriptor**

1755 The redirection operator:

```
1756 [n]<&word
```

1757 is used to duplicate one input file descriptor from another, or to close one. If *word* evaluates to  
 1758 one or more digits, the file descriptor denoted by *n*, or standard input if *n* is not specified, shall  
 1759 be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent  
 1760 a file descriptor already open for input, a redirection error shall result; see Section 2.8.1 (on page  
 1761 2255). If *word* evaluates to '- ', file descriptor *n*, or standard input if *n* is not specified, shall be  
 1762 closed. If *word* evaluates to something else, the behavior is unspecified.

### 1763 **2.7.6 Duplicating an Output File Descriptor**

1764 The redirection operator:

```
1765 [n]>&word
```

1766 is used to duplicate one output file descriptor from another, or to close one. If *word* evaluates to  
 1767 one or more digits, the file descriptor denoted by *n*, or standard output if *n* is not specified, shall  
 1768 be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent  
 1769 a file descriptor already open for output, a redirection error shall result; see Section 2.8.1 (on  
 1770 page 2255). If *word* evaluates to '- ', file descriptor *n*, or standard output if *n* is not specified, is  
 1771 closed. If *word* evaluates to something else, the behavior is unspecified.

**1772 2.7.7 Open File Descriptors for Reading and Writing**

1773 The redirection operator:

1774 `[n]<>word`

1775 shall cause the file whose name is the expansion of *word* to be opened for both reading and  
1776 writing on the file descriptor denoted by *n*, or standard input if *n* is not specified. If the file does  
1777 not exist, it shall be created.

1778 **2.8 Exit Status and Errors**1779 **2.8.1 Consequences of Shell Errors**

1780 For a non-interactive shell, an error condition encountered by a special built-in (see Section 2.15  
1781 (on page 2276)) or other type of utility shall cause the shell to write a diagnostic message to  
1782 standard error and exit as shown in the following table:

|      | <b>Error</b>                                   | <b>Special Built-In</b> | <b>Other Utilities</b> |
|------|------------------------------------------------|-------------------------|------------------------|
| 1783 |                                                |                         |                        |
| 1784 | Shell language syntax error                    | Shall exit              | Shall exit             |
| 1785 | Utility syntax error (option or operand error) | Shall exit              | Shall not exit         |
| 1786 | Redirection error                              | Shall exit              | Shall not exit         |
| 1787 | Variable assignment error                      | Shall exit              | Shall not exit         |
| 1788 | Expansion error                                | Shall exit              | Shall exit             |
| 1789 | Command not found                              | N/A                     | May exit               |
| 1790 | Dot script not found                           | Shall exit              | N/A                    |

1791 An expansion error is one that occurs when the shell expansions defined in Section 2.6 (on page  
1792 2244) are carried out (for example, "\$ {x!y}", because '!' is not a valid operator); an  
1793 implementation may treat these as syntax errors if it is able to detect them during tokenization,  
1794 rather than during expansion.

1795 If any of the errors shown as “shall exit” or “(may) exit” occur in a subshell, the subshell shall  
1796 (or may exit) with a non-zero status, but the script containing the subshell shall not exit because  
1797 of the error.

1798 In all of the cases shown in the table, an interactive shell shall write a diagnostic message to  
1799 standard error without exiting.

1800 **2.8.2 Exit Status for Commands**

1801 Each command has an exit status that can influence the behavior of other shell commands. The  
1802 exit status of commands that are not utilities is documented in this section. The exit status of the  
1803 standard utilities is documented in their respective sections.

1804 If a command is not found, the exit status shall be 127. If the command name is found, but it is  
1805 not an executable utility, the exit status shall be 126. Applications that invoke utilities without  
1806 using the shell should use these exit status values to report similar errors.

1807 If a command fails during word expansion or redirection, its exit status shall be greater than  
1808 zero.

1809 Internally, for purposes of deciding whether a command exits with a non-zero exit status, the  
1810 shell shall recognize the entire status value retrieved for the command by the equivalent of the  
1811 *wait()* function WEXITSTATUS macro (as defined in the System Interfaces volume of  
1812 IEEE Std. 1003.1-200x). When reporting the exit status with the special parameter '?', the shell  
1813 shall report the full eight bits of exit status available. The exit status of a command that  
1814 terminated because it received a signal shall be reported as greater than 128.

## 1815 2.9 Shell Commands

1816 This section describes the basic structure of shell commands. The following command  
1817 descriptions each describe a format of the command that is only used to aid the reader in  
1818 recognizing the command type, and does not formally represent the syntax. Each description  
1819 discusses the semantics of the command; for a formal definition of the command language,  
1820 consult Section 2.11 (on page 2266).

1821 A *command* is one of the following:

- 1822 • *Simple command* (see Section 2.9.1)
- 1823 • *Pipeline* (see Section 2.9.2 (on page 2258))
- 1824 • *List or compound-list* (see Section 2.9.3 (on page 2259))
- 1825 • *Compound command* (see Section 2.9.4 (on page 2261))
- 1826 • *Function definition* (see Section 2.9.5 (on page 2263))

1827 Unless otherwise stated, the exit status of a command is that of the last simple command  
1828 executed by the command. There is no limit on the size of any shell command other than that  
1829 imposed by the underlying system (memory constraints, {ARG\_MAX}, and so on).

### 1830 2.9.1 Simple Commands

1831 A *simple command* is a sequence of optional variable assignments and redirections, in any  
1832 sequence, optionally followed by words and redirections, terminated by a control operator.

1833 When a given simple command is required to be executed (that is, when any conditional  
1834 construct such as an AND-OR list or a **case** statement has not bypassed the simple command),  
1835 the following expansions, assignments, and redirections are all performed from the beginning of  
1836 the command text to the end:

- 1837 1. The words that are recognized as variable assignments or redirections according to Section  
1838 2.11.2 (on page 2266) are saved for processing in steps 3 and 4.
- 1839 2. The words that are not variable assignments or redirections shall be expanded. If any fields  
1840 remain following their expansion, the first field shall be considered the command name  
1841 and remaining fields are the arguments for the command.
- 1842 3. Redirections shall be performed as described in Section 2.7 (on page 2251).
- 1843 4. Each variable assignment shall be expanded for tilde expansion, parameter expansion,  
1844 command substitution, arithmetic expansion, and quote removal prior to assigning the  
1845 value.

1846 In the preceding list, the order of steps 3 and 4 may be reversed for the processing of special  
1847 built-in utilities; see Section 2.15 (on page 2276).

1848 If no command name results, variable assignments shall affect the current execution  
1849 environment. Otherwise, the variable assignments shall be exported for the execution  
1850 environment of the command and shall not affect the current execution environment (except for  
1851 special built-ins). If any of the variable assignments attempt to assign a value to a read-only  
1852 variable, a variable assignment error occurs. See Section 2.8.1 (on page 2255) for the  
1853 consequences of these errors.

1854 If there is no command name, any redirections shall be performed in a subshell environment; it  
1855 is unspecified whether this subshell environment is the same one as that used for a command  
1856 substitution within the command. (To affect the current execution environment, see the *exec* (on  
1857 page 2287) special built-in.) If any of the redirections performed in the current shell execution

1858 environment fail, the command shall immediately fail with an exit status greater than zero, and  
 1859 the shell shall write an error message indicating the failure. See Section 2.8.1 (on page 2255) for  
 1860 the consequences of these failures on interactive and non-interactive shells.

1861 If there is a command name, execution shall continue as described in Section 2.9.1.1. If there is  
 1862 no command name, but the command contained a command substitution, the command shall  
 1863 complete with the exit status of the last command substitution performed. Otherwise, the  
 1864 command shall complete with a zero exit status.

#### 1865 2.9.1.1 Command Search and Execution

1866 If a simple command results in a command name and an optional list of arguments, the  
 1867 following actions shall be performed:

1868 1. If the command name does not contain any slashes, the first successful step in the  
 1869 following sequence shall occur:

1870 a. If the command name matches the name of a special built-in utility, that special  
 1871 built-in utility shall be invoked.

1872 b. If the command name matches the name of a function known to this shell, the  
 1873 function shall be invoked as described in Section 2.9.5 (on page 2263). If the  
 1874 implementation has provided a standard utility in the form of a function, it shall not  
 1875 be recognized at this point. It shall be invoked in conjunction with the path search in  
 1876 step 1d.

1877 c. If the command name matches the name of a utility listed in the following table, that  
 1878 utility shall be invoked.

|      |                |                |               |                |
|------|----------------|----------------|---------------|----------------|
| 1879 | <i>alias</i>   | <i>false</i>   | <i>jobs</i>   | <i>true</i>    |
| 1880 | <i>bg</i>      | <i>fc</i>      | <i>kill</i>   | <i>umask</i>   |
| 1881 | <i>cd</i>      | <i>fg</i>      | <i>newgrp</i> | <i>unalias</i> |
| 1882 | <i>command</i> | <i>getopts</i> | <i>read</i>   | <i>wait</i>    |

1883 d. Otherwise, the command is searched for using the *PATH* environment variable as  
 1884 described in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 8,  
 1885 Environment Variables:

1886 i. If the search is successful:

1887 a. If the system has implemented the utility as a regular built-in or as a shell  
 1888 function, it shall be invoked at this point in the path search.

1889 b. Otherwise, the shell executes the utility in a separate utility environment  
 1890 (see Section 2.13 (on page 2273)) with actions equivalent to calling the  
 1891 *execve()* function as defined in the System Interfaces volume of  
 1892 IEEE Std. 1003.1-200x with the *path* argument set to the path name  
 1893 resulting from the search, *arg0* set to the command name, and the  
 1894 remaining arguments set to the operands, if any.

1895 If the *execve()* function fails due to an error equivalent to the [ENOEXEC]  
 1896 error defined in the System Interfaces volume of IEEE Std. 1003.1-200x,  
 1897 the shell shall execute a command equivalent to having a shell invoked  
 1898 with the command name as its first operand, along with any remaining  
 1899 arguments passed along. If the executable file is not a text file, the shell  
 1900 may bypass this command execution, write an error message, and return  
 1901 an exit status of 126.

1902                   Once a utility has been searched for and found (either as a result of this specific  
 1903 search or as part of an unspecified shell start-up activity), an implementation  
 1904 may remember its location and need not search for the utility again unless the  
 1905 *PATH* variable has been the subject of an assignment. If the remembered  
 1906 location fails for a subsequent invocation, the shell shall repeat the search to  
 1907 find the new location for the utility, if any.

1908                   ii. If the search is unsuccessful, the command shall fail with an exit status of 127  
 1909 and the shell shall write an error message.

1910           2. If the command name contains at least one slash, the shell shall execute the utility in a  
 1911 separate utility environment with actions equivalent to calling the *execve()* function  
 1912 defined in the System Interfaces volume of IEEE Std. 1003.1-200x with the *path* and *arg0*  
 1913 arguments set to the command name, and the remaining arguments set to the operands, if  
 1914 any.

1915           If the *execve()* function fails due to an error equivalent to the [ENOEXEC] error, the shell  
 1916 shall execute a command equivalent to having a shell invoked with the command name as  
 1917 its first operand, along with any remaining arguments passed along. If the executable file is  
 1918 not a text file, the shell may bypass this command execution, write an error message, and  
 1919 return an exit status of 126.

## 1920 2.9.2 Pipelines

1921           A *pipeline* is a sequence of one or more commands separated by the control operator '|'. The  
 1922 standard output of all but the last command shall be connected to the standard input of the next  
 1923 command.

1924           The format for a pipeline is:

```
1925 [!] command1 [| command2 ...]
```

1926           The standard output of *command1* shall be connected to the standard input of *command2*. The  
 1927 standard input, standard output, or both of a command shall be considered to be assigned by the  
 1928 pipeline before any redirection specified by redirection operators that are part of the command  
 1929 (see Section 2.7 (on page 2251)).

1930           If the pipeline is not in the background (see Section 2.9.3.1 (on page 2259)), the shell shall wait for  
 1931 the last command specified in the pipeline to complete, and may also wait for all commands to  
 1932 complete.

## 1933 Exit Status

1934           If the reserved word ! does not precede the pipeline, the exit status shall be the exit status of the  
 1935 last command specified in the pipeline. Otherwise, the exit status shall be the logical NOT of the  
 1936 exit status of the last command. That is, if the last command returns zero, the exit status shall be  
 1937 1; if the last command returns greater than zero, the exit status shall be zero.

1938 **2.9.3 Lists**

1939 An *AND-OR list* is a sequence of one or more pipelines separated by the operators "&&" and  
 1940 "||".

1941 A *list* is a sequence of one or more AND-OR lists separated by the operators ';' and '&' and  
 1942 optionally terminated by ';', '&', or <newline>.

1943 The operators "&&" and "||" shall have equal precedence and are evaluated from beginning to  
 1944 end. For example, both of the following commands write solely **bar** to standard output:

```
1945 false && echo foo || echo bar
1946 true || echo foo && echo bar
```

1947 A ';' or <newline> character terminator shall cause the preceding AND-OR list to be executed  
 1948 sequentially; an '&' shall cause asynchronous execution of the preceding AND-OR list.

1949 The term *compound-list* is derived from the grammar in Section 2.11 (on page 2266); it is  
 1950 equivalent to a sequence of *lists*, separated by <newline> characters, that can be preceded or  
 1951 followed by an arbitrary number of <newline> characters.

1952 **Examples**

1953 The following is an example that illustrates <newline> characters in compound-lists:

```
1954 while
1955 # a couple of <newline>s
1956 # a list
1957 date && who || ls; cat file
1958 # a couple of <newline>s
1959 # another list
1960 wc file > output & true
1961 do
1962 # 2 lists
1963 ls
1964 cat file
1965 done
```

1966 **2.9.3.1 Asynchronous Lists**

1967 If a command is terminated by the control operator ampersand ('&'), the shell shall execute the  
 1968 command asynchronously in a subshell. This means that the shell shall not wait for the  
 1969 command to finish before executing the next command.

1970 The format for running a command in the background is:

```
1971 command1 & [command2 & ...]
```

1972 The standard input for an asynchronous list, before any explicit redirections are performed, shall  
 1973 be considered to be assigned to a file that has the same properties as **/dev/null**. If it is an  
 1974 interactive shell, this need not happen. In all cases, explicit redirection of standard input shall  
 1975 override this activity.

1976 When an element of an asynchronous list (the portion of the list ended by an ampersand, such as  
 1977 *command1*, above) is started by the shell, the process ID of the last command in the asynchronous  
 1978 list element shall become known in the current shell execution environment; see Section 2.13 (on  
 1979 page 2273). This process ID shall remain known until:

- 1980 1. The command terminates and the application waits for the process ID.  
1981 2. Another asynchronous list invoked before "\$!" (corresponding to the previous  
1982 asynchronous list) is expanded in the current execution environment.

1983 The implementation need not retain more than the {CHILD\_MAX} most recent entries in its list  
1984 of known process IDs in the current shell execution environment.

1985 **Exit Status**

1986 The exit status of an asynchronous list shall be zero.

1987 **2.9.3.2 Sequential Lists**

1988 Commands that are separated by a semicolon ( ; ) shall be executed sequentially.

1989 The format for executing commands sequentially shall be:

1990 `command1 [ ; command2 ] . . .`

1991 Each command shall be expanded and executed in the order specified.

1992 **Exit Status**

1993 The exit status of a sequential list shall be the exit status of the last command in the list.

1994 **2.9.3.3 AND Lists**

1995 The control operator "&&" denotes an AND list. The format shall be:

1996 `command1 [ && command2 ] . . .`

1997 First *command1* shall be executed. If its exit status is zero, *command2* shall be executed, and so on,  
1998 until a command has a non-zero exit status or there are no more commands left to execute. The  
1999 commands are expanded only if they are executed.

2000 **Exit Status**

2001 The exit status of an AND list shall be the exit status of the last command that is executed in the  
2002 list.

2003 **2.9.3.4 OR Lists**

2004 The control operator " || " denotes an OR List. The format shall be:

2005 `command1 [ || command2 ] . . .`

2006 First, *command1* shall be executed. If its exit status is non-zero, *command2* shall be executed, and  
2007 so on, until a command has a zero exit status or there are no more commands left to execute.

2008 **Exit Status**

2009 The exit status of an OR list shall be the exit status of the last command that is executed in the  
2010 list.



2011 **2.9.4 Compound Commands**

2012 The shell has several programming constructs that are *compound commands*, which provide  
 2013 control flow for commands. Each of these compound commands has a reserved word or control  
 2014 operator at the beginning, and a corresponding terminator reserved word or operator at the end.  
 2015 In addition, each can be followed by redirections on the same line as the terminator. Each  
 2016 redirection shall apply to all the commands within the compound command that do not  
 2017 explicitly override that redirection.

2018 **2.9.4.1 Grouping Commands**

2019 The format for grouping commands is as follows:

2020 (*compound-list*) Execute *compound-list* in a subshell environment; see Section 2.13 (on page  
 2021 2273). Variable assignments and built-in commands that affect the  
 2022 environment shall not remain in effect after the list finishes.

2023 { *compound-list*; } Execute *compound-list* in the current process environment. The semicolon  
 2024 shown here is an example of a control operator delimiting the } reserved  
 2025 word. Other delimiters are possible, as shown in Section 2.11 (on page  
 2026 2266); a <newline> character is frequently used.

2027 **Exit Status**

2028 The exit status of a grouping command shall be the exit status of *list*.

2029 **2.9.4.2 For Loop**

2030 The **for** loop executes a sequence of commands for each member in a list of *items*. The **for** loop  
 2031 requires that the reserved words **do** and **done** be used to delimit the sequence of commands.

2032 The format for the **for** loop is as follows:

```
2033 for name [in [word ...]]
2034 do
2035 compound-list
2036 done
```

2037 First, the list of words following **in** shall be expanded to generate a list of items. Then, the  
 2038 variable *name* shall be set to each item, in turn, and the *compound-list* executed each time. If no  
 2039 items result from the expansion, the *compound-list* shall not be executed. Omitting:

```
2040 in word...
```

2041 is equivalent to:

```
2042 in "$@"
```

2043 **Exit Status**

2044 The exit status of a **for** command shall be the exit status of the last command that executes. If  
 2045 there are no items, the exit status shall be zero.

2046 2.9.4.3 *Case Conditional Construct*

2047 The conditional construct **case** shall execute the *compound-list* corresponding to the first one of  
 2048 several *patterns* (see Section 2.14 (on page 2274)) that is matched by the string resulting from the  
 2049 tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote  
 2050 removal of the given word. The reserved word **in** shall denote the beginning of the patterns to be  
 2051 matched. Multiple patterns with the same *compound-list* shall be delimited by the '|' symbol.  
 2052 The control operator ')' terminates a list of patterns corresponding to a given action. The  
 2053 *compound-list* for each list of patterns, with the possible exception of the last, shall be terminated  
 2054 with ";". The **case** construct terminates with the reserved word **esac** (**case** reversed).

2055 The format for the **case** construct is as follows:

```
2056 case word in
2057 [(]pattern1) compound-list;;
2058 [(]pattern[| pattern] ...) compound-list;;] ...
2059 [(]pattern[| pattern] ...) compound-list]
2060 esac
```

2061 The ";" is optional for the last *compound-list*.

2062 In order from the beginning to the end of the **case** statement, each *pattern* that labels a  
 2063 *compound-list* shall be subjected to tilde expansion, parameter expansion, command substitution,  
 2064 and arithmetic expansion, and the result of these expansions shall be compared against the  
 2065 expansion of *word*, according to the rules described in Section 2.14 (on page 2274) (which also  
 2066 describes the effect of quoting parts of the pattern). After the first match, no more patterns shall  
 2067 be expanded, and the *compound-list* shall be executed. The order of expansion and comparison of  
 2068 multiple *patterns* that label a *compound-list* statement is unspecified.

2069 **Exit Status**

2070 The exit status of **case** shall be zero if no patterns are matched. Otherwise, the exit status shall be  
 2071 the exit status of the last command executed in the *compound-list*.

2072 2.9.4.4 *If Conditional Construct*

2073 The **if** command shall execute a *compound-list* and use its exit status to determine whether to  
 2074 execute another *compound-list*.

2075 The format for the **if** construct is as follows:

```
2076 if compound-list
2077 then
2078 compound-list
2079 [elif compound-list
2080 then
2081 compound-list] ...
2082 [else
2083 compound-list]
```

2084 The **if** *compound-list* shall be executed; if its exit status is zero, the **then** *compound-list* shall be  
 2085 executed and the command shall complete. Otherwise, each **elif** *compound-list* shall be executed,  
 2086 in turn, and if its exit status is zero, the **then** *compound-list* shall be executed and the command  
 2087 shall complete. Otherwise, the **else** *compound-list* shall be executed.

2088        **Exit Status**

2089        The exit status of the **if** command shall be the exit status of the **then** or **else** *compound-list* that  
 2090        was executed, or zero, if none was executed.

2091    2.9.4.5    *While Loop*

2092        The **while** loop shall continuously execute one *compound-list* as long as another *compound-list* has  
 2093        a zero exit status.

2094        The format of the **while** loop is as follows:

```
2095 while compound-list-1
2096 do
2097 compound-list-2
2098 done
```

2099        The *compound-list-1* shall be executed, and if it has a non-zero exit status, the **while** command  
 2100        shall complete. Otherwise, the *compound-list-2* shall be executed, and the process shall repeat.

2101        **Exit Status**

2102        The exit status of the **while** loop shall be the exit status of the last *compound-list-2* executed, or  
 2103        zero if none was executed.

2104    2.9.4.6    *Until Loop*

2105        The **until** loop shall continuously execute one *compound-list* as long as another *compound-list* has  
 2106        a non-zero exit status.

2107        The format of the **until** loop is as follows:

```
2108 until compound-list-1
2109 do
2110 compound-list-2
2111 done
```

2112        The *compound-list-1* shall be executed, and if it has a zero exit status, the **until** command  
 2113        completes. Otherwise, the *compound-list-2* shall be executed, and the process repeats.

2114        **Exit Status**

2115        The exit status of the **until** loop shall be the exit status of the last *compound-list-2* executed, or  
 2116        zero if none was executed.

2117    2.9.5        **Function Definition Command**

2118        A function is a user-defined name that is used as a simple command to call a compound  
 2119        command with new positional parameters. A function is defined with a *function definition*  
 2120        *command*.

2121        The format of a function definition command is as follows:

```
2122 fname() compound-command[io-redirect ...]
```

2123        The function is named *fname*; the application shall ensure that it is a name (see the Base  
 2124        Definitions volume of IEEE Std. 1003.1-200x, Section 3.232, Name). An implementation may  
 2125        allow other characters in a function name as an extension. The implementation shall maintain  
 2126        separate name spaces for functions and variables.

2127 The argument *compound-command* represents a compound command, as described in Section  
2128 2.9.4 (on page 2261).

2129 When the function is declared, none of the expansions in Section 2.6 (on page 2244) shall be  
2130 performed on the text in *compound-command* or *io-redirect*; all expansions shall be performed as  
2131 normal each time the function is called. Similarly, the optional *io-redirect* redirections and any  
2132 variable assignments within *compound-command* shall be performed during the execution of the  
2133 function itself, not the function definition. See Section 2.8.1 (on page 2255) for the consequences  
2134 of failures of these operations on interactive and non-interactive shells.

2135 When a function is executed, it shall have the syntax-error and variable-assignment properties  
2136 described for special built-in utilities in the enumerated list at the beginning of Section 2.15 (on  
2137 page 2276).

2138 The *compound-command* shall be executed whenever the function name is specified as the name  
2139 of a simple command (see Section 2.9.1.1 (on page 2257)). The operands to the command  
2140 temporarily shall become the positional parameters during the execution of the *compound-*  
2141 *command*; the special parameter '# ' also shall be changed to reflect the number of operands. The  
2142 special parameter 0 shall be unchanged. When the function completes, the values of the  
2143 positional parameters and the special parameter '# ' shall be restored to the values they had  
2144 before the function was executed. If the special built-in *return* is executed in the *compound-*  
2145 *command*, the function completes and execution shall resume with the next command after the  
2146 function call.

#### 2147 **Exit Status**

2148 The exit status of a function definition shall be zero if the function was declared successfully;  
2149 otherwise, it shall be greater than zero. The exit status of a function invocation shall be the exit  
2150 status of the last command executed by the function.

**2.10 Executable Script**

2152 XSI XSI-Conformant systems shall support executable scripts. A successful call to a function of the  
2153 *exec* family with an executable script as the first parameter shall result in a new process, where  
2154 the process image that is started is that of the interpreter. The path name of the interpreter  
2155 follows the "#!" characters.

2156 If the executable script has a first line:

```
2157 #! interpreter [arg]
```

2158 then the interpreter shall be called with an argument array consisting of an unspecified zero'th  
2159 argument, followed by *arg* (if present), followed by a path name for the script, followed by the  
2160 arguments following the zero'th argument in the *exec* call of the script.

2161 No shell operations (as described in Section 2.1 (on page 2235)) shall be performed on the first  
2162 line of an executable script.

2163 The behavior shall be unspecified if the first line of the executable script does not meet all of the  
2164 following criteria:

2165 1. The first line shall be in one of the formats below:

```
2166 "#!%s\n" interpreter
```

```
2167 "#!<delta>%s\n" interpreter
```

```
2168 "#!%s<delta>%s\n" interpreter arg
```

```
2169 "#!<delta>%s<delta>%s\n" interpreter arg
```

2170 2. The *interpreter* argument shall be an absolute path name of an executable file other than an  
2171 executable script.

2172 3. The *interpreter* argument and the *arg* argument, if present, shall not contain any quoting  
2173 characters.

2174 4. The *interpreter* argument and the *arg* argument, if present, shall not contain any white-  
2175 space characters.

2176 5. The length of the first line shall be no longer than 80 bytes.  
2177

## 2178 2.11 Shell Grammar

2179 The following grammar defines the Shell Command Language. This formal syntax shall take  
2180 precedence over the preceding text syntax description.

### 2181 2.11.1 Shell Grammar Lexical Conventions

2182 The input language to the shell must be first recognized at the character level. The resulting  
2183 tokens shall be classified by their immediate context according to the following rules (applied in  
2184 order). These rules are used to determine what a “token” is that is subject to parsing at the  
2185 token level. The rules for token recognition in Section 2.3 (on page 2238) shall apply.

- 2186 1. A <newline> character shall be returned as the token identifier **NEWLINE**.
- 2187 2. If the token is an operator, the token identifier for that operator shall result.
- 2188 3. If the string consists solely of digits and the delimiter character is one of '<' or '>', the  
2189 token identifier **IO\_NUMBER** shall be returned.
- 2190 4. Otherwise, the token identifier **TOKEN** results.

2191 Further distinction on **TOKEN** is context-dependent. It may be that the same **TOKEN** yields  
2192 **WORD**, a **NAME**, an **ASSIGNMENT**, or one of the reserved words below, dependent upon the  
2193 context. Some of the productions in the grammar below are annotated with a rule number from  
2194 the following list. When a **TOKEN** is seen where one of those annotated productions could be  
2195 used to reduce the symbol, the applicable rule shall be applied to convert the token identifier  
2196 type of the **TOKEN** to a token identifier acceptable at that point in the grammar. The reduction  
2197 shall then proceed based upon the token identifier type yielded by the rule applied. When more  
2198 than one rule applies, the highest numbered rule shall apply (which in turn may refer to another  
2199 rule). (Note that except in rule 7, the presence of an '=' in the token has no effect.)

2200 The **WORD** tokens shall have the word expansion rules applied to them immediately before the  
2201 associated command is executed, not at the time the command is parsed.

### 2202 2.11.2 Shell Grammar Rules

- 2203 1. [Command Name]

2204 When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word  
2205 shall result. Otherwise, the token **WORD** shall be returned. Also, if the parser is in any  
2206 state where only a reserved word could be the next correct token, proceed as above. This  
2207 rule applies rather narrowly: when a compound list is terminated by some clear delimiter  
2208 (such as the closing **fi** of an inner **if\_clause**) then it would apply; where the compound list  
2209 might continue (as in after a ';'), rule 7a (and consequently the first sentence of this rule)  
2210 would apply. In many instances the two conditions are identical, but this part of this rule  
2211 does not give license to treating a **WORD** as a reserved word unless it is in a place where a  
2212 reserved word shall appear.

2213 **Note:** Because at this point quote marks are retained in the token, quoted strings  
2214 cannot be recognized as reserved words. This rule also implies that reserved  
2215 words are not recognized except in certain positions in the input, such as after a  
2216 <newline> character or semicolon; the grammar presumes that if the reserved  
2217 word is intended, it is properly delimited by the user, and does not attempt to  
2218 reflect that requirement directly. Also note that line joining is done before  
2219 tokenization, as described in Section 2.2.1 (on page 2236), so escaped  
2220 <newline>s are already removed at this point.

- 2221 Rule 1 is not directly referenced in the grammar, but is referred to by other rules, or applies  
2222 globally.
- 2223 2. [Redirection to or from file name]
- 2224 The expansions specified in Section 2.7 (on page 2251) shall occur. As specified there,  
2225 exactly one field can result (or the result is unspecified), and there are additional  
2226 requirements on path name expansion.
- 2227 3. [Redirection from here-document]
- 2228 Quote removal shall be applied to the word to determine the delimiter that is used to find  
2229 the end of the here-document that begins after the next <newline> character.
- 2230 4. [Case statement termination]
- 2231 When the **TOKEN** is exactly the reserved word **esac**, the token identifier for **esac** shall  
2232 result. Otherwise, the token **WORD** shall be returned.
- 2233 5. [**NAME** in **for**]
- 2234 When the **TOKEN** meets the requirements for a name (see the Base Definitions volume of  
2235 IEEE Std. 1003.1-200x, Section 3.232, Name), the token identifier **NAME** shall result. |  
2236 Otherwise, the token **WORD** shall be returned.
- 2237 6. [Third word of **for** and **case**]
- 2238 When the **TOKEN** is exactly the reserved word **in**, the token identifier for **in** shall result.  
2239 Otherwise, the token **WORD** shall be returned. (As indicated in the grammar, a *linebreak*  
2240 precedes the token **in**. If <newline> characters are present at the indicated location, it is  
2241 the token after them that is treated in this fashion.)
- 2242 7. [Assignment preceding command name]
- 2243 a. [When the first word]
- 2244 If the **TOKEN** does not contain the character '=' , rule 1 is applied. Otherwise, 7b  
2245 shall be applied.
- 2246 b. [Not the first word]
- 2247 If the **TOKEN** contains the equal sign character:
- 2248 — If it begins with '=' , the token **WORD** shall be returned.
- 2249 — If all the characters preceding '=' form a valid name (see the Base Definitions |  
2250 volume of IEEE Std. 1003.1-200x, Section 3.232, Name), the token |  
2251 **ASSIGNMENT\_WORD** shall be returned. (Quoted characters cannot participate  
2252 in forming a valid name.)
- 2253 — Otherwise, it is unspecified whether it is **ASSIGNMENT\_WORD** or **WORD** that  
2254 is returned.
- 2255 Assignment to the **NAME** shall occur as specified in Section 2.9.1 (on page 2256).
- 2256 8. [**NAME** in function]
- 2257 When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word  
2258 shall result. Otherwise, when the **TOKEN** meets the requirements for a name, the token  
2259 identifier **NAME** shall result. Otherwise, rule 7 applies.
- 2260 9. [Body of function]

```

2261 Word expansion and assignment shall never occur, even when required by the rules above,
2262 when this rule is being parsed. Each TOKEN that might either be expanded or have
2263 assignment applied to it shall instead be returned as a single WORD consisting only of
2264 characters that are exactly the token described in Section 2.3 (on page 2238).

2265 /* -----
2266 The grammar symbols
2267 ----- */

2268 %token WORD
2269 %token ASSIGNMENT_WORD
2270 %token NAME
2271 %token NEWLINE
2272 %token IO_NUMBER

2273 /* The following are the operators mentioned above. */

2274 %token AND_IF OR_IF DSEMI
2275 /* '&&' '|' ';' */

2276 %token DLESS DGREAT LESSAND GREATAND LESSGREAT DLESSDASH
2277 /* '<<' '>>' '<&' '>&' '<>' '<<-' */

2278 %token CLOBBER
2279 /* '>|' */

2280 /* The following are the reserved words. */

2281 %token If Then Else Elif Fi Do Done
2282 /* 'if' 'then' 'else' 'elif' 'fi' 'do' 'done' */

2283 %token Case Esac While Until For
2284 /* 'case' 'esac' 'while' 'until' 'for' */

2285 /* These are reserved words, not operator tokens, and are
2286 recognized when reserved words are recognized. */

2287 %token Lbrace Rbrace Bang
2288 /* '{' '}' '!' */

2289 %token In
2290 /* 'in' */

2291 /* -----
2292 The Grammar
2293 ----- */

2294 %start complete_command
2295 %%
2296 complete_command : list separator
2297 | list
2298 ;
2299 list : list separator_op and_or
2300 | and_or
2301 ;
2302 and_or : pipeline
2303 | and_or AND_IF linebreak pipeline
2304 | and_or OR_IF linebreak pipeline
2305 ;

```



```

2306 pipeline : pipe_sequence
2307 | Bang pipe_sequence
2308 ;
2309 pipe_sequence : command
2310 | pipe_sequence '|' linebreak command
2311 ;
2312 command : simple_command
2313 | compound_command
2314 | compound_command redirect_list
2315 | function_definition
2316 ;
2317 compound_command : brace_group
2318 | subshell
2319 | for_clause
2320 | case_clause
2321 | if_clause
2322 | while_clause
2323 | until_clause
2324 ;
2325 subshell : '(' compound_list ')'
2326 ;
2327 compound_list : term
2328 | newline_list term
2329 | term separator
2330 | newline_list term separator
2331 ;
2332 term : term separator and_or
2333 | and_or
2334 ;
2335 for_clause : For name linebreak do_group
2336 | For name linebreak in sequential_sep_do_group
2337 | For name linebreak in wordlist sequential_sep do_group
2338 ;
2339 name : NAME /* Apply rule 5 */
2340 ;
2341 in : In /* Apply rule 6 */
2342 ;
2343 wordlist : wordlist WORD
2344 | WORD
2345 ;
2346 case_clause : Case WORD linebreak in linebreak case_list Esac
2347 | Case WORD linebreak in linebreak case_list_ns Esac
2348 | Case WORD linebreak in linebreak Esac
2349 ;
2350 case_list_ns : case_list case_item_ns
2351 | case_item_ns
2352 ;
2353 case_list : case_list case_item
2354 | case_item
2355 ;
2356 case_item_ns : pattern ')' linebreak linebreak
2357 | pattern ')' compound_list linebreak

```

```

2358 | '(' pattern ')' linebreak linebreak
2359 | '(' pattern ')' compound_list linebreak
2360 ;
2361 case_item : pattern ')' linebreak DSEMI linebreak
2362 | pattern ')' compound_list linebreak
2363 | '(' pattern ')' linebreak linebreak
2364 | '(' pattern ')' compound_list linebreak
2365 ;
2366 pattern : WORD /* Apply rule 4 */
2367 | pattern '|' WORD /* Do not apply rule (4) */
2368 ;
2369 if_clause : If compound_list Then compound_list else_part Fi
2370 | If compound_list Then compound_list Fi
2371 ;
2372 else_part : Elif compound_list Then else_part
2373 | Else compound_list
2374 ;
2375 while_clause : While compound_list do_group
2376 ;
2377 until_clause : Until compound_list do_group
2378 ;
2379 function_definition : fname '(' ')' linebreak function_body
2380 ;
2381 function_body : compound_command /* Apply rule 9 */
2382 | compound_command redirect_list /* Apply rule 9 */
2383 ;
2384 fname : NAME /* Apply rule 8 */
2385 ;
2386 brace_group : Lbrace compound_list Rbrace
2387 ;
2388 do_group : Do compound_list Done
2389 ;
2390 simple_command : cmd_prefix cmd_word cmd_suffix
2391 | cmd_prefix cmd_word
2392 | cmd_prefix
2393 | cmd_name cmd_suffix
2394 | cmd_name
2395 ;
2396 cmd_name : WORD /* Apply rule 7a */
2397 ;
2398 cmd_word : WORD /* Apply rule 7b */
2399 ;
2400 cmd_prefix : io_redirect
2401 | cmd_prefix io_redirect
2402 | ASSIGNMENT_WORD
2403 | cmd_prefix ASSIGNMENT_WORD
2404 ;
2405 cmd_suffix : io_redirect
2406 | cmd_suffix io_redirect
2407 | WORD
2408 | cmd_suffix WORD
2409 ;

```

```

2410 redirect_list : io_redirect
2411 | redirect_list io_redirect
2412 ;
2413 io_redirect : io_file
2414 | IO_NUMBER io_file
2415 | io_here
2416 | IO_NUMBER io_here
2417 ;
2418 io_file : '<' filename
2419 | LESSAND filename
2420 | '>' filename
2421 | GREATAND filename
2422 | DGREAT filename
2423 | LESSGREAT filename
2424 | CLOBBER filename
2425 ;
2426 filename : WORD /* Apply rule 2 */
2427 ;
2428 io_here : DLESS here_end
2429 | DLESSDASH here_end
2430 ;
2431 here_end : WORD /* Apply rule 3 */
2432 ;
2433 newline_list : NEWLINE
2434 | newline_list NEWLINE
2435 ;
2436 linebreak : newline_list
2437 | /* empty */
2438 ;
2439 separator_op : '&'
2440 | ';'
2441 ;
2442 separator : separator_op linebreak
2443 | newline_list
2444 ;
2445 sequential_sep : ';' linebreak
2446 | newline_list
2447 ;

```

**2448 2.12 Signals and Error Handling**

2449 When a command is in an asynchronous list, the shell shall prevent SIGQUIT and SIGINT  
2450 signals from the keyboard from interrupting the command. Otherwise, signals shall have the  
2451 values inherited by the shell from its parent (see also the *trap* (on page 2307) special built-in).

2452 When a signal for which a trap has been set is received while the shell is waiting for the  
2453 completion of a utility executing a foreground command, the trap associated with that signal  
2454 shall not be executed until after the foreground command has completed. When the shell is  
2455 waiting, by means of the *wait* utility, for asynchronous commands to complete, the reception of a  
2456 signal for which a trap has been set shall cause the *wait* utility to return immediately with an exit  
2457 status >128, immediately after which the trap associated with that signal shall be taken.

2458 If multiple signals are pending for the shell for which there are associated trap actions, the order  
2459 of execution of trap actions is unspecified.

## 2460 2.13 Shell Execution Environment

2461 A shell execution environment consists of the following:

- 2462 • Open files inherited upon invocation of the shell, plus open files controlled by *exec*
- 2463 • Working directory as set by *cd*
- 2464 • File creation mask set by *umask*
- 2465 • Current traps set by *trap*
- 2466 • Shell parameters that are set by variable assignment (see the *set* (on page 2297) special built-
- 2467 in) or from the System Interfaces volume of IEEE Std. 1003.1-200x environment inherited by
- 2468 the shell when it begins (see the *export* (on page 2291) special built-in)
- 2469 • Shell functions; see Section 2.9.5 (on page 2263)
- 2470 • Options turned on at invocation or by *set*
- 2471 • Process IDs of the last commands in asynchronous lists known to this shell environment; see
- 2472 Section 2.9.3.1 (on page 2259)
- 2473 • Shell aliases; see Section 2.3.1 (on page 2239)

2474 Utilities other than the special built-ins (see Section 2.15 (on page 2276)) shall be invoked in a  
2475 separate environment that consists of the following. The initial value of these objects shall be the  
2476 same as that for the parent shell, except as noted below.

- 2477 • Open files inherited on invocation of the shell, open files controlled by the *exec* special built-
- 2478 in plus any modifications, and additions specified by any redirections to the utility
- 2479 • Current working directory
- 2480 • File creation mask
- 2481 • If the utility is a shell script, traps caught by the shell shall be set to the default values and
- 2482 traps ignored by the shell shall be set to be ignored by the utility; if the utility is not a shell
- 2483 script, the trap actions (default or ignore) shall be mapped into the appropriate signal
- 2484 handling actions for the utility
- 2485 • Variables with the *export* attribute, along with those explicitly exported for the duration of the
- 2486 command, shall be passed to the utility as System Interfaces volume of IEEE Std. 1003.1-200x
- 2487 environment variables

2488 The environment of the shell process shall not be changed by the utility unless explicitly  
2489 specified by the utility description (for example, *cd* and *umask*).

2490 A subshell environment shall be created as a duplicate of the shell environment, except that  
2491 signal traps set by that shell environment shall be set to the default values. Changes made to the  
2492 subshell environment shall not affect the shell environment. Command substitution, commands  
2493 that are grouped with parentheses, and asynchronous lists shall be executed in a subshell  
2494 environment. Additionally, each command of a multi-command pipeline is in a subshell  
2495 environment; as an extension, however, any or all commands in a pipeline may be executed in  
2496 the current environment. All other commands shall be executed in the current shell  
2497 environment.

## 2498 2.14 Pattern Matching Notation

2499 The pattern matching notation described in this section is used to specify patterns for matching  
2500 strings in the shell. Historically, pattern matching notation is related to, but slightly different  
2501 from, the regular expression notation described in the Base Definitions volume of  
2502 IEEE Std. 1003.1-200x, Chapter 9, Regular Expressions. For this reason, the description of the  
2503 rules for this pattern matching notation are based on the description of regular expression  
2504 notation, modified to include backslash escape processing.

### 2505 2.14.1 Patterns Matching a Single Character

2506 The following *patterns matching a single character* match a single character: *ordinary characters*,  
2507 *special pattern characters*, and *pattern bracket expressions*. The pattern bracket expression also shall  
2508 match a single collating element. A backslash character shall escape the following character. The  
2509 escaping backslash shall be discarded.

2510 An ordinary character is a pattern that shall match itself. It can be any character in the supported  
2511 character set except for NUL, those special shell characters in Section 2.2 (on page 2236) that  
2512 require quoting, and the following three special pattern characters. Matching shall be based on  
2513 the bit pattern used for encoding the character, not on the graphic representation of the  
2514 character. If any character (ordinary, shell special, or pattern special) is quoted, that pattern shall  
2515 match the character itself. The shell special characters always require quoting.

2516 When unquoted and outside a bracket expression, the following three characters shall have  
2517 special meaning in the specification of patterns:

- 2518 ? A question-mark is a pattern that shall match any character.
- 2519 \* An asterisk is a pattern that shall match multiple characters, as described in Section 2.14.2.
- 2520 [ The open bracket shall introduce a pattern bracket expression.

2521 The description of basic regular expression bracket expressions in the Base Definitions volume  
2522 of IEEE Std. 1003.1-200x, Section 9.3.5, RE Bracket Expression shall also apply to the pattern  
2523 bracket expression, except that the exclamation mark character ('!') shall replace the  
2524 circumflex character ('^') in its role in a *non-matching list* in the regular expression notation. A  
2525 bracket expression starting with an unquoted circumflex character produces unspecified results.

2526 When pattern matching is used where shell quote removal is not performed (such as in the  
2527 argument to the *find name* primary when *find* is being called using one of the *exec* functions as  
2528 defined in the System Interfaces volume of IEEE Std. 1003.1-200x, or in the *pattern* argument to  
2529 the *fnmatch()* function), special characters can be escaped to remove their special meaning by  
2530 preceding them with a backslash character. This escaping backslash is discarded. The sequence  
2531 "\\\" represents one literal backslash. All of the requirements and effects of quoting on ordinary,  
2532 shell special, and special pattern characters shall apply to escaping in this context.

### 2533 2.14.2 Patterns Matching Multiple Characters

2534 The following rules are used to construct *patterns matching multiple characters* from *patterns*  
2535 *matching a single character*:

- 2536 1. The asterisk ('\*') is a pattern that shall match any string, including the null string.
- 2537 2. The concatenation of *patterns matching a single character* is a valid pattern that shall match  
2538 the concatenation of the single characters or collating elements matched by each of the  
2539 concatenated patterns.

- 2540 3. The concatenation of one or more *patterns matching a single character* with one or more  
 2541 asterisks is a valid pattern. In such patterns, each asterisk shall match a string of zero or  
 2542 more characters, matching the greatest possible number of characters that still allows the  
 2543 remainder of the pattern to match the string.

### 2544 2.14.3 Patterns Used for File Name Expansion

2545 The rules described so far in Section 2.14.1 (on page 2274) and Section 2.14.2 (on page 2274) are  
 2546 qualified by the following rules that apply when pattern matching notation is used for file name  
 2547 expansion:

- 2548 1. The application shall ensure that the slash character in a path name is explicitly matched  
 2549 by using one or more slashes in the pattern; it cannot be matched by the asterisk or  
 2550 question-mark special characters or by a bracket expression. Slashes in the pattern are  
 2551 identified before bracket expressions; thus, a slash cannot be included in a pattern bracket  
 2552 expression used for file name expansion. If a slash character is found following an  
 2553 unescaped open square bracket character before a corresponding closing square bracket is  
 2554 found, the open bracket is treated as an ordinary character. For example, the pattern  
 2555 "a[b/c]d" does not match such path names as **abd** or **a/d**. It only matches a path name  
 2556 of literally **a[b/c]d**.
- 2557 2. If a file name begins with a period ( '.' ), the application shall ensure that the period is  
 2558 explicitly matched by using a period as the first character of the pattern or immediately  
 2559 following a slash character. The leading period shall not be matched by:

- 2560 • The asterisk or question-mark special characters
- 2561 • A bracket expression containing a non-matching list, such as "[!a]", a range  
 2562 expression, such as "[%-0]", or a character class expression, such as "[[:punct:]]"

2563 It is unspecified whether an explicit period in a bracket expression matching list, such as  
 2564 "[.abc]", can match a leading period in a file name.

- 2565 3. Specified patterns are matched against existing file names and path names, as appropriate.  
 2566 Each component that contains a pattern character requires read permission in the directory  
 2567 containing that component. Any component, except the last, that does not contain a  
 2568 pattern character requires search permission. For example, given the pattern:

```
2569 /foo/bar/x*/bam
```

2570 search permission is needed for directories / and **foo**, search and read permissions are  
 2571 needed for directory **bar**, and search permission is needed for each **x\*** directory. If the  
 2572 pattern matches any existing file names or path names, the pattern shall be replaced with  
 2573 those file names and path names, sorted according to the collating sequence in effect in the  
 2574 current locale. If the pattern contains an invalid bracket expression or does not match any  
 2575 existing file names or path names, the pattern string shall be left unchanged.

**2.15 Special Built-In Utilities**

The following *special built-in* utilities shall be supported in the shell command language. The output of each command, if any, shall be written to standard output, subject to the normal redirection and piping possible with all commands.

The term *built-in* implies that the shell can execute the utility directly and does not need to search for it. An implementation can choose to make any utility a built-in; however, the special built-in utilities described here differ from regular built-in utilities in two respects:

1. A syntax error in a special built-in utility may cause a shell executing that utility to abort, while a syntax error in a regular built-in utility shall not cause a shell executing that utility to abort. (See Section 2.8.1 (on page 2255) for the consequences of errors on interactive and non-interactive shells.) If a special built-in utility encountering a syntax error does not abort the shell, its exit value shall be non-zero.
2. Variable assignments specified with special built-in utilities remain in effect after the built-in completes; this shall not be the case with a regular built-in or other utility.

The special built-in utilities in this section need not be provided in a manner accessible via the *exec* family of functions defined in the System Interfaces volume of IEEE Std. 1003.1-200x.

Some of the special built-ins are described as conforming to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2, Utility Syntax Guidelines. For those that are not, the requirement in Section 1.11 (on page 2224) that "—" be recognized as a first argument to be discarded does not apply and a portable application shall not use that argument.



2596 **NAME**

2597 break — exit from for, while, or until loop

2598 **SYNOPSIS**2599 break [*n*]2600 **DESCRIPTION**

2601 The *break* utility shall exit from the smallest enclosing **for**, **while**, or **until** loop, if any; or from the  
2602 *n*th enclosing loop if *n* is specified. The value of *n* is an unsigned decimal integer greater than or  
2603 equal to 1. The default shall be equivalent to *n*=1. If *n* is greater than the number of enclosing  
2604 loops, the last enclosing loop shall be exited from. Execution shall continue with the command  
2605 immediately following the loop.

2606 **OPTIONS**

2607 None.

2608 **OPERANDS**

2609 None.

2610 **STDIN**

2611 None.

2612 **INPUT FILES**

2613 None.

2614 **ENVIRONMENT VARIABLES**

2615 None.

2616 **ASYNCHRONOUS EVENTS**

2617 None.

2618 **STDOUT**

2619 None.

2620 **STDERR**

2621 None.

2622 **OUTPUT FILES**

2623 None.

2624 **EXTENDED DESCRIPTION**

2625 None.

2626 **EXIT STATUS**

2627 0 Successful completion.

2628 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.2629 **CONSEQUENCES OF ERRORS**

2630 None.

2631 **APPLICATION USAGE**

2632       None.

2633 **EXAMPLES**

```
2634 for i in * do
2635 if test -d "$i" then break fi done
```

2636 **RATIONALE**

2637       In early proposals, consideration was given to expanding the syntax of *break* and *continue* to refer  
2638       to a label associated with the appropriate loop as a preferable alternative to the *n* method.  
2639       However, this volume of IEEE Std. 1003.1-200x does reserve the namespace of command names  
2640       ending with a colon. It is anticipated that a future implementation could take advantage of this  
2641       and provide something like:

```
2642 outofloop: for i in a b c d e
2643 do
2644 for j in 0 1 2 3 4 5 6 7 8 9
2645 do
2646 if test -r "${i}${j}"
2647 then break outofloop
2648 fi
2649 done
2650 done
```

2651       and that this might be standardized after implementation experience is achieved.

2652 **FUTURE DIRECTIONS**

2653       None.

2654 **SEE ALSO**

2655       Section 2.15 (on page 2276)

2656 **CHANGE HISTORY**

2657       None.

2658 **NAME**  
2659       colon — null utility

2660 **SYNOPSIS**  
2661       : [*argument* ...]

2662 **DESCRIPTION**  
2663       This utility shall only expand command *arguments*. It is used when a command is needed, as in  
2664       the *then* condition of an **if** command, but nothing is to be done by the command.

2665 **OPTIONS**  
2666       None.

2667 **OPERANDS**  
2668       None.

2669 **STDIN**  
2670       None.

2671 **INPUT FILES**  
2672       None.

2673 **ENVIRONMENT VARIABLES**  
2674       None.

2675 **ASYNCHRONOUS EVENTS**  
2676       None.

2677 **STDOUT**  
2678       None.

2679 **STDERR**  
2680       None.

2681 **OUTPUT FILES**  
2682       None.

2683 **EXTENDED DESCRIPTION**  
2684       None.

2685 **EXIT STATUS**  
2686       Zero.

2687 **CONSEQUENCES OF ERRORS**  
2688       None.

2689 **APPLICATION USAGE**  
2690       None.

2691 **EXAMPLES**  
2692       : \${X=abc}  
2693       if       false  
2694       then     :  
2695       else     echo \$X  
2696       fi  
2697       **abc**

2698       As with any of the special built-ins, the null utility can also have variable assignments and  
2699       redirections associated with it, such as:

2700            `x=y : > z`

2701            which sets variable `x` to the value `y` (so that it persists after the null utility completes) and creates  
2702            or truncates file `z`.

2703 **RATIONALE**

2704            None.

2705 **FUTURE DIRECTIONS**

2706            None.

2707 **SEE ALSO**

2708            Section 2.15 (on page 2276)

2709 **CHANGE HISTORY**

2710            None.

2711 **NAME**  
2712 continue — continue for, while, or until loop

2713 **SYNOPSIS**  
2714 continue [*n*]

2715 **DESCRIPTION**  
2716 The *continue* utility shall return to the top of the smallest enclosing **for**, **while**, or **until** loop, or to  
2717 the top of the *n*th enclosing loop, if *n* is specified. This involves repeating the condition list of a  
2718 **while** or **until** loop or performing the next assignment of a **for** loop, and reexecuting the loop if  
2719 appropriate.

2720 The value of *n* is a decimal integer greater than or equal to 1. The default is equivalent to *n*=1. If  
2721 *n* is greater than the number of enclosing loops, the last enclosing loop shall be used.

2722 **OPTIONS**  
2723 None.

2724 **OPERANDS**  
2725 None.

2726 **STDIN**  
2727 None.

2728 **INPUT FILES**  
2729 None.

2730 **ENVIRONMENT VARIABLES**  
2731 None.

2732 **ASYNCHRONOUS EVENTS**  
2733 None.

2734 **STDOUT**  
2735 None.

2736 **STDERR**  
2737 None.

2738 **OUTPUT FILES**  
2739 None.

2740 **EXTENDED DESCRIPTION**  
2741 None.

2742 **EXIT STATUS**  
2743 0 Successful completion.  
2744 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.

2745 **CONSEQUENCES OF ERRORS**  
2746 None.

2747 **APPLICATION USAGE**

2748       None.

2749 **EXAMPLES**

```
2750 for i in *
2751 do
2752 if test -d "$i"
2753 then continue
2754 fi
2755 echo "\"$i\" is not a directory.
2756 done
```

2757 **RATIONALE**

2758       None.

2759 **FUTURE DIRECTIONS**

2760       None.

2761 **SEE ALSO**

2762       Section 2.15 (on page 2276)

2763 **CHANGE HISTORY**

2764       None.

2765 **NAME**  
2766       dot — execute commands in current environment

2767 **SYNOPSIS**  
2768       . *file*

2769 **DESCRIPTION**  
2770       The shell shall execute commands from the *file* in the current environment.  
2771       If *file* does not contain a slash, the shell shall use the search path specified by *PATH* to find the  
2772       directory containing *file*. Unlike normal command search, however, the file searched for by the  
2773       *dot* utility need not be executable. If no readable file is found, a non-interactive shell shall abort;  
2774       an interactive shell shall write a diagnostic message to standard error, but this condition shall  
2775       not be considered a syntax error.

2776 **OPTIONS**  
2777       None.

2778 **OPERANDS**  
2779       None.

2780 **STDIN**  
2781       None.

2782 **INPUT FILES**  
2783       None.

2784 **ENVIRONMENT VARIABLES**  
2785       None.

2786 **ASYNCHRONOUS EVENTS**  
2787       None.

2788 **STDOUT**  
2789       None.

2790 **STDERR**  
2791       None.

2792 **OUTPUT FILES**  
2793       None.

2794 **EXTENDED DESCRIPTION**  
2795       None.

2796 **EXIT STATUS**  
2797       Returns the value of the last command executed, or a zero exit status if no command is executed.

2798 **CONSEQUENCES OF ERRORS**  
2799       None.

2800 **APPLICATION USAGE**  
2801       None.

2802 **EXAMPLES**  
2803       cat foobar  
2804       **foo=hello bar=world**  
2805       . foobar  
2806       echo \$foo \$bar  
2807       **hello world**

**2808 RATIONALE**

2809           Some older implementations searched the current directory for the *file*, even if the value of *PATH*  
2810           disallowed it. This behavior was omitted from this volume of IEEE Std. 1003.1-200x due to  
2811           concerns about introducing the susceptibility to trojan horses that the user might be trying to  
2812           avoid by leaving *dot* out of *PATH*.

2813           The KornShell version of *dot* takes optional arguments that are set to the positional parameters.  
2814           This is a valid extension that allows a *dot* script to behave identically to a function.

**2815 FUTURE DIRECTIONS**

2816           None.

**2817 SEE ALSO**

2818           Section 2.15 (on page 2276)

**2819 CHANGE HISTORY**

2820           None.



2821 **NAME**  
2822       eval — construct command by concatenating arguments

2823 **SYNOPSIS**  
2824       eval [*argument* ...]

2825 **DESCRIPTION**  
2826       The *eval* utility shall construct a command by concatenating *arguments* together, separating each  
2827       with a <space> character. The constructed command shall be read and executed by the shell.

2828 **OPTIONS**  
2829       None.

2830 **OPERANDS**  
2831       None.

2832 **STDIN**  
2833       None.

2834 **INPUT FILES**  
2835       None.

2836 **ENVIRONMENT VARIABLES**  
2837       None.

2838 **ASYNCHRONOUS EVENTS**  
2839       None.

2840 **STDOUT**  
2841       None.

2842 **STDERR**  
2843       None.

2844 **OUTPUT FILES**  
2845       None.

2846 **EXTENDED DESCRIPTION**  
2847       None.

2848 **EXIT STATUS**  
2849       If there are no *arguments*, or only null arguments, *eval* shall return a zero exit status; otherwise, it  
2850       shall return the exit status of the command defined by the string of concatenated *arguments*  
2851       separated by spaces.

2852 **CONSEQUENCES OF ERRORS**  
2853       None.

2854 **APPLICATION USAGE**  
2855       None.

2856 **EXAMPLES**  
2857       foo=10 x=foo  
2858       y=' '\$x  
2859       echo \$y  
2860       **\$foo**  
2861       eval y=' '\$x  
2862       echo \$y  
2863       **10**

2864 **RATIONALE**

2865       None.

2866 **FUTURE DIRECTIONS**

2867       None.

2868 **SEE ALSO**

2869       Section 2.15 (on page 2276)

2870 **CHANGE HISTORY**

2871       None.

2872 **NAME**

2873           exec — execute commands and open, close, or copy file descriptors

2874 **SYNOPSIS**

2875           exec [*command* [*argument* ...]]

2876 **DESCRIPTION**

2877           The *exec* utility shall open, close, and/or copy file descriptors as specified by any redirections as  
2878           part of the command.

2879           If *exec* is specified without *command* or *arguments*, and any file descriptors with numbers greater  
2880           than 2 are opened with associated redirection statements, it is unspecified whether those file  
2881           descriptors remain open when the shell invokes another utility. Scripts concerned that child  
2882           shells could misuse open file descriptors can always close them explicitly, as shown in one of the  
2883           following examples.

2884           If *exec* is specified with *command*, it shall replace the shell with *command* without creating a new  
2885           process. If *arguments* are specified, they shall be arguments to *command*. Redirection affects the  
2886           current shell execution environment.

2887 **OPTIONS**

2888           None.

2889 **OPERANDS**

2890           None.

2891 **STDIN**

2892           None.

2893 **INPUT FILES**

2894           None.

2895 **ENVIRONMENT VARIABLES**

2896           None.

2897 **ASYNCHRONOUS EVENTS**

2898           None.

2899 **STDOUT**

2900           None.

2901 **STDERR**

2902           None.

2903 **OUTPUT FILES**

2904           None.

2905 **EXTENDED DESCRIPTION**

2906           None.

2907 **EXIT STATUS**

2908           If *command* is specified, *exec* shall not return to the shell; rather, the exit status of the process shall  
2909           be the exit status of the program implementing *command*, which overlaid the shell. If *command* is  
2910           not found, the exit status shall be 127. If *command* is found, but it is not an executable utility, the  
2911           exit status shall be 126. If a redirection error occurs (see Section 2.8.1 (on page 2255)), the shell  
2912           shall exit with a value in the range 1–125. Otherwise, *exec* shall return a zero exit status.

2913 **CONSEQUENCES OF ERRORS**

2914 None.

2915 **APPLICATION USAGE**

2916 None.

2917 **EXAMPLES**2918 Open *readfile* as file descriptor 3 for reading:2919 `exec 3< readfile`2920 Open *writefile* as file descriptor 4 for writing:2921 `exec 4> writefile`

2922 Make file descriptor 5 a copy of file descriptor 0:

2923 `exec 5<&0`

2924 Close file descriptor 3:

2925 `exec 3<&-`2926 Cat the file **maggie** by replacing the current shell with the *cat* utility:2927 `exec cat maggie`2928 **RATIONALE**

2929 Most historical implementations were not conformant in that:

2930 `foo=bar exec cmd`2931 did not pass **foo** to **cmd**.2932 **FUTURE DIRECTIONS**

2933 None.

2934 **SEE ALSO**

2935 Section 2.15 (on page 2276)

2936 **CHANGE HISTORY**

2937 None.

2938 **NAME**  
2939        exit — cause the shell to exit

2940 **SYNOPSIS**  
2941        exit [*n*]

2942 **DESCRIPTION**  
2943        The *exit* utility shall cause the shell to exit with the exit status specified by the unsigned decimal  
2944        integer *n*. If *n* is specified, but its value is not between 0 and 255 inclusively, the exit status is  
2945        undefined.

2946        A *trap* on **EXIT** shall be executed before the shell terminates, except when the *exit* utility is  
2947        invoked in that *trap* itself, in which case the shell shall exit immediately.

2948 **OPTIONS**  
2949        None.

2950 **OPERANDS**  
2951        None.

2952 **STDIN**  
2953        None.

2954 **INPUT FILES**  
2955        None.

2956 **ENVIRONMENT VARIABLES**  
2957        None.

2958 **ASYNCHRONOUS EVENTS**  
2959        None.

2960 **STDOUT**  
2961        None.

2962 **STDERR**  
2963        None.

2964 **OUTPUT FILES**  
2965        None.

2966 **EXTENDED DESCRIPTION**  
2967        None.

2968 **EXIT STATUS**  
2969        The exit status shall be *n*, if specified. Otherwise, the value shall be the exit value of the last  
2970        command executed, or zero if no command was executed. When *exit* is executed in a *trap* action,  
2971        the last command is considered to be the command that executed immediately preceding the  
2972        *trap* action.

2973 **CONSEQUENCES OF ERRORS**  
2974        None.

2975 **APPLICATION USAGE**  
2976        None.

2977 **EXAMPLES**  
2978        Exit with a *true* value:  
2979        exit 0

2980 Exit with a *false* value:

2981 `exit 1`

2982 **RATIONALE**

2983 As explained in other sections, certain exit status values have been reserved for special uses and  
2984 should be used by applications only for those purposes:

2985 126 A file to be executed was found, but it was not an executable utility.

2986 127 A utility to be executed was not found.

2987 >128 A command was interrupted by a signal.

2988 **FUTURE DIRECTIONS**

2989 None.

2990 **SEE ALSO**

2991 Section 2.15 (on page 2276)

2992 **CHANGE HISTORY**

2993 None.

2994 **NAME**

2995            export — set export attribute for variables

2996 **SYNOPSIS**2997            export name[=*word*]...

2998            export -p

2999 **DESCRIPTION**3000            The shell shall give the export attribute to the variables corresponding to the specified *names*,  
3001            which shall cause them to be in the environment of subsequently executed commands.3002            The *export* special built-in shall support the Base Definitions volume of IEEE Std. 1003.1-200x,  
3003            Section 12.2, Utility Syntax Guidelines.3004            When **-p** is specified, *export* shall write to the standard output the names and values of all  
3005            exported variables, in the following format:

3006            "export %s=%s\n", &lt;name&gt;, &lt;value&gt;

3007            The shell shall format the output, including the proper use of quoting, so that it is suitable for  
3008            reinput to the shell as commands that achieve the same exporting results.

3009            When no arguments are given, the results are unspecified.

3010 **OPTIONS**

3011            None.

3012 **OPERANDS**

3013            None.

3014 **STDIN**

3015            None.

3016 **INPUT FILES**

3017            None.

3018 **ENVIRONMENT VARIABLES**

3019            None.

3020 **ASYNCHRONOUS EVENTS**

3021            None.

3022 **STDOUT**

3023            None.

3024 **STDERR**

3025            None.

3026 **OUTPUT FILES**

3027            None.

3028 **EXTENDED DESCRIPTION**

3029            None.

3030 **EXIT STATUS**

3031            Zero.

3032 **CONSEQUENCES OF ERRORS**

3033 None.

3034 **APPLICATION USAGE**

3035 None.

3036 **EXAMPLES**3037 Export *PWD* and *HOME* variables:3038 `export PWD HOME`3039 Set and export the *PATH* variable:3040 `export PATH=/local/bin:$PATH`

3041 Save and restore all exported variables:

3042 `export -p > temp-file`3043 `unset a lot of variables`3044 `... processing`3045 `. temp-file`3046 **RATIONALE**

3047 Some historical shells use the no-argument case as the functional equivalent of what is required  
3048 here with **-p**. This feature was left unspecified because it is not historical practice in all shells,  
3049 and some scripts may rely on the now-unspecified results on their implementations. Attempts to  
3050 specify the **-p** output as the default case were unsuccessful in achieving consensus. The **-p**  
3051 option was added to allow portable access to the values that can be saved and then later restored  
3052 using; for example, a *dot* script.

3053 **FUTURE DIRECTIONS**

3054 None.

3055 **SEE ALSO**

3056 Section 2.15 (on page 2276)

3057 **CHANGE HISTORY**

3058 None.



3059 **NAME**

3060       readonly — set read-only attribute for variables

3061 **SYNOPSIS**3062       readonly name[=*word*]...

3063       readonly -p

3064 **DESCRIPTION**

3065       The variables whose *names* are specified shall be given the *readonly* attribute. The values of  
3066       variables with the *readonly* attribute cannot be changed by subsequent assignment, nor can those  
3067       variables be unset by the *unset* utility.

3068       The *readonly* special built-in shall support the Base Definitions volume of IEEE Std. 1003.1-200x,  
3069       Section 12.2, Utility Syntax Guidelines.

3070       When **-p** is specified, *readonly* writes to the standard output the names and values of all read-  
3071       only variables, in the following format:

3072       "readonly %s=%s\n", &lt;name&gt;, &lt;value&gt;

3073       The shell shall format the output, including the proper use of quoting, so that it is suitable for  
3074       reinput to the shell as commands that achieve the same attribute-setting results.

3075       When no arguments are given, the results are unspecified.

3076 **OPTIONS**

3077       None.

3078 **OPERANDS**

3079       None.

3080 **STDIN**

3081       None.

3082 **INPUT FILES**

3083       None.

3084 **ENVIRONMENT VARIABLES**

3085       None.

3086 **ASYNCHRONOUS EVENTS**

3087       None.

3088 **STDOUT**

3089       None.

3090 **STDERR**

3091       None.

3092 **OUTPUT FILES**

3093       None.

3094 **EXTENDED DESCRIPTION**

3095       None.

3096 **EXIT STATUS**

3097       Zero.

3098 **CONSEQUENCES OF ERRORS**

3099 None.

3100 **APPLICATION USAGE**

3101 None.

3102 **EXAMPLES**3103 `readonly HOME PWD`3104 **RATIONALE**

3105 Some historical shells preserve the read-only attribute across separate invocations. This volume  
3106 of IEEE Std. 1003.1-200x allows this behavior, but does not require it.

3107 The `-p` option allows portable access to the values that can be saved and then later restored  
3108 using; for example, a *dot* script. Also see the RATIONALE for *export* (on page 2291) for a  
3109 description of the no-argument and `-p` output cases and a related example.

3110 Read-only functions were considered, but they were omitted as not being historical practice or  
3111 particularly useful. Furthermore, functions must not be *readonly* across invocations to preclude  
3112 *spoofing* (spoofing is the term for the practice of creating a program that acts like a well-known  
3113 utility with the intent of subverting the real intent of the user) of administrative or security-  
3114 relevant (or security-conscious) shell scripts.

3115 **FUTURE DIRECTIONS**

3116 None.

3117 **SEE ALSO**

3118 Section 2.15 (on page 2276)

3119 **CHANGE HISTORY**

3120 None.

3121 **NAME**

3122           return — return from a function

3123 **SYNOPSIS**3124           return [*n*]3125 **DESCRIPTION**3126           The *return* utility shall cause the shell to stop executing the current function or *dot* script. If the  
3127           shell is not currently executing a function or *dot* script, the results are unspecified.3128 **OPTIONS**

3129           None.

3130 **OPERANDS**

3131           None.

3132 **STDIN**

3133           None.

3134 **INPUT FILES**

3135           None.

3136 **ENVIRONMENT VARIABLES**

3137           None.

3138 **ASYNCHRONOUS EVENTS**

3139           None.

3140 **STDOUT**

3141           None.

3142 **STDERR**

3143           None.

3144 **OUTPUT FILES**

3145           None.

3146 **EXTENDED DESCRIPTION**

3147           None.

3148 **EXIT STATUS**3149           The value of the special parameter '?' shall be set to *n*, an unsigned decimal integer, or to the  
3150           exit status of the last command executed if *n* is not specified. If the value of *n* is greater than 255,  
3151           the results are undefined. When *return* is executed in a *trap* action, the last command is  
3152           considered to be the command that executed immediately preceding the *trap* action.3153 **CONSEQUENCES OF ERRORS**

3154           None.

3155 **APPLICATION USAGE**

3156           None.

3157 **EXAMPLES**

3158           None.

3159 **RATIONALE**3160           The behavior of *return* when not in a function or *dot* script differs between the System V shell  
3161           and the KornShell. In the System V shell this is an error, whereas in the KornShell, the effect is  
3162           the same as *exit*.

3163           The results of returning a number greater than 255 are undefined because of differing practices  
3164           in the various historical implementations. Some shells AND out all but the low-order 8 bits;  
3165           others allow larger values, but not of unlimited size.

3166           See the discussion of appropriate exit status values under *exit* (on page 2289).

3167 **FUTURE DIRECTIONS**

3168           None.

3169 **SEE ALSO**

3170           Section 2.15 (on page 2276)

3171 **CHANGE HISTORY**

3172           None.

3173 **NAME**

3174 set — set or unset options and positional parameters

3175 **SYNOPSIS**3176 xSI set [-abCefmnuvx] [-h] [-o *option*] [*argument...*]3177 xSI set [+abCefmnuvx] [+h] [+o *option*] [*argument...*]3178 set — [*argument...*]

3179 set -o

3180 set +o

3181 **DESCRIPTION**

3182 If no options or *arguments* are specified, *set* shall write the names and values of all shell variables  
 3183 in the collation sequence of the current locale. Each *name* shall start on a separate line, using the  
 3184 format:

3185 "%s=%s\n", <*name*>, <*value*>

3186 The *value* string shall be written with appropriate quoting so that it is suitable for reinput to the  
 3187 shell, setting or resetting, as far as possible, the variables that are currently set. Read-only  
 3188 variables cannot be reset; see the description of shell quoting in Section 2.2 (on page 2236).

3189 When options are specified, they shall set or unset attributes of the shell, as described below.  
 3190 When *arguments* are specified, they cause positional parameters to be set or unset, as described  
 3191 below. Setting or unsetting attributes and positional parameters are not necessarily related  
 3192 actions, but they can be combined in a single invocation of *set*.

3193 The *set* special built-in shall support the Base Definitions volume of IEEE Std. 1003.1-200x,  
 3194 Section 12.2, Utility Syntax Guidelines except that options can be specified with either a leading  
 3195 hyphen (meaning enable the option) or plus sign (meaning disable it).

3196 Implementations shall support the options in the following list in both their hyphen and plus-  
 3197 sign forms. These options can also be specified as options to *sh*.

3198 **-a** When this option is on, the export attribute shall be set for each variable to which an  
 3199 assignment is performed; see the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 3200 4.16, Variable Assignment. If the assignment precedes a utility name in a command, the  
 3201 export attribute shall not persist in the current execution environment after the utility  
 3202 completes, with the exception that preceding one of the special built-in utilities causes the  
 3203 export attribute to persist after the built-in has completed. If the assignment does not  
 3204 precede a utility name in the command, or if the assignment is a result of the operation of  
 3205 the *getopts* or *read* utilities, the export attribute shall persist until the variable is unset.

3206 **-b** This option is supported if the system supports the User Portability Utilities option. It shall  
 3207 cause the shell to notify the user asynchronously of background job completions. The  
 3208 following message is written to standard error:

3209 "[%d]%c %s%s\n", <*job-number*>, <*current*>, <*status*>, <*job-name*>

3210 where the fields shall be as follows:

3211 <*current*> The character '+' identifies the job that would be used as a default for  
 3212 the *fg* or *bg* utilities; this job can also be specified using the *job\_id* "%+" or  
 3213 "%%". The character '-' identifies the job that would become the default  
 3214 if the current default job were to exit; this job can also be specified using  
 3215 the *job\_id* "%-". For other jobs, this field is a <space> character. At most  
 3216 one job can be identified with '+' and at most one job can be identified

- 3217 with '-'. If there is any suspended job, then the current job shall be a  
 3218 suspended job. If there are at least two suspended jobs, then the previous  
 3219 job also shall be a suspended job.
- 3220 <job-number> A number that can be used to identify the process group to the *wait*, *fg*, *bg*,  
 3221 and *kill* utilities. Using these utilities, the job can be identified by  
 3222 prefixing the job number with '%'.  
 3223 <status> Unspecified.  
 3224 <job-name> Unspecified.
- 3225 When the shell notifies the user a job has been completed, it may remove the job's process  
 3226 ID from the list of those known in the current shell execution environment; see Section  
 3227 2.9.3.1 (on page 2259). Asynchronous notification shall not be enabled by default.
- 3228 -C (Uppercase C.) Prevent existing files from being overwritten by the shell's '>' redirection  
 3229 operator (see Section 2.7.2 (on page 2252)); the ">|" redirection operator shall override this  
 3230 *noclobber* option for an individual file.
- 3231 -e When this option is on, if a simple command fails for any of the reasons listed in Section  
 3232 2.8.1 (on page 2255) or returns an exit status value >0, and is not part of the compound list  
 3233 following a **while**, **until**, or **if** keyword, and is not a part of an AND or OR list, and is not a  
 3234 pipeline preceded by the ! reserved word, then the shell shall immediately exit.
- 3235 -f The shell shall disable path name expansion.
- 3236 XSI -h Locate and remember utilities invoked by functions as those functions are defined (the  
 3237 utilities are normally located when the function is executed).
- 3238 -m This option is supported if the system supports the User Portability Utilities option. All jobs  
 3239 shall be run in their own process groups. Immediately before the shell issues a prompt after  
 3240 completion of the background job, a message reporting the exit status of the background job  
 3241 shall be written to standard error. If a foreground job stops, the shell shall write a message  
 3242 to standard error to that effect, formatted as described by the *jobs* utility. In addition, if a job  
 3243 changes status other than exiting (for example, if it stops for input or output or is stopped  
 3244 by a SIGSTOP signal), the shell shall write a similar message immediately prior to writing  
 3245 the next prompt. This option is enabled by default for interactive shells.
- 3246 -n The shell shall read commands but does not execute them; this can be used to check for  
 3247 shell script syntax errors. An interactive shell may ignore this option.
- 3248 -o Write the current settings of the options to standard output in an unspecified format.
- 3249 +o Write the current option settings to standard output in a format that is suitable for reinput  
 3250 to the shell as commands that achieve the same options settings.
- 3251 -o *option*  
 3252 This option is supported if the system supports the User Portability Utilities option. It shall  
 3253 set various options, many of which shall be equivalent to the single option letters. The  
 3254 following values of *option* shall be supported:
- 3255 *allexport* Equivalent to -a.  
 3256 *errexit* Equivalent to -e.  
 3257 *ignoreeof* Prevent an interactive shell from exiting on end-of-file. This setting prevents  
 3258 accidental logouts when <control>-D is entered. A user shall explicitly *exit* to  
 3259 leave the interactive shell.

|      |                              |                                                                                                                    |
|------|------------------------------|--------------------------------------------------------------------------------------------------------------------|
| 3260 | <i>monitor</i>               | Equivalent to <b>-m</b> . This option is supported if the system supports the User Portability Utilities option.   |
| 3261 |                              |                                                                                                                    |
| 3262 | <i>noclobber</i>             | Equivalent to <b>-C</b> (uppercase C).                                                                             |
| 3263 | <i>noglob</i>                | Equivalent to <b>-f</b> .                                                                                          |
| 3264 | <i>noexec</i>                | Equivalent to <b>-n</b> .                                                                                          |
| 3265 | <i>nolog</i>                 | Prevent the entry of function definitions into the command history; see                                            |
| 3266 |                              | <b>Command History List</b> (on page 3064).                                                                        |
| 3267 | <i>notify</i>                | Equivalent to <b>-b</b> .                                                                                          |
| 3268 | <i>nounset</i>               | Equivalent to <b>-u</b> .                                                                                          |
| 3269 | <i>verbose</i>               | Equivalent to <b>-v</b> .                                                                                          |
| 3270 | <i>vi</i>                    | Allow shell command line editing using the built-in <i>vi</i> editor. Enabling <i>vi</i>                           |
| 3271 |                              | mode shall disable any other command line editing mode provided as an                                              |
| 3272 |                              | implementation extension.                                                                                          |
| 3273 |                              | It need not be possible to set <i>vi</i> mode on for certain block-mode terminals.                                 |
| 3274 | <i>xtrace</i>                | Equivalent to <b>-x</b> .                                                                                          |
| 3275 | <b>-u</b>                    | The shell writes a message to standard error when it tries to expand a variable that is not set                    |
| 3276 |                              | and immediately exit. An interactive shell shall not exit.                                                         |
| 3277 | <b>-v</b>                    | The shell writes its input to standard error as it is read.                                                        |
| 3278 | <b>-x</b>                    | The shell writes to standard error a trace for each command after it expands the command                           |
| 3279 |                              | and before it executes it. It is unspecified whether the command that turns tracing off is                         |
| 3280 |                              | traced.                                                                                                            |
| 3281 |                              | The default for all these options is off (unset) unless the shell was invoked with them on; see <i>sh</i> .        |
| 3282 |                              | The remaining arguments shall be assigned in order to the positional parameters. The special                       |
| 3283 |                              | parameter ' <b>#</b> ' shall be set to reflect the number of positional parameters. All positional                 |
| 3284 |                              | parameters shall be unset before any new values are assigned.                                                      |
| 3285 |                              | The special argument " <b>---</b> " immediately following the <i>set</i> command name can be used to delimit       |
| 3286 |                              | the arguments if the first argument begins with ' <b>+</b> ' or ' <b>-</b> ', or to prevent inadvertent listing of |
| 3287 |                              | all shell variables when there are no arguments. The command <i>set--</i> without <i>argument</i> shall            |
| 3288 |                              | unset all positional parameters and set the special parameter ' <b>#</b> ' to zero.                                |
| 3289 | <b>OPTIONS</b>               |                                                                                                                    |
| 3290 |                              | None.                                                                                                              |
| 3291 | <b>OPERANDS</b>              |                                                                                                                    |
| 3292 |                              | None.                                                                                                              |
| 3293 | <b>STDIN</b>                 |                                                                                                                    |
| 3294 |                              | None.                                                                                                              |
| 3295 | <b>INPUT FILES</b>           |                                                                                                                    |
| 3296 |                              | None.                                                                                                              |
| 3297 | <b>ENVIRONMENT VARIABLES</b> |                                                                                                                    |
| 3298 |                              | None.                                                                                                              |

3299 **ASYNCHRONOUS EVENTS**

3300 None.

3301 **STDOUT**

3302 None.

3303 **STDERR**

3304 None.

3305 **OUTPUT FILES**

3306 None.

3307 **EXTENDED DESCRIPTION**

3308 None.

3309 **EXIT STATUS**

3310 Zero.

3311 **CONSEQUENCES OF ERRORS**

3312 None.

3313 **APPLICATION USAGE**

3314 None.

3315 **EXAMPLES**

3316 Write out all variables and their values:

3317 `set`

3318 Set \$1, \$2, and \$3 and set "\$#" to 3:

3319 `set c a b`3320 Turn on the `-x` and `-v` options:3321 `set -xv`

3322 Unset all positional parameters:

3323 `set --`3324 Set \$1 to the value of `-x`, even if `x` begins with `'-'` or `'+'`:3325 `set -- "$x"`3326 Set the positional parameters to the expansion of `x`, even if `x` expands with a leading `'-'` or `'+'`:3327 `set -- $x`3328 **RATIONALE**

3329 The `set --` form is listed specifically in the SYNOPSIS even though this usage is implied by the  
 3330 Utility Syntax Guidelines. The explanation of this feature removes any ambiguity about whether  
 3331 the `set --` form might be misinterpreted as being equivalent to `set` without any options or  
 3332 arguments. The functionality of this form has been adopted from the KornShell. In System V, `set`  
 3333 `--` only unsets parameters if there is at least one argument; the only way to unset all parameters  
 3334 is to use *shift*. Using the KornShell version should not affect System V scripts because there  
 3335 should be no reason to issue it without arguments deliberately; if it were issued as, for example:

3336 `set -- "$@"`

3337 and there were in fact no arguments resulting from `"$@"`, unsetting the parameters would have  
 3338 no result.



3339 The *set* + form in early proposals was omitted as being an unnecessary duplication of *set* alone  
3340 and not widespread historical practice.

3341 The *noclobber* option was changed to allow *set -C* as well as the *set -o noclobber* option. The  
3342 single-letter version was added so that the historical "\$-" paradigm would not be broken; see  
3343 Section 2.5.2 (on page 2241).

3344 The *-h* flag is related to command name hashing and is only required on XSI-conformant  
3345 systems.

3346 The following *set* flags were omitted intentionally with the following rationale:

3347 **-k** The *-k* flag was originally added by the author of the Bourne shell to make it easier for  
3348 users of pre-release versions of the shell. In early versions of the Bourne shell the construct  
3349 *set name=value*, had to be used to assign values to shell variables. The problem with *-k* is  
3350 that the behavior affects parsing, virtually precluding writing any compilers. To explain the  
3351 behavior of *-k*, it is necessary to describe the parsing algorithm, which is implementation-  
3352 defined. For example:

```
3353 set -k; echo name=value
```

3354 and:

```
3355 set x--k
3356 echo name=value
```

3357 behave differently. The interaction with functions is even more complex. What is more, the  
3358 *-k* flag is never needed, since the command line could have been reordered.

3359 **-t** The *-t* flag is hard to specify and almost never used. The only known use could be done  
3360 with here-documents. Moreover, the behavior with *ksh* and *sh* differs. The reference page  
3361 says that it exits after reading and executing one command. What is one command? If the  
3362 input is *date;date*, *sh* executes both *date* commands while *ksh* does only the first.

3363 Consideration was given to rewriting *set* to simplify its confusing syntax. A specific suggestion  
3364 was that the *unset* utility should be used to unset options instead of using the non-*getopt()*-able  
3365 *+option* syntax. However, the conclusion was reached that the historical practice of using *+option*  
3366 was satisfactory and that there was no compelling reason to modify such widespread historical  
3367 practice.

3368 The *-o* option was adopted from the KornShell to address user needs. In addition to its generally  
3369 friendly interface, *-o* is needed to provide the *vi* command line editing mode, for which  
3370 historical practice yields no single-letter option name. (Although it might have been possible to  
3371 invent such a letter, it was recognized that other editing modes would be developed and *-o*  
3372 provides ample name space for describing such extensions.)

3373 Historical implementations are inconsistent in the format used for *-o* option status reporting.  
3374 The *+o* format without an option-argument was added to allow portable access to the options  
3375 that can be saved and then later restored using, for instance, a dot script.

3376 Historically, *sh* did trace the command *set +x*, but *ksh* did not.

3377 The *ignoreeof* setting prevents accidental logouts when the end-of-file character (typically  
3378 <control>-D) is entered. A user shall explicitly *exit* to leave the interactive shell.

3379 The *set -m* option was added to apply only to the UPE because it applies primarily to interactive  
3380 use, not shell script applications.

3381 The ability to do asynchronous notification became available in the 1988 version of the  
3382 KornShell. To have it occur, the user had to issue the command:

```
3383 trap "jobs -n" CLD
```

3384 The C shell provides two different levels of an asynchronous notification capability. The  
3385 environment variable *notify* is analogous to what is done in *set -b* or *set -o notify*. When set, it  
3386 notifies the user immediately of background job completions. When unset, this capability is  
3387 turned off.

3388 The other notification ability comes through the built-in utility *notify*. The syntax is:

```
3389 notify [%job ...]
```

3390 By issuing *notify* with no operands, it causes the C shell to notify the user asynchronously when  
3391 the state of the current job changes. If given operands, *notify* asynchronously informs the user of  
3392 changes in the states of the specified jobs.

3393 To add asynchronous notification to the POSIX shell, neither the KornShell extensions to *trap*,  
3394 nor the C shell *notify* environment variable seemed appropriate (*notify* is not a proper POSIX  
3395 environment variable name).

3396 The *set -b* option was selected as a compromise.

3397 The *notify* built-in was considered to have more functionality than was required for simple  
3398 asynchronous notification.

#### 3399 **FUTURE DIRECTIONS**

3400 None.

#### 3401 **SEE ALSO**

3402 Section 2.15 (on page 2276)

#### 3403 **CHANGE HISTORY**

##### 3404 **Issue 6**

3405 The obsolescent *set* command name followed by *'-'* has been removed.

3406 The following new requirements on POSIX implementations derive from alignment with the  
3407 Single UNIX Specification:

- 3408 • The *nolog* option is added to *set -o*.

3409 **NAME**

3410 shift — shift positional parameters

3411 **SYNOPSIS**3412 shift [*n*]3413 **DESCRIPTION**

3414 The positional parameters shall be shifted. Positional parameter 1 shall be assigned the value of  
3415 parameter (1+*n*), parameter 2 shall be assigned the value of parameter (2+*n*), and so on. The  
3416 parameters represented by the numbers "\$#" down to "\$#-*n*+1" shall be unset, and the  
3417 parameter '#' is updated to reflect the new number of positional parameters.

3418 The value *n* shall be an unsigned decimal integer less than or equal to the value of the special  
3419 parameter '#'. If *n* is not given, it shall be assumed to be 1. If *n* is 0, the positional and special  
3420 parameters are not changed.

3421 **OPTIONS**

3422 None.

3423 **OPERANDS**

3424 None.

3425 **STDIN**

3426 None.

3427 **INPUT FILES**

3428 None.

3429 **ENVIRONMENT VARIABLES**

3430 None.

3431 **ASYNCHRONOUS EVENTS**

3432 None.

3433 **STDOUT**

3434 None.

3435 **STDERR**

3436 None.

3437 **OUTPUT FILES**

3438 None.

3439 **EXTENDED DESCRIPTION**

3440 None.

3441 **EXIT STATUS**3442 The exit status is >0 if *n*>\$#; otherwise, it is zero.3443 **CONSEQUENCES OF ERRORS**

3444 None.

3445 **APPLICATION USAGE**

3446       None.

3447 **EXAMPLES**

3448       \$ set a b c d e

3449       \$ shift 2

3450       \$ echo \$\*

3451       c d e

3452 **RATIONALE**

3453       None.

3454 **FUTURE DIRECTIONS**

3455       None.

3456 **SEE ALSO**

3457       Section 2.15 (on page 2276)

3458 **CHANGE HISTORY**

3459       None.

3460 **NAME**

3461 times — write process times

3462 **SYNOPSIS**

3463 times

3464 **DESCRIPTION**3465 Write the accumulated user and system times for the shell and for all of its child processes, in the  
3466 following POSIX locale format:

```
3467 "%dm%fs %dm%fs\n%dm%fs %dm%fs\n", <shell user minutes>,
3468 <shell user seconds>, <shell system minutes>,
3469 <shell system seconds>, <children user minutes>,
3470 <children user seconds>, <children system minutes>,
3471 <children system seconds>
```

3472 The four pairs of times correspond to the members of the `<sys/times.h>` `tms` structure (defined  
3473 in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 13, Headers) as returned by  
3474 `times()`: `tms_utime`, `tms_stime`, `tms_cutime`, and `tms_cstime`, respectively.3475 **OPTIONS**

3476 None.

3477 **OPERANDS**

3478 None.

3479 **STDIN**

3480 None.

3481 **INPUT FILES**

3482 None.

3483 **ENVIRONMENT VARIABLES**

3484 None.

3485 **ASYNCHRONOUS EVENTS**

3486 None.

3487 **STDOUT**

3488 None.

3489 **STDERR**

3490 None.

3491 **OUTPUT FILES**

3492 None.

3493 **EXTENDED DESCRIPTION**

3494 None.

3495 **EXIT STATUS**

3496 Zero.

3497 **CONSEQUENCES OF ERRORS**

3498 None.

3499 **APPLICATION USAGE**

3500 None.

3501 **EXAMPLES**

3502 \$ times

3503 0m0.43s 0m1.11s

3504 8m44.18s 1m43.23s

3505 **RATIONALE**3506 The *times* special built-in from the Single UNIX Specification is now required for all conforming  
3507 shells.3508 **FUTURE DIRECTIONS**

3509 None.

3510 **SEE ALSO**

3511 Section 2.15 (on page 2276)

3512 **CHANGE HISTORY**

3513 None.

3514 **NAME**

3515 trap — trap signals

3516 **SYNOPSIS**3517 trap [*action condition ...*]3518 **DESCRIPTION**

3519 If *action* is '-', the shell shall reset each *condition* to the default value. If *action* is null (" "), the  
 3520 shell shall ignore each specified *condition* if it arises. Otherwise, the argument *action* shall be read  
 3521 and executed by the shell when one of the corresponding conditions arises. The action of *trap*  
 3522 shall override a previous action (either default action or one explicitly set). The value of "\$?"  
 3523 after the *trap* action completes shall be the value it had before *trap* was invoked.

3524 The condition can be EXIT, 0 (equivalent to EXIT), or a signal specified using a symbolic name,  
 3525 without the SIG prefix, as listed in the tables of signal names in the <signal.h> header defined in  
 3526 the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 13, Headers; for example, HUP,  
 3527 INT, QUIT, TERM. Implementations may permit lowercase signal names or names with the SIG  
 3528 prefix as an extension. Setting a trap for SIGKILL or SIGSTOP produces undefined results.

3529 The environment in which the shell executes a *trap* on EXIT shall be identical to the environment  
 3530 immediately after the last command executed before the *trap* on EXIT was taken.

3531 Each time *trap* is invoked, the *action* argument shall be processed in a manner equivalent to:

3532 eval "\$action"

3533 Signals that were ignored on entry to a non-interactive shell cannot be trapped or reset, although  
 3534 no error need be reported when attempting to do so. An interactive shell may reset or catch  
 3535 signals ignored on entry. Traps shall remain in place for a given shell until explicitly changed  
 3536 with another *trap* command.

3537 When a subshell is entered, traps that are not being ignored are set to the default actions. This  
 3538 does not imply that the *trap* command cannot be used within the subshell to set new traps.

3539 The *trap* command with no arguments shall write to standard output a list of commands  
 3540 associated with each condition. The format shall be:

3541 "trap — %s %s ... \n", &lt;action&gt;, &lt;condition&gt; ...

3542 The shell shall format the output, including the proper use of quoting, so that it is suitable for  
 3543 reinput to the shell as commands that achieve the same trapping results. For example:

3544 save\_traps=\$(trap)

3545 ...

3546 eval "\$save\_traps"

3547 XSI the following signal names:  
 3548

3549

3550

3551 XSI

3552 XSI

3553 XSI

3554 XSI

3555 XSI

3556 XSI

3557 XSI

| Signal Number | Signal Name |
|---------------|-------------|
| 1             | SIGHUP      |
| 2             | SIGINT      |
| 3             | SIGQUIT     |
| 6             | SIGABRT     |
| 9             | SIGKILL     |
| 14            | SIGALRM     |
| 15            | SIGTERM     |

3558

3559

The *trap* special built-in shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

3560 **OPTIONS**

3561 None.

3562 **OPERANDS**

3563 None.

3564 **STDIN**

3565 None.

3566 **INPUT FILES**

3567 None.

3568 **ENVIRONMENT VARIABLES**

3569 None.

3570 **ASYNCHRONOUS EVENTS**

3571 None.

3572 **STDOUT**

3573 None.

3574 **STDERR**

3575 None.

3576 **OUTPUT FILES**

3577 None.

3578 **EXTENDED DESCRIPTION**

3579 None.

3580 **EXIT STATUS**

3581 XSI If the trap name or number is invalid, a non-zero exit status shall be returned; otherwise, zero

3582 XSI shall be returned. For both interactive and non-interactive shells, invalid signal names or

3583 numbers shall not be considered a syntax error and do not cause the shell to abort.

3584 **CONSEQUENCES OF ERRORS**

3585 None.

3586 **APPLICATION USAGE**

3587 None.

3588 **EXAMPLES**

3589 Write out a list of all traps and actions:

3590 trap

3591 Set a trap so the *logout* utility in the directory referred to by the *HOME* environment variable

3592 executes when the shell terminates:



3593 `trap '$HOME/logout' EXIT`

3594 `or:`

3595 `trap '$HOME/logout' 0`

3596 Unset traps on INT, QUIT, TERM, and EXIT:

3597 `trap - INT QUIT TERM EXIT`

#### 3598 **RATIONALE**

3599 Implementations may permit lowercase signal names as an extension. Implementations may  
3600 also accept the names with the SIG prefix; no known historical shell does so. The *trap* and *kill*  
3601 utilities in this volume of IEEE Std. 1003.1-200x are now consistent in their omission of the SIG  
3602 prefix for signal names. Some *kill* implementations do not allow the prefix, and *kill -l* lists the  
3603 signals without prefixes.

3604 Trapping SIGKILL or SIGSTOP is syntactically accepted by some historical implementations, but  
3605 it has no effect. Portable POSIX applications cannot attempt to trap these signals.

3606 The output format is not historical practice. Since the output of historical *trap* commands is not  
3607 portable (because numeric signal values are not portable) and had to change to become so, an  
3608 opportunity was taken to format the output in a way that a shell script could use to save and  
3609 then later reuse a trap if it wanted.

3610 The KornShell uses an **ERR** trap that is triggered whenever *set -e* would cause an exit. This is  
3611 allowable as an extension, but was not mandated, as other shells have not used it.

3612 The text about the environment for the EXIT trap invalidates the behavior of some historical  
3613 versions of interactive shells which, for example, close the standard input before executing a  
3614 trap on 0. For example, in some historical interactive shell sessions the following trap on 0 would  
3615 always print "—":

3616 `trap 'read foo; echo "$foo—" ' 0`

#### 3617 **FUTURE DIRECTIONS**

3618 None.

#### 3619 **SEE ALSO**

3620 Section 2.15 (on page 2276)

#### 3621 **CHANGE HISTORY**

##### 3622 **Issue 6**

3623 XSI-conforming implementations provide the mapping of signal names to numbers given above  
3624 (previously this had been marked obsolescent). Other implementations need not provide this  
3625 optional mapping.

3626 **NAME**

3627           unset — unset values and attributes of variables and functions

3628 **SYNOPSIS**3629           unset [-fv] *name* ...3630 **DESCRIPTION**3631           Each variable or function specified by *name* shall be unset.3632           If **-v** is specified, *name* refers to a variable name and the shell shall unset it and remove it from  
3633           the environment. Read-only variables cannot be unset.3634           If **-f** is specified, *name* refers to a function and the shell shall unset the function definition.3635           If neither **-f** nor **-v** is specified, *name* refers to a variable; if a variable by that name does not  
3636           exist, it is unspecified whether a function by that name, if any, shall be unset.3637           Unsetting a variable or function that was not previously set shall not be considered an error and  
3638           does not cause the shell to abort.3639           The *unset* special built-in shall support the Base Definitions volume of IEEE Std. 1003.1-200x,  
3640           Section 12.2, Utility Syntax Guidelines.

3641           Note that:

3642           VARIABLE=

3643           is not equivalent to an *unset* of **VARIABLE**; in the example, **VARIABLE** is set to " ". Also, the  
3644           variables that can be *unset* should not be misinterpreted to include the special parameters (see  
3645           Section 2.5.2 (on page 2241)).3646 **OPTIONS**

3647           None.

3648 **OPERANDS**

3649           None.

3650 **STDIN**

3651           None.

3652 **INPUT FILES**

3653           None.

3654 **ENVIRONMENT VARIABLES**

3655           None.

3656 **ASYNCHRONOUS EVENTS**

3657           None.

3658 **STDOUT**

3659           None.

3660 **STDERR**

3661           None.

3662 **OUTPUT FILES**

3663           None.

3664 **EXTENDED DESCRIPTION**

3665           None.

3666 **EXIT STATUS**3667           0 All *name* operands were successfully unset.3668           >0 At least one *name* could not be unset.3669 **CONSEQUENCES OF ERRORS**

3670           None.

3671 **APPLICATION USAGE**

3672           None.

3673 **EXAMPLES**3674           Unset *VISUAL* variable:

3675                 unset -v VISUAL

3676           Unset the functions **foo** and **bar**:

3677                 unset -f foo bar

3678 **RATIONALE**3679           Consideration was given to omitting the **-f** option in favor of an *unfunction* utility, but the  
3680           standard developers decided to retain historical practice.3681           The **-v** option was introduced because System V historically used one name space for both  
3682           variables and functions. When *unset* is used without options, System V historically unset either a  
3683           function or a variable, and there was no confusion about which one was intended. A portable  
3684           POSIX application can use *unset* without an option to unset a variable, but not a function; the **-f**  
3685           option must be used.3686 **FUTURE DIRECTIONS**

3687           None.

3688 **SEE ALSO**

3689           Section 2.15 (on page 2276)

3690 **CHANGE HISTORY**

3691           None.



## Batch Environment Services

3693

3694 BE This chapter describes the services and utilities that shall be implemented on all systems that  
3695 claim conformance to the Batch Environment option. This functionality is dependent on support  
3696 of this option (and the rest of this section is not further shaded for this option).

### 3697 3.1 General Concepts

#### 3698 3.1.1 Batch Client-Server Interaction

3699 Batch jobs are created and managed by batch servers. A batch client interacts with a batch server  
3700 to access batch services on behalf of the user. In order to use batch services, a user must have  
3701 access to a batch client.

3702 A batch server is a computational entity, such as a daemon process, that provides batch services.  
3703 Batch servers route, queue, modify, and execute batch jobs on behalf of batch clients.

3704 The batch utilities described in this volume of IEEE Std. 1003.1-200x (and listed in Table 3-1 (on  
3705 page 2314)) are clients of batch services; they allow users to perform actions on the job such as  
3706 creating, modifying, and deleting batch jobs from a shell command line. Although these batch  
3707 utilities may be said to accomplish certain services, they actually obtain services on behalf of a  
3708 user by means of requests to batch servers.

3709

**Table 3-1** Batch Utilities

3710

*qalter*    *qmove*    *qrls*    *qstat*

3711

*qdel*    *qmsg*    *qselect*    *qsub*

3712

*qhold*    *qrerun*    *qsig*

3713

Client-server interaction takes place by means of the batch requests defined in this chapter. Because direct access to batch jobs and queues is limited to batch servers, clients and servers of different implementations can interoperate, since dependencies on private structures for batch jobs and queues are limited to batch servers. Also, batch servers may be clients of other batch servers.

3714

3715

3716

3717

### 3718 **3.1.2 Batch Queues**

3719

Two types of batch queue are described: *routing queues* and *execution queues*. When a batch job is placed in a routing queue, it is a candidate for routing. A batch job is removed from routing queues under the following conditions:

3720

3721

3722

- The batch job has been routed to another queue.

3723

- The batch job has been deleted from the batch queue.

3724

- The batch job has been aborted.

3725

When a batch job is placed in an execution queue, it is a candidate for execution.

3726

A batch job is removed from an execution queue under the following conditions:

3727

- The batch job has been executed and exited.

3728

- The batch job has been aborted.

3729

- The batch job has been deleted from the batch queue.

3730

- The batch job has been moved to another queue.

3731

Access to a batch queue is limited to the batch server that manages the batch queue. Clients never access a batch queue or a batch job directly, either to read or write information; all client access to batch queues or jobs takes place through batch servers.

3732

3733

### 3734 **3.1.3 Batch Job Creation**

3735

When a batch server creates a batch job on behalf of a client, it assigns a batch job identifier to the job. A batch job identifier consists of both a sequence number that is unique among the sequence numbers issued by that server and the name of the server. Since the batch server name is unique within a name space, the job identifier is likewise unique within the name space.

3736

3737

3738

3739

The batch server that creates a batch job returns the batch server-assigned job identifier to the client that requested the job creation. If the batch server routes or moves the job to another server, it sends the job identifier with the job. Once assigned, the job identifier of a batch job never changes.

3740

3741

3742

**3743 3.1.4 Batch Job Tracking**

3744 Since a batch job may be moved after creation, the batch server name component of the job  
3745 identifier does not always indicate the location of the job. An implementation may provide a  
3746 batch job tracking mechanism, in which case the user generally does not need to know the  
3747 location of the job. However, an implementation is not required to provide a batch job tracking  
3748 mechanism, in which case the user must find routed jobs by probing the possible destinations.

**3749 3.1.5 Batch Job Routing**

3750 To route a batch job, a batch server either moves the job to some other queue that is managed by  
3751 the batch server, or requests that some other batch server accept the job.

3752 Each routing queue has one or more queues to which it can route batch jobs. The batch server  
3753 administrator creates routing queues.

3754 A batch server may route a batch job from a routing queue to another routing queue. Batch  
3755 servers shall prevent or otherwise handle cases of circular routing paths. As a deferred service, a  
3756 batch server routes jobs from the routing queues that it manages. The algorithm by which a  
3757 batch server selects a batch queue to which to route a batch job is implementation-defined.

3758 A batch job need not be eligible for routing to all the batch queues fed by the routing queue from  
3759 which it is routed. A batch server that has been asked to accept the job may reject the request if  
3760 the job requires resources that are unavailable to that batch server, or if the client is not  
3761 authorized to access the batch server.

3762 Batch servers may route high-priority jobs before low-priority jobs, but, on other than  
3763 overloaded systems, the effect may be imperceptible to the user. If all the batch servers fed by a  
3764 routing queue reject requests to accept the job for reasons that are permanent, the batch server  
3765 that manages the job aborts the job. If all or some rejections are temporary, the batch server  
3766 should try to route the job again at some later point.

3767 The conformance document for an implementation shall list the reasons for rejecting the routing  
3768 of a batch job. The conformance document shall indicate the reasons for which the routing  
3769 should be retried later and the reasons for which the job should be aborted.

**3770 3.1.6 Batch Job Execution**

3771 To execute a batch job is to create a session leader (a process) that runs the shell program  
3772 indicated by the *Shell\_Path* attribute of the job. The script is passed to the program as its  
3773 standard input. An implementation of the batch server may pass the script to the program by  
3774 other means. The implementation shall document the alternate means in the conformance  
3775 document. At the time a batch job begins execution, it is defined to enter the RUNNING state.  
3776 The primary program that is executed by a batch job is typically, though not necessarily, a shell  
3777 program.

3778 A batch server executes eligible jobs as a deferred service—no client request is necessary once  
3779 the batch job is created and eligible. However, the attributes of a batch job, such as the job hold  
3780 type, may render the job ineligible. A batch server scans the execution queues that it manages for  
3781 jobs that are eligible for execution. The algorithm by which the batch server selects eligible jobs  
3782 for execution is implementation-defined.

3783 As part of creating the process for the batch job, the batch server opens the standard output and  
3784 standard error streams of the session.

3785 The attributes of a batch job may indicate that the batch server that executes the job is to send  
3786 mail to a list of users at the time it begins execution of the job.

**3787 3.1.7 Batch Job Exit**

3788 When the session leader of an executing job terminates, the job exits. As part of exiting a batch  
3789 job, the batch server that manages the job shall remove the job from the batch queue in which it  
3790 resides. The server shall transfer output files of the job to a location described by the attributes of  
3791 the job.

3792 The attributes of a batch job may indicate that the batch server that manages the job should send  
3793 mail to a list of users at the time the job exits.

**3794 3.1.8 Batch Job Abort**

3795 A batch server aborts jobs for which a required deferred service cannot be performed. The  
3796 attributes of a batch job may indicate that the batch server that aborts the job shall send mail to a  
3797 list of users at the time it aborts the job.

**3798 3.1.9 Batch Authorization**

3799 In order to access batch services, a user must have execute access to a batch client. For example,  
3800 to use the command language interface defined in this section, the user must be able to execute  
3801 the programs that embody those utilities.

3802 Clients, such as the batch environment utilities (marked BE), access batch services by means of  
3803 requests to one or more batch servers. To acquire the services of any given batch server, the user  
3804 identifier under which the client runs must be authorized to use that batch server.

3805 The user with an associated user name that creates a batch job owns the job and can perform  
3806 actions such as read, modify, delete, and move.

3807 A user identifier of the same value at a different host need not be the same user. For example,  
3808 user name *smith* at host **alpha** may or may not represent the same person as user name *smith* at  
3809 host **beta**. Likewise, the same person may have access to different user names on different hosts.

3810 An implementation may optionally provide an authorization mechanism that permits one user  
3811 name to access jobs under another user name.

3812 A process on a client host may be authorized to run processes under multiple user names at a  
3813 batch server host. Where appropriate, the utilities defined in this volume of  
3814 IEEE Std. 1003.1-200x provide a means for a user to choose from among such user names when  
3815 creating or modifying a batch job.

**3816 3.1.10 Batch Administration**

3817 The processing of a batch job by a batch server is affected by the attributes of the job. The  
3818 processing of a batch job may also be affected by the attributes of the batch queue in which the  
3819 job resides and by the status of the batch server that manages the job.

3820 A batch administrator is a user that is authorized to modify all the attributes of queues and jobs  
3821 and to change the status of a batch server. A batch operator is a user that is authorized to modify  
3822 some, but not all, of the attributes of jobs and queues, and may change the status of the batch  
3823 server.



3824 **3.1.11 Batch Notification**

3825 Whereas batch servers are persistent entities, clients are often transient. For example, the *qsub*  
 3826 utility creates a batch job and exits. For this reason, batch servers notify users of batch job events  
 3827 by sending mail to the user that owns the job, or to other designated users.

3828 **3.2 Batch Services**

3829 The presence of Batch Environment option services is indicated by the configuration variable  
 3830 POSIX2\_PBS. A conforming batch server provides services as defined in this section.

3831 A batch server provides batch services in two ways:

- 3832 1. The batch server provides a service at the request of a client.
- 3833 2. The batch server provides a deferred service as a result of a change in conditions  
 3834 monitored by the batch server.

3835 If a batch server cannot complete a request, it rejects the request. If a batch server cannot  
 3836 complete a deferred service for a batch job, the batch server aborts the batch job. Table 3-2 is a  
 3837 summary of environment variables that shall be supported by an implementation of the batch  
 3838 server and utilities.

3839 **Table 3-2 Environment Variable Summary**

| Variable        | Description                                                                     |
|-----------------|---------------------------------------------------------------------------------|
| PBS_DPREFIX     | Defines the directive prefix (see <i>qsub</i> )                                 |
| PBS_ENVIRONMENT | Batch Job is batch or interactive (see Section 3.2.2.1 (on page 2319))          |
| PBS_JOBID       | The <i>job_identifier</i> attribute of job (see Section 3.2.3.8 (on page 2331)) |
| PBS_JOBNAME     | The <i>job_name</i> attribute of job (see Section 3.2.3.8 (on page 2331))       |
| PBS_O_HOME      | Defines the <i>HOME</i> of the batch client (see <i>qsub</i> )                  |
| PBS_O_HOST      | Defines the host name of the batch client (see <i>qsub</i> )                    |
| PBS_O_LANG      | Defines the <i>LANG</i> of the batch client (see <i>qsub</i> )                  |
| PBS_O_LOGNAME   | Defines the <i>LOGNAME</i> of the batch client (see <i>qsub</i> )               |
| PBS_O_MAIL      | Defines the <i>MAIL</i> of the batch client (see <i>qsub</i> )                  |
| PBS_O_PATH      | Defines the <i>PATH</i> of the batch client (see <i>qsub</i> )                  |
| PBS_O_QUEUE     | Defines the submit queue of the batch client (see <i>qsub</i> )                 |
| PBS_O_SHELL     | Defines the <i>SHELL</i> of the batch client (see <i>qsub</i> )                 |
| PBS_O_TZ        | Defines the <i>TZ</i> of the batch client (see <i>qsub</i> )                    |
| PBS_O_WORKDIR   | Defines the working directory of the batch client (see <i>qsub</i> )            |
| PBS_QUEUE       | Defines the initial execution queue (see Section 3.2.2.1 (on page 2319))        |

3859 **3.2.1 Batch Job States**

3860 A batch job is always in one of several states: QUEUED, RUNNING, HELD, WAITING,  
 3861 EXITING, or TRANSITING. The state of a batch job determines the types of requests that the  
 3862 batch server that manages the batch job can accept for the batch job. A batch server changes the  
 3863 state of a batch job either in response to service requests from clients or as a result of deferred  
 3864 services, such as job execution or job routing.

3865 A batch job that is in the QUEUED state resides in a queue but is still pending either execution or  
 3866 routing, depending on the queue type.

3867 A batch server that queues a batch job in a routing queue shall put the batch job in the QUEUED  
 3868 state. A batch server that puts a batch job in an execution queue, but has not yet executed the  
 3869 batch job, shall put the batch job in the QUEUED state. A batch job that resides in an execution  
 3870 queue and is executing is defined to be in the RUNNING state. While a batch job is in the  
 3871 RUNNING state, a session leader is associated with the batch job.

3872 A batch job that resides in an execution queue, but is ineligible to run because of a hold attribute,  
 3873 is defined to be in the HELD state.

3874 A batch job that is not held, but must wait until a future date and time before executing, is  
 3875 defined to be in the WAITING state.

3876 When the session leader associated with a running job exits, the batch job shall be placed in the  
 3877 EXITING state.

3878 A batch job for which the session leader has terminated is defined to be in the EXITING state,  
 3879 and the batch server that manages such a batch job cannot accept job modification requests that  
 3880 affect the batch job. While a batch job is in the EXITING state, the batch server that manages the  
 3881 batch job is staging output files and notifying clients of job completion. Once a batch job has  
 3882 exited, it no longer exists as an object managed by a batch server.

3883 A batch job that is being moved from a routing queue to another queue is defined to be in the  
 3884 TRANSITING state.

3885 When a batch job in a routing queue has been selected to be moved to a new destination, then  
 3886 the batch job is in either the QUEUED state or the TRANSITING state, depending on the batch  
 3887 server implementation.

3888 Batch jobs with either a *Execution\_Time* attribute value set in the future or a *Hold\_Types* attribute  
 3889 of value not equal to NO\_HOLD, or both, may be routed or held in the routing queue. An  
 3890 implementation shall document the treatment of jobs with the *Execution\_Time* or *Hold\_Types*  
 3891 attributes in a routing queue.

3892 When a batch job in a routing queue has not been selected to be moved to a new destination and  
 3893 the batch job has a *Hold\_Types* attribute value of other than NO\_HOLD, then the job should be in  
 3894 the HELD state.

3895 **Note:** The effect of a hold upon a batch job in a routing queue is implementation-defined. |  
 3896 The implementation should use the state that matches whether the batch job can |  
 3897 route with a hold or not.

3898 When a batch job in a routing queue has not been selected to be moved to a new destination and  
 3899 the batch job has:

- 3900 • A *Hold\_Types* attribute value of NO\_HOLD
- 3901 • An *Execution\_Time* attribute in the past

3902 then the batch job shall be in the QUEUED state.

3903 When a batch job in a routing queue has not been selected to be moved to a new destination and  
3904 the batch job has:

3905 • A *Hold\_Types* attribute value of NO\_HOLD

3906 • A *Execution\_Time* attribute in the future

3907 then the batch job may be in the WAITING state.

3908 **Note:** The effect of a future execution time upon a batch job in a routing queue is  
3909 implementation-defined. The implementation should use the state that matches  
3910 whether the batch job can route with a hold or not.

3911 Table 3-3 describes the next state of a batch job, given the current state of the batch job and the  
3912 type of request. Table 3-4 (on page 2321) describes the response of a batch server to a request,  
3913 given the current state of the batch job and the type of request.

### 3914 3.2.2 Deferred Batch Services

3915 This section describes the deferred services performed by batch servers: job execution, job  
3916 routing, job exit, job abort, and the rerunning of jobs after a restart.

#### 3917 3.2.2.1 Batch Job Execution

3918 To execute a batch job is to create a session leader (a process) that runs the shell program  
3919 indicated by the *Shell\_Path\_List* attribute of the batch job. The script is passed to the program as  
3920 its standard input. An implementation of the batch server may pass the script to the program by  
3921 other means. The implementation shall document the alternate means in the conformance  
3922 document. At the time a batch job begins execution, it is defined to enter the RUNNING state.

3923 **Table 3-3** Next State Table

| Request Type               | Current State |   |       |   |   |   |   |
|----------------------------|---------------|---|-------|---|---|---|---|
|                            | X             | Q | R     | H | W | E | T |
| Queue Batch Job Request    | Q             | e | e     | e | e | e | e |
| Modify Batch Job Request   | e             | Q | R     | H | W | e | T |
| Delete Batch Job Request   | e             | X | E     | X | X | E | X |
| Batch Job Message Request  | e             | Q | R     | H | W | E | T |
| Rerun Batch Job Request    | e             | e | Q     | e | e | e | e |
| Signal Batch Job Request   | e             | e | R     | H | W | e | e |
| Batch Job Status Request   | e             | Q | R     | H | W | E | T |
| Batch Queue Status Request | X             | Q | R     | H | W | E | T |
| Server Status Request      | X             | Q | R     | H | W | E | T |
| Select Batch Jobs Request  | X             | Q | R     | H | W | E | T |
| Move Batch Job Request     | e             | Q | R     | H | W | e | T |
| Hold Batch Job Request     | e             | H | R/H   | H | H | e | T |
| Release Batch Job Request  | Q             | R | Q/W/H | W | e | T |   |
| Server Shutdown Request    | X             | Q | Q     | H | W | E | T |
| Locate Batch Job Request   | e             | Q | R     | H | W | E | T |

3941 **Legend**

3942 X Nonexistent

3943 Q QUEUED

3944 R RUNNING

3945 H HELD

3946 W WAITING

3947 E EXITING

3948 T TRANSITING

3949 e Error

3950 A batch server that has an execution queue containing jobs is said to own the queue and manage  
3951 the batch jobs in that queue. A batch server that has been started shall execute the batch jobs in  
3952 the execution queues owned by the batch server. The batch server shall schedule for execution  
3953 those jobs in the execution queues that are in the QUEUED state. The algorithm for scheduling  
3954 jobs is implementation-defined.

3955 A batch server that executes a batch job shall create, in the environment of the session leader of  
3956 the batch job, an environment variable named *PBS\_ENVIRONMENT*, the value of which is the  
3957 string *PBS\_BATCH* encoded in the portable character set.

3958 A batch server that executes a batch job shall create, in the environment of the session leader of  
3959 the batch job, an environment variable named *PBS\_QUEUE*, the value of which is the name of  
3960 the execution queue of the batch job encoded in the portable character set.

3961 To rerun a batch job is to requeue a batch job that is currently executing and then kill the session  
3962 leader of the executing job by sending a SIGKILL prior to completion; see Section 3.2.3.11 (on  
3963 page 2333). A batch server that reruns a batch job shall append the standard output and  
3964 standard error files of the batch job to the corresponding files of the previous execution, if they  
3965 exist, with appropriate annotation. If either file does not exist, that file shall be created as in  
3966 normal execution.

3967

Table 3-4 Results/Output Table

3968

3969

3970

3971

3972

3973

3974

3975

3976

3977

3978

3979

3980

3981

3982

3983

3984

| Request Type               | Current State |   |   |   |   |   |   |
|----------------------------|---------------|---|---|---|---|---|---|
|                            | X             | Q | R | H | W | E | T |
| Queue Batch Job Request    | O             | e | e | e | e | e | e |
| Modify Batch Job Request   | e             | O | e | O | O | e | e |
| Delete Batch Job Request   | e             | O | O | O | O | e | O |
| Batch Job Message Request  | e             | e | O | e | e | e | e |
| Rerun Batch Job Request    | e             | e | O | e | e | e | e |
| Signal Batch Job Request   | e             | e | O | e | e | e | e |
| Batch Job Status Request   | e             | O | O | O | O | O | O |
| Batch Queue Status Request | O             | O | O | O | O | O | O |
| Server Status Request      | O             | O | O | O | O | O | O |
| Select Batch Job Request   | e             | O | O | O | O | O | O |
| Move Batch Job Request     | e             | O | O | O | O | e | e |
| Hold Batch Job Request     | e             | O | O | O | O | e | e |
| Release Batch Job Request  | e             | O | e | O | O | e | e |
| Server Shutdown Request    | O             | O | e | O | O | e | e |
| Locate Batch Job Request   | e             | O | O | O | O | O | O |

3985

**Legend**

3986

O OK

3987

e Error message

3988

The execution of a batch job by a batch server is controlled by job, queue, and server attributes, as defined in this section.

3989

3990

**Account\_Name Attribute**

3991

Batch accounting is an optional feature of batch servers. If a batch server implements accounting, the statements in this section apply and the configuration variable `POSIX2_PBS_ACCOUNTING` shall be set to 1.

3992

3993

3994

A batch server that executes a batch job shall charge the account named in the *Account\_Name* attribute of the batch job for resources consumed by the batch job.

3995

3996

If the *Account\_Name* attribute of the batch job is absent from the batch job attribute list or is altered while the batch job is in execution, the batch server action is implementation-defined.

3997

3998

**Checkpoint Attribute**

3999

Batch checkpointing is an optional feature of batch servers. If a batch server implements checkpointing, the statements in this section apply and the configuration variable `POSIX2_PBS_CHECKPOINT` shall be set to 1.

4000

4001

4002

There are two attributes associated with the checkpointing feature: *Checkpoint* and *Minimum\_Cpu\_Interval*. *Checkpoint* is a batch job attribute, while *Minimum\_Cpu\_Interval* is a queue attribute. An implementation that does not support checkpointing shall support the *Checkpoint* job attribute to the extent that the batch server shall maintain and pass this attribute to other servers.

4003

4004

4005

4006

4007

The behavior of a batch server that executes a batch job for which the value of the *Checkpoint* attribute is `CHECKPOINT_UNSPECIFIED` is implementation-defined. The implementation shall document the behavior of the batch server. A batch server that executes a batch job for which the

4008

4009

- 4010 value of the *Checkpoint* attribute is NO\_CHECKPOINT shall not checkpoint the batch job.
- 4011 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is  
4012 CHECKPOINT\_AT\_SHUTDOWN shall checkpoint the batch job only when the batch server  
4013 accepts a request to shut down during the time when the batch job is in the RUNNING state.
- 4014 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is  
4015 CHECKPOINT\_AT\_MIN\_CPU\_INTERVAL shall checkpoint the batch job at the interval  
4016 specified by the *Minimum\_Cpu\_Interval* attribute of the queue for which the batch job has been  
4017 selected. The *Minimum\_Cpu\_Interval* attribute shall be specified in units of CPU minutes.
- 4018 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is an  
4019 unsigned integer shall checkpoint the batch job at an interval that is the value of either the  
4020 *Checkpoint* attribute, or the *Minimum\_Cpu\_Interval* attribute of the queue for which the batch job  
4021 has been selected, whichever is greater. Both intervals shall be in units of CPU minutes. When  
4022 the *Minimum\_Cpu\_Interval* attribute is greater than the *Checkpoint* attribute, the batch job shall  
4023 write a warning message to the standard error stream of the batch job.
- 4024 **Error\_Path Attribute**
- 4025 The *Error\_Path* attribute of a running job cannot be changed by a *Modify Batch Job Request*. When  
4026 the *Join\_Path* attribute of the batch job is set to the value FALSE and the *Keep\_Files* attribute of  
4027 the batch job does not contain the value KEEP\_STD\_ERROR, a batch server that executes a batch  
4028 job shall perform one of the following actions:
- 4029 • Set the standard error stream of the session leader of the batch job to the path described by  
4030 the value of the *Error\_Path* attribute of the batch job.
  - 4031 • Buffer the standard error of the session leader of the batch job until completion of the batch  
4032 job, and when the batch job exits return the contents to the destination described by the value  
4033 of the *Error\_Path* attribute of the batch job. Where the batch server buffers standard error is  
4034 implementation-defined.
- 4035 Applications shall not rely on having access to the standard error of a batch job prior to the  
4036 completion of the batch job.
- 4037 When the *Error\_Path* attribute does not specify a host name, then the batch server shall retain the  
4038 standard error of the batch job on the host of execution.
- 4039 When the *Error\_Path* attribute does specify a host name and the *Keep\_Files* attribute does not  
4040 contain the value KEEP\_STD\_ERROR, then the final destination of the standard error of the  
4041 batch job shall be on the host whose host name is specified.
- 4042 If the path indicated by the value of the *Error\_Path* attribute of the batch job is a relative path, the  
4043 batch server shall expand the path relative to the home directory of the user on the host to which  
4044 the file is being returned.
- 4045 When the batch server buffers the standard error of the batch job and the file cannot be opened  
4046 for write upon completion of the batch job, then the server shall place the standard error in an  
4047 implementation-defined location and notify the user of the location via mail. It shall be possible  
4048 for the user to process this mail using the *mailx* utility.
- 4049 If a batch server that does not buffer the standard error cannot open the standard error path of  
4050 the batch job for write access, then the batch server shall abort the batch job.

**4051 Execution\_Time Attribute**

4052 A batch server shall not execute a batch job before the time represented by the value of the  
4053 *Execution\_Time* attribute of the batch job. The *Execution\_Time* attribute is defined in seconds since  
4054 the Epoch.

**4055 Hold\_Types Attribute**

4056 A batch server shall support the following hold types:

4057 **s** Can be set or released by a user with at least a privilege level of batch administrator  
4058 (SYSTEM).

4059 **o** Can be set or released by a user with at least a privilege level of batch operator  
4060 (OPERATOR).

4061 **u** Can be set or released by the user with at least a privilege level of user, where the user is  
4062 defined in the *Job\_Owner* attribute (USER).

4063 **n** Indicates that none of the *Hold\_Types* attributes are set (NO\_HOLD).

4064 An implementation may define other hold types. The conformance document for an  
4065 implementation shall describe any additional hold types, how they are specified, their internal  
4066 representation, their behavior, and how they affect the behavior of other utilities.

4067 The value of the *Hold\_Types* attribute shall be the union of the valid hold types (**ss**, **oo**, **uu**, and  
4068 any implementation-defined hold types), or **nn**.

4069 A batch server shall not execute a batch job if the *Hold\_Types* attribute of the batch job has a  
4070 value other than NO\_HOLD. If the *Hold\_Types* attribute of the batch job has a value other than  
4071 NO\_HOLD, the batch job shall be in the HELD state.

**4072 Job\_Owner Attribute**

4073 The *Job\_Owner* attribute consists of a pair of user name and host name values of the form:

4074 `username@hostname`

4075 A batch server that accepts a *Queue Batch Job Request* shall set the *Job\_Owner* attribute to a string  
4076 that is the *username@hostname* of the user who submitted the job.

**4077 Join\_Path Attribute**

4078 A batch server that executes a batch job for which the value of the *Join\_Path* attribute is TRUE  
4079 shall ignore the value of the *Error\_Path* attribute and merge the standard error of the batch job  
4080 with the standard output of the batch job.

**4081 Keep\_Files Attribute**

4082 A batch server that executes a batch job for which the value of the *Keep\_Files* attribute includes  
4083 the value KEEP\_STD\_OUTPUT shall retain the standard output of the batch job on the host  
4084 where execution occurs. The standard output shall be retained in the home directory of the user  
4085 under whose user ID the batch job is executed and the file name shall be the default file name for  
4086 the standard output as defined under the **-o** option of the *qsub* utility. The *Output\_Path* attribute  
4087 is not modified.

4088 A batch server that executes a batch job for which the value of the *Keep\_Files* attribute includes  
4089 the value KEEP\_STD\_ERROR shall retain the standard error of the batch job on the host where  
4090 execution occurs. The standard error shall be retained in the home directory of the user under  
4091 whose user ID the batch job is executed and the file name shall be the default file name for

4092 standard error as defined under the `-e` option of the `qsub` utility. The `Error_Path` attribute is not  
4093 modified.

4094 A batch server that executes a batch job for which the value of the `Keep_Files` attribute includes  
4095 values other than `KEEP_STD_OUTPUT` and `KEEP_STD_ERROR` shall retain these other files on  
4096 the host where execution occurs. These files shall be retained in the home directory of the user  
4097 under whose user identifier the batch job is executed and the file names shall be the default file  
4098 names for the files as defined in the conformance document for the implementation.

#### 4099 **Mail\_Points and Mail\_Users Attributes**

4100 A batch server that executes a batch job for which one of the values of the `Mail_Points` attribute is  
4101 the value `MAIL_AT_BEGINNING` shall send a mail message to each user account listed in the  
4102 `Mail_Users` attribute of the batch job.

4103 The mail message shall contain at least the batch job identifier, queue, and server at which the  
4104 batch job currently resides, and the `Job_Owner` attribute.

#### 4105 **Output\_Path Attribute**

4106 The `Output_Path` attribute of a running job cannot be changed by a *Modify Batch Job Request*.  
4107 When the `Keep_Files` attribute of the batch job does not contain the value `KEEP_STD_OUTPUT`, a  
4108 batch server that executes a batch job shall either:

4109 • Set the standard output stream of the session leader of the batch job to the destination  
4110 described by the value of the `Output_Path` attribute of the batch job.

4111 or:

4112 • Buffer the standard output of the session leader of the batch job until completion of the batch  
4113 job, and when the batch job exits return the contents to the destination described by the value  
4114 of the `Output_Path` attribute of the batch job.

4115 When the `Output_Path` attribute does not specify a host name, then the batch server shall retain  
4116 the standard output of the batch job on the host of execution.

4117 When the `Keep_Files` attribute does not contain the value `KEEP_STD_OUTPUT` and the  
4118 `Output_Path` attribute does specify a host name, then the final destination of the standard output  
4119 of the batch job shall be on the host specified.

4120 If the path specified in the `Output_Path` attribute of the batch job is a relative path, the batch  
4121 server shall expand the path relative to the home directory of the user on the host to which the  
4122 file is being returned.

4123 Whether or not the batch server buffers the standard output of the batch job until completion of  
4124 the batch job is implementation-defined. Applications shall not rely on having access to the  
4125 standard output of a batch job prior to the completion of the batch job.

4126 When the batch server does buffer the standard output of the batch job and the file cannot be  
4127 opened for write upon completion of the batch job, then the batch server shall place the standard  
4128 output in an implementation-defined location and notify the user of the location via mail. It shall  
4129 be possible for the user to process this mail using the `mailx` utility.

4130 If a batch server that does not buffer the standard output cannot open the standard output path  
4131 of the batch job for write access, then the batch server shall abort the batch job.



**4132 Priority Attribute**

4133 A batch server implementation may choose to preferentially execute a batch job based on the  
4134 *Priority* attribute. The interpretation of the batch job *Priority* attribute by a batch server is  
4135 implementation-defined. If an implementation uses the *Priority* attribute, it shall interpret larger  
4136 values of the *Priority* attribute to mean the batch job shall be preferentially selected for execution.

**4137 Rerunable Attribute**

4138 A batch job that began execution but did not complete, because the batch server either shut  
4139 down or terminated abnormally, shall be requeued if the *Rerunable* attribute of the batch job has  
4140 the value TRUE.

4141 If a batch job, which was requeued after beginning execution but prior to completion, has a valid  
4142 checkpoint file and the batch server supports checkpointing, then the batch job shall be restarted  
4143 from the last valid checkpoint.

4144 If the batch job cannot be restarted from a checkpoint, then when a batch job has a *Rerunable*  
4145 attribute value of TRUE and was requeued after beginning execution but prior to completion,  
4146 the batch server shall place the batch job into execution at the beginning of the job.

4147 When a batch job has a *Rerunable* attribute value other than TRUE and was requeued after  
4148 beginning execution but prior to completion, and the batch job cannot be restarted from a  
4149 checkpoint, then the batch server shall abort the batch job.

**4150 Resource\_List Attribute**

4151 A batch server that executes a batch job shall establish the resource limits of the session leader of  
4152 the batch job according to the values of the *Resource\_List* attribute of the batch job. Resource  
4153 limits shall be enforced by an implementation-defined method.

**4154 Shell\_Path\_List Attribute**

4155 The *Shell\_Path\_List* job attribute consists of a list of pairs of path name and host name values.  
4156 The host name component can be omitted, in which case the path name serves as the default  
4157 path name when a batch server cannot find the name of the host on which it is running in the  
4158 list.

4159 A batch server that executes a batch job shall select, from the value of the *Shell\_Path\_List*  
4160 attribute of the batch job, a path name where the shell to execute the batch job shall be found.  
4161 The batch server shall select the path name, in order of preference, according to the following  
4162 methods:

- 4163 • Select the path name that contains the name of the host on which the batch server is running.
- 4164 • Select the path name for which the host name has been omitted.
- 4165 • Select the path name for the login shell of the user under which the batch job is to execute.

4166 If the shell path value selected is an invalid path name, the batch server shall abort the batch job.

4167 If the value of the selected path name from the *Shell\_Path\_List* attribute of the batch job  
4168 represents a partial path, the batch server shall expand the path relative to a path that is  
4169 implementation-defined.

4170 The batch server that executes the batch job shall execute the program that was selected from the  
4171 *Shell\_Path\_List* attribute of the batch job. The batch server shall pass the path to the script of the  
4172 batch job as the first argument to the shell program.

4173 **User\_List Attribute**

4174 The *User\_List* job attribute consists of a list of pairs of user name and host name values. The host  
4175 name component can be omitted, in which case the user name serves as a default when a batch  
4176 server cannot find the name of the host on which it is running in the list.

4177 A batch server that executes a batch job shall select, from the value of the *User\_List* attribute of  
4178 the batch job, a user name under which to create the session leader. The server shall select the  
4179 user name, in order of preference, according to the following methods:

- 4180 • Select the user name of a value that contains the name of the host on which the batch server  
4181 executes.
- 4182 • Select the user name of a value for which the host name has been omitted.
- 4183 • Select the user name from the *Job\_Owner* attribute of the batch job.

4184 **Variable\_List Attribute**

4185 A batch server that executes a batch job shall create, in the environment of the session leader of  
4186 the batch job, each environment variable listed in the *Variable\_List* attribute of the batch job, and  
4187 set the value of each such environment variable to that of the corresponding variable in the  
4188 variable list.

4189 **3.2.2.2 Batch Job Routing**

4190 To route a batch job is to select a queue from a list and move the batch job to that queue.

4191 A batch server that has routing queues, which have been started, shall route the jobs in the  
4192 routing queues owned by the batch server. A batch server is allowed to delay the routing of a  
4193 batch job. The algorithm for selecting a batch job and the queue to which it will be routed is  
4194 implementation-defined.

4195 When a routing queue has multiple possible destinations specified, then the precedence of the  
4196 destination is implementation-defined.

4197 A batch server that routes a batch job to a queue at another server shall move the batch job into  
4198 the target queue with a *Queue Batch Job Request*.

4199 If the target server rejects the *Queue Batch Job Request*, the routing server shall retry routing the  
4200 batch job or abort the batch job. A batch server that retries failed routings shall provide a means  
4201 for the batch administrator to specify the number of retries and the minimum period of time  
4202 between retries. The means by which an administrator specifies the number of retries and the  
4203 delay between retries is implementation-defined. When the number of retries specified by the  
4204 batch administrator has been exhausted, the batch server shall abort the batch job and perform  
4205 the functions of *Batch Job Exit*; see Section 3.2.2.3.

4206 **3.2.2.3 Batch Job Exit**

4207 For each job in the EXITING state, the batch server that exited the batch job shall perform the  
4208 following deferred services in the order specified:

- 4209 1. If buffering standard error, move that file into the location specified by the *Error\_Path*  
4210 attribute of the batch job.
- 4211 2. If buffering standard output, move that file into the location specified by the *Output\_Path*  
4212 attribute of the batch job.
- 4213 3. If the *Mail\_Points* attribute of the batch job includes MAIL\_AT\_EXIT, send mail to the users  
4214 listed in the *Mail\_Users* attribute of the batch job. The mail message shall contain at least

4215 the batch job identifier, queue, and server at which the batch job currently resides, and the  
4216 *Job\_Owner* attribute.

4217 4. Remove the batch job from the queue.

4218 If a batch server that buffers the standard error output cannot return the standard error file to  
4219 the standard error path at the time the batch job exits, the batch server shall do one of the  
4220 following:

- 4221 • Mail the standard error file to the batch job owner.
- 4222 • Save the standard error file and mail the location and name of the file where the standard  
4223 error is stored to the batch job owner.
- 4224 • Save the standard error file and notify the user by other means, in which case the  
4225 conformance document for the implementation shall document the method of notification.

4226 If a batch server that buffers the standard output cannot return the standard output file to the  
4227 standard output path at the time the batch job exits, the batch server shall do one of the  
4228 following:

- 4229 • Mail the standard output file to the batch job owner.
- 4230 • Save the standard output file and mail the location and name of the file where the standard  
4231 output is stored to the batch job owner.
- 4232 • Save the standard output file and notify the user by other means, in which case the  
4233 conformance document for the implementation shall document the method of notification.

4234 At the conclusion of job exit processing, the batch job is no longer managed by a batch server.

#### 4235 3.2.2.4 *Batch Server Restart*

4236 A batch server that has been either shutdown or terminated abnormally, and has returned to  
4237 operation, is said to have *restarted*.

4238 Upon restarting, a batch server shall requeue those jobs managed by the batch server that were  
4239 in the RUNNING state at the time the batch server shut down and for which the *Rerunable*  
4240 attribute of the batch job has the value TRUE.

4241 Queues are defined to be non-volatile. A batch server shall store the content of queues that it  
4242 controls in such a way that server and system shutdowns do not erase the content of the queues.

#### 4243 3.2.2.5 *Batch Job Abort*

4244 A batch server that cannot perform a deferred service for a batch job shall abort the batch job.

4245 A batch server that aborts a batch job shall perform the following services:

- 4246 • Delete the batch job from the queue in which it resides.
- 4247 • If the *Mail\_Points* attribute of the batch job includes the value MAIL\_AT\_ABORT, send mail  
4248 to the users listed in the value of the *Mail\_Users* attribute of the job. The mail message shall  
4249 contain at least the batch job identifier, queue, and server at which the batch job currently  
4250 resides, the *Job\_Owner* attribute, and the reason for the abort.
- 4251 • If the batch job was in the RUNNING state, terminate the session leader of the executing job  
4252 by sending the session leader a SIGKILL, place the batch job in the EXITING state, and  
4253 perform the services of *Batch Job Exit*.

4254 **3.2.3 Requested Batch Services**

4255 This section describes the services provided by batch servers in response to requests from  
 4256 clients. Table 3-5 summarizes the current set of batch service requests and for each gives its type  
 4257 (deferred or not) and whether it is an optional function.

4258 **Table 3-5 Batch Services Summary**

| Batch Service                     | Deferred | Optional |
|-----------------------------------|----------|----------|
| <i>Batch Job Execution</i>        | Yes      | No       |
| <i>Batch Job Routing</i>          | Yes      | No       |
| <i>Batch Job Exit</i>             | Yes      | No       |
| <i>Batch Server Restart</i>       | Yes      | No       |
| <i>Batch Job Abort</i>            | Yes      | No       |
| <i>Delete Batch Job Request</i>   | No       | No       |
| <i>Hold Batch Job Request</i>     | No       | No       |
| <i>Batch Job Message Request</i>  | No       | Yes      |
| <i>Batch Job Status Request</i>   | No       | No       |
| <i>Locate Batch Job Request</i>   | No       | Yes      |
| <i>Modify Batch Job Request</i>   | No       | No       |
| <i>Move Batch Job Request</i>     | No       | No       |
| <i>Queue Batch Job Request</i>    | No       | No       |
| <i>Batch Queue Status Request</i> | No       | No       |
| <i>Release Batch Job Request</i>  | No       | No       |
| <i>Rerun Batch Job Request</i>    | No       | No       |
| <i>Select Batch Jobs Request</i>  | No       | No       |
| <i>Server Shutdown Request</i>    | No       | No       |
| <i>Server Status Request</i>      | No       | No       |
| <i>Signal Batch Job Request</i>   | No       | No       |
| <i>Track Batch Job Request</i>    | No       | Yes      |

4281 If a request is rejected because the batch client is not authorized to perform the action, the batch  
 4282 server shall return the same status as when the batch job does not exist.

4283 **3.2.3.1 Delete Batch Job Request**

4284 A batch job is defined to have been deleted when it has been removed from the queue in which it  
 4285 resides and not instantiated in another queue. A client requests that the server that manages a  
 4286 batch job delete the batch job. Such a request is called a *Delete Batch Job Request*.

4287 A batch server shall reject a *Delete Batch Job Request* if any of the following statements are true:

- 4288 • The user of the batch client is not authorized to delete the designated job.
- 4289 • The designated job is not managed by the batch server.
- 4290 • The designated job is in a state inconsistent with the delete request.

4291 A batch server may reject a *Delete Batch Job Request* for other reasons. The conformance document  
 4292 for an implementation shall describe the reasons for which a *Delete Batch Job Request* may be  
 4293 rejected. The conformance document for an implementation shall describe the method used to  
 4294 determine whether the user of a client is authorized to perform the requested action.

4295 A batch server requested to delete a batch job shall delete the batch job if the batch job exists and  
 4296 is not in the EXITING state.

4297 A batch server that deletes a batch job in the RUNNING state shall send a SIGKILL signal to the  
4298 session leader of the batch job. A batch server may send additional signals to the session leader  
4299 of the job prior to sending the SIGKILL signal. The conformance document for such a batch  
4300 server shall document the signals that are sent to the session leader.

4301 A batch server that deletes a batch job in the RUNNING state shall place the batch job in the  
4302 EXITING state after it has killed the session leader of the batch job and shall perform the services  
4303 of batch job exit.

#### 4304 3.2.3.2 *Hold Batch Job Request*

4305 A batch client can request that the batch server add one or more holds to a batch job. Such a  
4306 request is called a *Hold Batch Job Request*.

4307 A batch server shall reject a *Hold Batch Job Request* if any of the following statements are true:

- 4308 • The batch server does not support one or more of the requested holds to be added to the  
4309 batch job.
- 4310 • The user of the batch client is not authorized to add one or more of the requested holds to the  
4311 batch job.
- 4312 • The batch server does not manage the specified job.
- 4313 • The designated job is in the EXITING state.

4314 A batch server may reject a *Hold Batch Job Request* for other reasons. The conformance document  
4315 for an implementation shall document the reasons for which a *Hold Batch Job Request* may be  
4316 rejected. The conformance document for an implementation shall describe the method used to  
4317 determine whether the user of a client is authorized to perform the requested action.

4318 A batch server that accepts a *Hold Batch Job Request* for a batch job in the RUNNING state shall  
4319 place a hold on the batch job. The conformance document shall describe what effect, if any, the  
4320 hold will have on a batch job in the RUNNING state.

4321 A batch server that accepts a *Hold Batch Job Request* shall add each type of hold listed in the *Hold*  
4322 *Batch Job Request*, that is not already present, to the value of the *Hold\_Types* attribute of the batch  
4323 job.

#### 4324 3.2.3.3 *Batch Job Message Request*

4325 *Batch Job Message Request* is an optional feature of batch servers. If an implementation supports  
4326 *Batch Job Message Request*, the statements in this section apply and the configuration variable  
4327 POSIX2\_PBS\_MESSAGE shall be set to 1.

4328 A batch client can request that a batch server write a message into certain output files of a batch  
4329 job. Such a request is called a *Batch Job Message Request*.

4330 A batch server shall reject a *Batch Job Message Request* if any of the following statements are true:

- 4331 • The batch server does not support sending messages to jobs.
- 4332 • The user of the batch client is not authorized to post a message to the designated job.
- 4333 • The designated job does not exist on the batch server.
- 4334 • The designated job is not in the RUNNING state.

4335 A batch server may reject a *Batch Job Message Request* for other reasons. The conformance  
4336 document for an implementation shall describe the reasons for which a *Batch Job Message Request*  
4337 may be rejected. The conformance document for an implementation shall describe the method

4338 used to determine whether the user of a client is authorized to perform the requested action.

4339 A batch server that accepts a *Batch Job Message Request* shall write the message sent by the batch  
4340 client into the files indicated by the batch client.

#### 4341 3.2.3.4 *Batch Job Status Request*

4342 A batch client can request that a batch server respond with the status and attributes of a batch  
4343 job. Such a request is called a *Batch Job Status Request*.

4344 A batch server shall reject a *Batch Job Status Request* if any of the following statements are true:

- 4345 • The user of the batch client is not authorized to query the status of the designated job.
- 4346 • The designated job is not managed by the batch server.

4347 A batch server may reject a *Batch Job Status Request* for other reasons. The conformance  
4348 document for an implementation shall describe the reasons for which a *Batch Job Status Request*  
4349 may be rejected. The conformance document for an implementation shall describe the method  
4350 used to determine whether the user of a client is authorized to perform the requested action.

4351 A batch server that accepts a *Batch Job Status Request* shall return a *Batch Job Status Message* to the  
4352 batch client.

4353 A batch server may return other information in response to a *Batch Job Status Request*.

#### 4354 3.2.3.5 *Locate Batch Job Request*

4355 *Locate Batch Job Request* is an optional feature of batch servers. If an implementation supports  
4356 *Locate Batch Job Request*, the statements in this section apply and the configuration variable  
4357 POSIX2\_PBS\_LOCATE shall be set to 1.

4358 A batch client can ask a batch server to respond with the location of a batch job that was created  
4359 by the batch server. Such a request is called a *Locate Batch Job Request*.

4360 A batch server that accepts a *Locate Batch Job Request* shall return a *Batch Job Location Message* to  
4361 the batch client.

4362 A batch server may reject a *Locate Batch Job Request* for a batch job that was not created by that  
4363 server.

4364 A batch server may reject a *Locate Batch Job Request* for a batch job that is no longer managed by  
4365 that server; that is, for a batch job that is not in a queue owned by that server.

4366 A batch server may reject a *Locate Batch Job Request* for other reasons. The conformance  
4367 document for an implementation shall document the reasons for which a *Locate Batch Job Request*  
4368 may be rejected.

#### 4369 3.2.3.6 *Modify Batch Job Request*

4370 Batch clients modify (alter) the attributes of a batch job by making a request to the server that  
4371 manages the batch job. Such a request is called a *Modify Batch Job Request*.

4372 A batch server shall reject a *Modify Batch Job Request* if any of the following statements are true:

- 4373 • The user of the batch client is not authorized to make the requested modification to the batch  
4374 job.
- 4375 • The designated job is not managed by the batch server.
- 4376 • The requested modification is inconsistent with the state of the batch job.

- 4377           • An unrecognized resource is requested for a batch job in an execution queue.
- 4378           A batch server may reject a *Modify Batch Job Request* for other reasons. The conformance  
4379           document for an implementation shall describe the reasons for which a *Modify Batch Job Request*  
4380           may be rejected. The conformance document for an implementation shall describe the method  
4381           used to determine whether the user of a client is authorized to perform the requested action.
- 4382           A batch server that accepts a *Modify Batch Job Request* shall modify all the specified attributes of  
4383           the batch job. A batch server that rejects a *Modify Batch Job Request* shall modify none of the  
4384           attributes of the batch job.
- 4385           If the servicing by a batch server of an otherwise valid request would result in no change, then  
4386           the batch server shall indicate successful completion of the request.
- 4387   3.2.3.7   *Move Batch Job Request*
- 4388           A batch client can request that a batch server move a batch job to another destination. Such a  
4389           request is called a *Move Batch Job Request*.
- 4390           A batch server shall reject a *Move Batch Job Request* if any of the following statements are true:
- 4391           • The user of the batch client is not authorized to remove the designated job from the queue in  
4392            which the batch job resides.
- 4393           • The user of the batch client is not authorized to move the designated job to the destination.
- 4394           • The designated job is not managed by the batch server.
- 4395           • The designated job is in the EXITING state.
- 4396           • The destination is inaccessible.
- 4397           A batch server can reject a *Move Batch Job Request* for other reasons. The conformance document  
4398           for an implementation shall describe the reasons for which a *Move Batch Job Request* may be  
4399           rejected. The conformance document for an implementation shall describe the method used to  
4400           determine whether the user of a client is authorized to perform the requested action.
- 4401           A batch server that accepts a *Move Batch Job Request* shall perform the following services:
- 4402           • Queue the designated job at the destination.
- 4403           • Remove the designated job from the queue in which the batch job resides.
- 4404           If the destination resides on another batch server, the batch server shall queue the batch job at  
4405           the destination by sending a *Queue Batch Job Request* to the other server. If the *Queue Batch Job*  
4406           *Request* fails, the batch server shall reject the *Move Batch Job Request*. If the *Queue Batch Job Request*  
4407           succeeds, the batch server shall remove the batch job from its queue.
- 4408           The batch server shall not modify any attributes of the batch job.
- 4409   3.2.3.8   *Queue Batch Job Request*
- 4410           A batch queue is controlled by one and only one batch server. A batch server is said to own the  
4411           queues that it controls. Batch clients make requests of batch servers to have jobs queued. Such a  
4412           request is called a *Queue Batch Job Request*.
- 4413           A batch server requested to queue a batch job for which the queue is unspecified shall select a  
4414           queue for the batch job. Such a queue is called the *default queue* of the batch server. The  
4415           conformance document for the implementation shall document the means by which the batch  
4416           server determines the default queue. The implementation shall provide the means for a batch  
4417           administrator to specify the default queue. The queue, whether specified or defaulted, is called

4418 the *target queue*.

4419 A batch server shall reject a *Queue Batch Job Request* if any of the following statements are true:

- 4420 • The client is not authorized to create a batch job in the target queue.
- 4421 • The request specifies a queue that does not exist on the batch server.
- 4422 • The target queue is an execution queue and the batch server cannot satisfy a resource  
4423 requirement of the batch job.
- 4424 • The target queue is an execution queue and an unrecognized resource is requested.
- 4425 • The target queue is an execution queue, the batch server does not support checkpointing, and  
4426 the value of the *Checkpoint* attribute of the batch job is not NO\_CHECKPOINT.
- 4427 • The job requires access to a user identifier that the batch client is not authorized to access.

4428 A batch server may reject a *Queue Batch Job Request* for other reasons. The conformance  
4429 document for an implementation shall document the reasons for which a *Queue Batch Job Request*  
4430 may be rejected.

4431 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the  
4432 PBS\_O\_QUEUE value is missing from the value of the *Variable\_List* attribute of the batch job  
4433 shall add that variable to the list and set the value to the name of the target queue. Once set, no  
4434 server shall change the value of PBS\_O\_QUEUE, even if the batch job is moved to another  
4435 queue.

4436 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the PBS\_JOBID  
4437 value is missing from the value of the *Variable\_List* attribute shall add that variable to the list and  
4438 set the value to the batch job identifier assigned by the server in the format:

4439       sequence\_number.server

4440 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the  
4441 PBS\_JOBNAME value is missing from the value of the *Variable\_List* attribute of the batch job  
4442 shall add that variable to the list and set the value to the *Job\_Name* attribute of the batch job.

### 4443 3.2.3.9 *Batch Queue Status Request*

4444 A batch client can request that a batch server respond with the status and attributes of a queue.  
4445 Such a request is called a *Batch Queue Status Request*.

4446 A batch server shall reject a *Batch Queue Status Request* if any of the following statements are true:

- 4447 • The user of the batch client is not authorized to query the status of the designated queue.
- 4448 • The designated queue does not exist on the batch server.

4449 A batch server may reject a *Batch Queue Status Request* for other reasons. The conformance  
4450 document for an implementation shall describe the reasons for which a *Batch Queue Status*  
4451 *Request* is rejected. The conformance document for an implementation shall describe the method  
4452 used to determine whether the user of a client is authorized to perform the requested action.

4453 A batch server that accepts a *Batch Queue Status Request* shall return a *Batch Queue Status Reply* to  
4454 the batch client.



4455 3.2.3.10 *Release Batch Job Request*

4456 A batch client can request that server remove one or more holds from a batch job. Such a request  
4457 is called a *Release Batch Job Request*.

4458 A batch server shall reject a *Release Batch Job Request* if any of the following statements are true:

- 4459 • The user of the batch client is not authorized to remove one or more of the requested holds  
4460 from the batch job.
- 4461 • The batch server does not manage the specified job.

4462 A batch server may reject a *Release Batch Job Request* for other reasons. The conformance  
4463 document for an implementation shall document the reasons for which a *Release Batch Job*  
4464 *Request* may be rejected. The conformance document for an implementation shall describe the  
4465 method used to determine whether the user of a client is authorized to perform the requested  
4466 action.

4467 A batch server that accepts a *Release Batch Job Request* shall remove each type of hold listed in the  
4468 *Release Batch Job Request*, that is present, from the value of the *Hold\_Types* attribute of the batch  
4469 job.

4470 3.2.3.11 *Rerun Batch Job Request*

4471 To rerun a batch job is to kill the session leader of the batch job and leave the batch job eligible  
4472 for re-execution. A batch client can request that a batch server rerun a batch job. Such a request is  
4473 called *Rerun Batch Job Request*.

4474 A batch server shall reject a *Rerun Batch Job Request* if any of the following statements are true:

- 4475 • The user of the batch client is not authorized to rerun the designated job.
- 4476 • The *Rerunable* attribute of the designated job has the value FALSE.
- 4477 • The designated job is not in the RUNNING state.
- 4478 • The batch server does not manage the designated job.

4479 A batch server may reject a *Rerun Batch Job Request* for other reasons. The conformance document  
4480 for an implementation shall describe the reasons for which a *Rerun Batch Job Request* may be  
4481 rejected. The conformance document for an implementation shall describe the method used to  
4482 determine whether the user of a client is authorized to perform the requested action.

4483 A batch server that rejects a *Rerun Batch Job Request* shall in no way modify the execution of the  
4484 batch job.

4485 A batch server that accepts a request to rerun a batch job shall perform the following services:

- 4486 • Requeue the batch job in the execution queue in which it was executing.
- 4487 • Send a SIGKILL signal to the process group of the session leader of the batch job.

4488 An implementation may indicate to the batch job owner that the batch job has been rerun. The  
4489 conformance document for an implementation shall state whether the batch job owner is  
4490 notified that a batch job is rerun, and if so, shall describe the means used.

4491 A batch server that reruns a batch job may send other signals to the session leader of the batch  
4492 job prior to sending the SIGKILL signal. The conformance document for an implementation  
4493 shall describe any other signals that may be sent.

4494 A batch server may preferentially select a rerun job for execution. The conformance document  
4495 for an implementation shall state whether rerun jobs shall be selected for execution before other

4496 jobs.

4497 **3.2.3.12** *Select Batch Jobs Request*

4498 A batch client can request from a batch server a list of jobs managed by that server that match a  
4499 list of selection criteria. Such a request is called a *Select Batch Jobs Request*. All the batch jobs  
4500 managed by the batch server that receives the request are candidates for selection.

4501 A batch server that accepts a *Select Batch Jobs Request* shall return a list of zero or more job  
4502 identifiers that correspond to jobs that meet the selection criteria.

4503 If the batch client is not authorized to query the status of a batch job, the batch server shall not  
4504 select the batch job.

4505 **3.2.3.13** *Server Shutdown Request*

4506 A batch server is defined to have shut down when it does not respond to requests from clients  
4507 and does not perform deferred services for jobs. A batch client can request that a batch server  
4508 shut down. Such a request is called a *Server Shutdown Request*.

4509 A batch server shall reject a *Server Shutdown Request* from a client that is not authorized to shut  
4510 down the batch server. The conformance document for an implementation shall describe the  
4511 method used to determine whether the user of a client is authorized to perform the requested  
4512 action.

4513 A batch server may reject a *Server Shutdown Request* for other reasons. The conformance  
4514 document for an implementation shall document the reasons for which a *Server Shutdown*  
4515 *Request* may be rejected.

4516 At server shutdown, a batch server shall do, in order of preference, one of the following:

- 4517 • If checkpointing is implemented and the batch job is checkpointable, then checkpoint the  
4518 batch job and requeue it.
- 4519 • If the batch job is rerunnable, then requeue the batch job to be rerun (restarted from the  
4520 beginning).
- 4521 • Abort the batch job.

4522 **3.2.3.14** *Server Status Request*

4523 A batch client can request that a batch server respond with the status and attributes of the batch  
4524 server. Such a request is called a *Server Status Request*.

4525 A batch server shall reject a *Server Status Request* if the following statement is true:

- 4526 • The user of the batch client is not authorized to query the status of the designated server.

4527 A batch server may reject a *Server Status Request* for other reasons. The conformance document  
4528 for an implementation shall describe the reasons for which a *Server Status Request* may be  
4529 rejected. The conformance document for an implementation shall describe the method used to  
4530 determine whether the user of a client is authorized to perform the requested action.

4531 A batch server that accepts a *Server Status Request* shall return a *Server Status Reply* to the batch  
4532 client.

4533 3.2.3.15 *Signal Batch Job Request*

4534 A batch client can request that a batch server signal the session leader of a batch job. Such a  
4535 request is called a *Signal Batch Job Request*.

4536 A batch server shall reject a *Signal Batch Job Request* if any of the following statements are true:

- 4537 • The user of the batch client is not authorized to signal the batch job.
- 4538 • The job is not in the RUNNING state.
- 4539 • The batch server does not manage the designated job.
- 4540 • The requested signal is not supported by the implementation.

4541 A batch server may reject a *Signal Batch Job Request* for other reasons. The conformance  
4542 document for an implementation shall describe the reasons for which a *Signal Batch Job Request*  
4543 may be rejected. The conformance document for an implementation shall describe the method  
4544 used to determine whether the user of a client is authorized to perform the requested action.

4545 A batch server that accepts a request to signal a batch job shall send the signal requested by the  
4546 batch client to the process group of the session leader of the batch job.

4547 3.2.3.16 *Track Batch Job Request*

4548 *Track Batch Job Request* is an optional feature of batch servers. If an implementation supports  
4549 *Track Batch Job Request*, the statements in this section apply and the configuration variable  
4550 POSIX2\_PBS\_TRACK shall be set to 1.

4551 *Track Batch Job Request* provides a method for tracking the current location of a batch job. Clients  
4552 may use the tracking information to determine the batch server that should receive a batch  
4553 server request.

4554 If *Track Batch Job Request* is supported by a batch server, then when the batch server queues a  
4555 batch job as a result of a *Queue Batch Job Request*, and the batch server is not the batch server that  
4556 created the batch job, the batch server shall send a *Track Batch Job Request* to the batch server that  
4557 created the job.

4558 If *Track Batch Job Request* is supported by a batch server, then the *Track Batch Job Request* may also  
4559 be sent to other servers as a backup to the primary server. The method by which backup servers  
4560 are specified is implementation-defined.

4561 If *Track Batch Job Request* is supported by a batch server that receives a *Track Batch Job Request*,  
4562 then the batch server shall record the current location of the batch job as contained in the  
4563 request.

### 4564 3.3 Common Behavior for Batch Environment Utilities

#### 4565 3.3.1 Batch Job Identifier

4566 A utility shall recognize *job\_identifiers* of the format:

4567 [sequence\_number][.server\_name][@server]

4568 where:

4569 *sequence\_number* An integer that, when combined with *server\_name*, provides a batch job  
4570 identifier that is unique within the batch system.

4571 *server\_name* The name of the batch server to which the batch job was originally submitted.

4572 *server* The name of the batch server that is currently managing the batch job.

4573 If the application omits the batch *server\_name* portion of a batch job identifier, a utility shall use  
4574 the name of a default batch server.

4575 If the application omits the batch *server* portion of a batch job identifier, a utility shall use:

- 4576 • The batch server indicated by *server\_name*, if present.
- 4577 • The name of the default batch server.
- 4578 • The name of the batch server that is currently managing the batch job.

4579 If only *@server* is specified, then the status of all jobs owned by the user on the requested server  
4580 is listed.

4581 The means by which a utility determines the default batch server is implementation-defined.

4582 If the application presents the batch *server* portion of a batch job identifier to a utility, the utility  
4583 shall send the request to the specified server.

4584 A strictly conforming application shall use the syntax described for the job identifier. Whenever  
4585 a batch job identifier is specified whose syntax is not recognized by an implementation, then a  
4586 message for each error that occurs shall be written to standard error and the utility shall exit  
4587 with an exit status greater than zero.

4588 When a batch job identifier is supplied as an argument to a batch utility and the *server\_name*  
4589 portion of the batch job identifier is omitted, then the utility shall use the name of the default  
4590 batch server.

4591 When a batch job identifier is supplied as an argument to a batch utility and the batch *server*  
4592 portion of the batch job identifier is omitted, then the utility shall use either:

- 4593 • The name of the default batch server

4594 or:

- 4595 • The name of the batch server that is currently managing the batch job

4596 When a batch job identifier is supplied as an argument to a batch utility and the batch *server*  
4597 portion of the batch job identifier is specified, then the utility shall send the required *Batch Server*  
4598 *Request* to the specified server.

4599 **3.3.2 Destination**4600 The utility shall recognize a *destination* of the format:

4601 [queue][@server]

4602 where:

4603 *queue* The name of a valid execution or routing queue at the batch server denoted by  
 4604 @*server*, defined as a string of up to 15 alphanumeric characters in the portable  
 4605 character set (see the Base Definitions volume of IEEE Std. 1003.1-200x,  
 4606 Section 6.1, Portable Character Set) where the first character is alphabetic.

4607 *server* The name of a batch server, defined as a string of alphanumeric characters in  
 4608 the portable character set.

4609 If the application omits the batch *server* portion of a destination, then the utility shall use either:

- 4610 • The name of the default batch server

4611 or:

- 4612 • The name of the batch server that is currently managing the batch job

4613 The means by which a utility determines the default batch server is implementation-defined.

4614 If the application omits the *queue* portion of a destination, then the utility shall use the name of  
 4615 the default queue at the batch server chosen.

4616 The means by which a batch server determines its default queue is implementation-defined.

4617 If a destination is specified in the *queue*@*server* form, then the utility shall use the specified queue  
 4618 at the specified server.4619 A strictly conforming application shall use the syntax described for a destination. Whenever a  
 4620 destination is specified whose syntax is not recognized by an implementation, then a message  
 4621 shall be written to standard error and the utility shall exit with an exit status greater than zero.4622 **3.3.3 Multiple Keyword-Value Pairs**4623 For each option that can have multiple keyword-value pair arguments, the following rules shall  
 4624 apply. Examples of options that can have list-oriented option-arguments are `-u value@keyword`  
 4625 and `-l keyword=value`.

- 4626 1. If a batch utility is presented with a list-oriented option-argument for which a keyword has  
 4627 a corresponding value that begins with a single or double quote, then the utility shall stop  
 4628 interpreting the input stream for delimiters until a second single or double quote,  
 4629 respectively, is encountered. This feature allows some flexibility for a comma (',') or  
 4630 equals sign ('=') to be part of the value string for a particular keyword; for example:

4631 `keywd1='val1,val2',keywd2="val3,val4"`4632 **Note:** This may require the user to escape the quotes as in the following command:4633 `foo -xkeywd1=\'val1,val2\',keywd2=\"val3,val4\"`

- 4634 2. If a batch server is presented with a list-oriented attribute that has a keyword that was  
 4635 encountered earlier in the list, then the later entry for that keyword shall replace the earlier  
 4636 entry.

- 4637 3. If a batch server is presented with a list-oriented attribute that has a keyword without any  
 4638 corresponding value of the form *keyword*= or @*keyword* and the same keyword was  
 4639 encountered earlier in the list, then the prior entry for that keyword shall be ignored by the

- 4640 batch server.
- 4641 4. If a batch utility is expecting a list-oriented option-argument entry of the form  
4642 *keyword=value*, but is presented with an entry of the form *keyword* without any  
4643 corresponding *value*, then the entry shall be treated as though a default value of NULL was  
4644 assigned (that is, *keyword=NULL*) for entry parsing purposes. The utility shall include only  
4645 the keyword, not the NULL value, in the associated job attribute.
  - 4646 5. If a batch utility is expecting a list-oriented option-argument entry of the form  
4647 *value@keyword*, but is presented with an entry of the form *value* without any corresponding  
4648 *keyword*, then the entry shall be treated as though a keyword of NULL was assigned (that  
4649 is, *value@NULL*) for entry parsing purposes. The utility shall include only the value, not  
4650 the NULL keyword, in the associated job attribute.
  - 4651 6. A batch server shall accept a list-oriented attribute that has multiple occurrences of the  
4652 same keyword, interpreting the keywords, in order, with the last value encountered taking  
4653 precedence over prior instances of the same keyword. This rule allows, but does not  
4654 require, a batch utility to preprocess the attribute to remove duplicate keywords.
  - 4655 7. If a batch utility is presented with multiple list-oriented option-arguments on the  
4656 command line or in script directives, or both, for a single option, then the utility shall  
4657 concatenate, in order, any command line keyword and value pairs to the end of any  
4658 directive keyword and value pairs separated by a single comma to produce a single string  
4659 that is an equivalent, valid option-argument. The resulting string shall be assigned to the  
4660 associated attribute of the batch job (after optionally removing duplicate entries as  
4661 described in item 6.

# *Chapter 4*

## *Utilities*

4662

4663

This chapter contains the definitions of the utilities, as follows:

4664

- Mandatory utilities that are present on every conformant system

4665

4666

4667

- Optional utilities that are present only on systems supporting the associated option; see Section 1.8.1 (on page 2212) for information on the options in this volume of IEEE Std. 1003.1-200x

## 4668 NAME

4669 admin — create and administer SCCS files (DEVELOPMENT)

## 4670 SYNOPSIS

```
4671 xSI admin -i[name][-n][-a login][-d flag][-f flag][-m mrlist][-r rel]
4672 [-t[name][-y[comment]] newfile
```

```
4673 admin -n[-a login][-d flag][-f flag][-m mrlist][-t[name]][-y[comment]]
4674 newfile ...
```

```
4675 admin [-a login][-d flag][-m mrlist][-r rel][-t[name]] file ...
```

```
4676 admin -h file ...
```

```
4677 admin -z file ...
```

4678

## 4679 DESCRIPTION

4680 The *admin* utility shall create new SCCS files or change parameters of existing ones. If a named  
 4681 file does not exist, it shall be created, and its parameters shall be initialized according to the  
 4682 specified options. Parameters not initialized by an option shall be assigned a default value. If a  
 4683 named file does exist, parameters corresponding to specified options shall be changed, and other  
 4684 parameters shall be left as is.

4685 All SCCS file names supplied by the application shall be of the form *s.filename*. New SCCS files  
 4686 shall be given read-only permission mode. Write permission in the parent directory is required  
 4687 to create a file. All writing done by *admin* shall be to a temporary *x-file*, named *x.filename* (see *get*)  
 4688 created with read-only mode if *admin* is creating a new SCCS file, or created with the same mode  
 4689 as that of the SCCS file if the file already exists. After successful execution of *admin*, the SCCS file  
 4690 shall be removed (if it exists), and the *x-file* shall be renamed with the name of the SCCS file. This  
 4691 ensures that changes are made to the SCCS file only if no errors occur.

4692 The *admin* utility shall also use a transient lock file (named *z.filename*), which is used to prevent  
 4693 simultaneous updates to the SCCS file; see *get* (on page 2685).

## 4694 OPTIONS

4695 The *admin* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 4696 12.2, Utility Syntax Guidelines, except that the *-i*, *-t*, and *-y* options have optional option-  
 4697 arguments. These optional option-arguments shall not be presented as separate arguments. The  
 4698 following options are supported:

4699 **-n** Create a new SCCS file. When *-n* is used without *-i*, the SCCS file shall be created  
 4700 with control information but without any file data.

4701 **-i[name]** Specify the *name* of a file from which the text for a new SCCS file shall be taken.  
 4702 The text constitutes the first delta of the file (see the *-r* option for delta numbering  
 4703 scheme). If the *-i* option is used, but the *name* option-argument is omitted, the text  
 4704 shall be obtained by reading the standard input. If this option is omitted, the SCCS  
 4705 file shall be created with control information but without any file data. The *-i*  
 4706 option implies the *-n* option.

4707 **-r rel** Specify the *release* into which the initial delta is inserted. If the *-r* option is not  
 4708 used, the initial delta shall be inserted into release 1. The level of the initial delta  
 4709 shall always be 1 (by default, initial deltas are named 1.1).

4710 **-t[name]** Specify the *name* of a file from which descriptive text for the SCCS file shall be  
 4711 taken. In the case of existing SCCS files (neither *-i* nor *-n* is specified):



- 4712                   • A **-t** option without a *name* option-argument shall cause the removal of  
4713                   descriptive text (if any) currently in the SCCS file.
- 4714                   • A **-t** option with a *name* option-argument shall cause the text (if any) in the  
4715                   named file to replace the descriptive text (if any) currently in the SCCS file.
- 4716           **-f flag**   Specify a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file.  
4717                   Several **-f** options may be supplied on a single *admin* command line. The allowable  
4718                   flags and their values are:
- 4719                   **b**           Allow use of the **-b** option on a *get* command to create branch deltas.
- 4720                   **cceil**       Specify the highest release (that is, ceiling), a number less than or equal to  
4721                   9 999, which may be retrieved by a *get* command for editing. The default  
4722                   value for an unspecified **c** flag shall be 9 999.
- 4723                   **ffloor**      Specify the lowest release (that is, floor), a number greater than 0 but less  
4724                   than 9 999, which may be retrieved by a *get* command for editing. The  
4725                   default value for an unspecified **f** flag shall be 1.
- 4726                   **dSID**       Specify the default delta number (SID) to be used by a *get* command.
- 4727                   **istr**       Treat the “No ID keywords” message issued by *get* or *delta* as a fatal  
4728                   error. In the absence of this flag, the message is only a warning. The  
4729                   message is issued if no SCCS identification keywords (see *get* (on page  
4730                   2685)) are found in the text retrieved or stored in the SCCS file. If a value  
4731                   is supplied, the application shall ensure that the keywords exactly match  
4732                   the given string; however, the string shall contain a keyword, and no  
4733                   embedded <newline>s.
- 4734                   **j**           Allow concurrent *get* commands for editing on the same SID of an SCCS  
4735                   file. This allows multiple concurrent updates to the same version of the  
4736                   SCCS file.
- 4737                   **l!list**      Specify a *list* of releases to which deltas can no longer be made (that is, *get*  
4738                   **-e** against one of these locked releases fails). The *list* has the following  
4739                   syntax:
- 4740                   <list> ::= a | <range-list>  
4741                   <range-list> ::= <range> | <range-list>, <range>
- 4742                   The character *a* in the *list* shall be equivalent to specifying all releases for  
4743                   the named SCCS file.
- 4744                   **n**           Cause *delta* to create a null delta in each of those releases (if any) being  
4745                   skipped when a delta is made in a new release (for example, in making  
4746                   delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas  
4747                   serve as anchor points so that branch deltas may later be created from  
4748                   them. The absence of this flag shall cause skipped releases to be  
4749                   nonexistent in the SCCS file, preventing branch deltas from being created  
4750                   from them in the future.
- 4751                   **qtext**      Substitute user-definable *text* for all occurrences of the %Q% keyword in  
4752                   the SCCS file text retrieved by *get*.
- 4753                   **mmod**      Specify the module name of the SCCS file substituted for all occurrences  
4754                   of the %M% keyword in the SCCS file text retrieved by *get*. If the **m** flag  
4755                   is not specified, the value assigned shall be the name of the SCCS file with  
4756                   the leading ' . ' removed.

- 4757            **type**     Specify the *type* of module in the SCCS file substituted for all occurrences  
4758            of the %Y% keyword in the SCCS file text retrieved by *get*.
- 4759            **vpgm**     Cause *delta* to prompt for modification request (MR) numbers as the  
4760            reason for creating a delta. The optional value specifies the name of an  
4761            MR number validation program. (If this flag is set when creating an SCCS  
4762            file, the application shall ensure that the **m** option is also used even if its  
4763            value is null.)
- 4764            **-d flag**    Remove (delete) the specified *flag* from an SCCS file. Several **-d** options may be  
4765            supplied on a single *admin* command. See the **-f** option for allowable *flag* names.  
4766            (The **l**list flag gives a *list* of releases to be unlocked. See the **-f** option for further  
4767            description of the **l** flag and the syntax of a *list*.)
- 4768            **-a login**    Specify a *login* name, or numerical group ID, to be added to the list of users who  
4769            may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying  
4770            all *login* names common to that group ID. Several **-a** options may be used on a  
4771            single *admin* command line. As many *logins*, or numerical group IDs, as desired  
4772            may be on the list simultaneously. If the list of users is empty, then anyone may  
4773            add deltas. If *login* or group ID is preceded by a '!', the users so specified are  
4774            denied permission to make deltas.
- 4775            **-e login**    Specify a *login* name, or numerical group ID, to be erased from the list of users  
4776            allowed to make deltas (changes) to the SCCS file. Specifying a group ID is  
4777            equivalent to specifying all *login* names common to that group ID. Several **-e**  
4778            options may be used on a single *admin* command line.
- 4779            **-y[comment]** Insert the *comment* text into the SCCS file as a comment for the initial delta in a  
4780            manner identical to that of *delta*. In the POSIX locale, omission of the **-y** option  
4781            results in a default comment line being inserted in the form:
- 4782            "date and time created %s %s by %s", <date>, <time>, <login>  
4783            where <date> is expressed in the *date* utility's %y/%m/%d format, <time> in the  
4784            *date* utility's %T format, and <login> is the login name of the user creating the file.
- 4785            **-m mrlist**    Insert the list of modification request (MR) numbers into the SCCS file as the  
4786            reason for creating the initial delta in a manner identical to *delta*. The application  
4787            shall ensure that the **v** flag is set and the MR numbers are validated if the **v** flag has  
4788            a value (the name of an MR number validation program). Diagnostics occur if the  
4789            **v** flag is not set or MR validation fails.
- 4790            **-h**         Check the structure of the SCCS file and compare the newly computed checksum  
4791            (the sum of all the characters in the SCCS file except those in the first line) with the  
4792            checksum that is stored in the first line of the SCCS file. Appropriate error  
4793            diagnostics are produced.
- 4794            **-z**         Recompute the SCCS file checksum and store it in the first line of the SCCS file (see  
4795            the **-h** option above). Note that use of this option on a truly corrupted file may  
4796            prevent future detection of the corruption.

4797 **OPERANDS**

4798         The following operands shall be supported:

- 4799            **file**         A path name of an existing SCCS file or a directory. If *file* is a directory, the *admin*  
4800            utility shall behave as though each file in the directory were specified as a named  
4801            file, except that non-SCCS files (last component of the path name does not begin  
4802            with **s**.) and unreadable files shall be silently ignored.

- 4803        *newfile*        A path name of an SCCS file to be created.
- 4804        If a single instance of *file* or *newfile* is specified as *'-'*, the standard input shall be read; each line  
4805        of the standard input shall be taken to be the name of an SCCS file to be processed. Non-SCCS  
4806        files and unreadable files shall be silently ignored.
- 4807 **STDIN**  
4808        The standard input shall be a text file used only if the *-i* is specified without an option-argument  
4809        or if a *file* or *newfile* operand is specified as *'-'*. If the first character of any standard input line is  
4810        SOH (binary 001), the results are unspecified.
- 4811 **INPUT FILES**  
4812        The existing SCCS files are text files of an unspecified format. The file named by the *-i* option's  
4813        *name* option-argument is a text file; if the first character of any line in this file is SOH (binary  
4814        001), the results are unspecified.
- 4815 **ENVIRONMENT VARIABLES**  
4816        The following environment variables shall affect the execution of *admin*:
- 4817        *LANG*        Provide a default value for the internationalization variables that are unset or null.  
4818        If *LANG* is unset or null, the corresponding value from the implementation-  
4819        defined default locale shall be used. If any of the internationalization variables  
4820        contains an invalid setting, the utility shall behave as if none of the variables had  
4821        been defined.
- 4822        *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
4823        internationalization variables.
- 4824        *LC\_CTYPE*      Determine the locale for the interpretation of sequences of bytes of text data as  
4825        characters (for example, single-byte as opposed to multi-byte characters in  
4826        arguments and input files).
- 4827        *LC\_MESSAGES*  
4828        Determine the locale that should be used to affect the format and contents of  
4829        diagnostic messages written to standard error and the contents of the default *-y*  
4830        comment.
- 4831        *NLSPATH*      Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 4832 **ASYNCHRONOUS EVENTS**  
4833        Default.
- 4834 **STDOUT**  
4835        Not used.
- 4836 **STDERR**  
4837        Used only for diagnostic messages.
- 4838 **OUTPUT FILES**  
4839        Any SCCS files created shall be text files of an unspecified format. During processing of a *file*, a  
4840        locking *z-file*, as described in *get* (on page 2685), may be created and deleted.
- 4841 **EXTENDED DESCRIPTION**  
4842        None.
- 4843 **EXIT STATUS**  
4844        The following exit values shall be returned:  
4845        0    Successful completion.

4846 >0 An error occurred.

4847 **CONSEQUENCES OF ERRORS**

4848 Default.

4849 **APPLICATION USAGE**

4850 It is recommended that directories containing SCCS files be writable by the owner only, and that  
4851 SCCS files themselves be read-only. The mode of the directories should allow only the owner to  
4852 modify SCCS files contained in the directories. The mode of the SCCS files prevents any  
4853 modification at all except by SCCS commands.

4854 **EXAMPLES**

4855 None.

4856 **RATIONALE**

4857 None.

4858 **FUTURE DIRECTIONS**

4859 None.

4860 **SEE ALSO**

4861 *delta, get, prs, what*

4862 **CHANGE HISTORY**

4863 First released in Issue 2.

4864 **Issue 4**

4865 Format reorganized.

4866 Conformance to Utility Syntax Guidelines mandated, with exceptions as noted.

4867 Internationalized environment variable support mandated.

4868 **Issue 6**

4869 The normative text is reworded to avoid use of the term “must” for application requirements. |

4870 The normative text is reworded to emphasise the term “shall” for implementation requirements. |

4871 The grammar is updated. |

4872 **NAME**

4873 alias — define or display aliases

4874 **SYNOPSIS**4875 UP alias [*alias-name*[=*string*] ...]  
48764877 **DESCRIPTION**4878 The *alias* utility shall create or redefine alias definitions or write the values of existing alias  
4879 definitions to standard output. An alias definition provides a string value that shall replace a  
4880 command name when it is encountered; see Section 2.3.1 (on page 2239).4881 An alias definition shall affect the current shell execution environment and the execution  
4882 environments of the subshells of the current shell. When used as specified by this volume of  
4883 IEEE Std. 1003.1-200x, the alias definition shall not affect the parent process of the current shell  
4884 nor any utility environment invoked by the shell; see Section 2.13 (on page 2273).4885 **OPTIONS**

4886 None.

4887 **OPERANDS**

4888 The following operands shall be supported:

4889 *alias-name* Write the alias definition to standard output.4890 *alias-name=string*4891 Assign the value of *string* to the alias *alias-name*.

4892 If no operands are given, all alias definitions shall be written to standard output.

4893 **STDIN**

4894 Not used.

4895 **INPUT FILES**

4896 None.

4897 **ENVIRONMENT VARIABLES**4898 The following environment variables shall affect the execution of *alias*:4899 *LANG* Provide a default value for the internationalization variables that are unset or null.  
4900 If *LANG* is unset or null, the corresponding value from the implementation-  
4901 defined default locale shall be used. If any of the internationalization variables  
4902 contains an invalid setting, the utility shall behave as if none of the variables had  
4903 been defined.4904 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
4905 internationalization variables.4906 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
4907 characters (for example, single-byte as opposed to multi-byte characters in  
4908 arguments).4909 *LC\_MESSAGES*4910 Determine the locale that should be used to affect the format and contents of  
4911 diagnostic messages written to standard error.4912 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

4913 **ASYNCHRONOUS EVENTS**

4914 Default.

4915 **STDOUT**4916 The format for displaying aliases (when no operands or only *name* operands are specified) shall  
4917 be:4918 "%s=%s\n", *name*, *value*4919 The *value* string shall be written with appropriate quoting so that it is suitable for reinput to the  
4920 shell. See the description of shell quoting in Section 2.2 (on page 2236).4921 **STDERR**

4922 Used only for diagnostic messages.

4923 **OUTPUT FILES**

4924 None.

4925 **EXTENDED DESCRIPTION**

4926 None.

4927 **EXIT STATUS**

4928 The following exit values shall be returned:

4929 0 Successful completion.

4930 >0 One of the *name* operands specified did not have an alias definition, or an error occurred.4931 **CONSEQUENCES OF ERRORS**

4932 Default.

4933 **APPLICATION USAGE**

4934 None.

4935 **EXAMPLES**4936 1. Change *ls* to give a columnated, more annotated output:4937 

```
alias ls="ls -CF"
```

4938 2. Create a simple “redo” command to repeat previous entries in the command history file:

4939 

```
alias r='fc -s'
```

4940 3. Use 1K units for *du*:4941 

```
alias du=du\ -k
```

4942 4. Set up *nohup* so that it can deal with an argument that is itself an alias name:4943 

```
alias nohup="nohup "
```

4944 **RATIONALE**4945 The *alias* description is based on historical KornShell implementations. Known differences exist  
4946 between that and the C shell. The KornShell version was adopted to be consistent with all the  
4947 other KornShell features in this volume of IEEE Std. 1003.1-200x, such as command line editing.4948 Since *alias* affects the current shell execution environment, it is generally provided as a shell  
4949 regular built-in.4950 Historical versions of the KornShell have allowed aliases to be exported to scripts that are  
4951 invoked by the same shell. This is triggered by the *alias -x* flag; it is allowed by this volume of  
4952 IEEE Std. 1003.1-200x only when an explicit extension such as *-x* is used. The standard  
4953 developers considered that aliases were of use primarily to interactive users and that they

4954 should normally not affect shell scripts called by those users; functions are available to such  
4955 scripts.

4956 Historical versions of the KornShell had not written aliases in a quoted manner suitable for  
4957 reentry to the shell, but this volume of IEEE Std. 1003.1-200x has made this a requirement for all  
4958 similar output. Therefore, consistency with this volume of IEEE Std. 1003.1-200x was chosen  
4959 over this detail of historical practice.

4960 **FUTURE DIRECTIONS**

4961 None.

4962 **SEE ALSO**

4963 Section 2.9.5 (on page 2263)

4964 **CHANGE HISTORY**

4965 First released in Issue 4.

4966 **Issue 6**

4967 This utility is now marked as part of the User Portability Utilities option.

4968 The APPLICATION USAGE section is added.

4969 **NAME**

4970 ar — create and maintain library archives

4971 **SYNOPSIS**

4972 SD ar -d[-v] archive file ...

4973

4974 XSI ar -m[-abiv][posname] archive file ...

4975

4976 XSI ar -p[-v][-s]archive [file ...]

4977 XSI ar -q[-cv] archive file ...

4978

4979 XSI ar -r[-cuv][-abi][posname]archive file ...

4980 XSI ar -t[-v][-s]archive [file ...]

4981 XSI ar -x[-v][-sCT]archive [file ...]

4982 **DESCRIPTION**

4983 The *ar* utility can be used to create and maintain groups of files combined into an archive. Once  
 4984 an archive has been created, new files can be added, and existing files can be extracted, deleted,  
 4985 or replaced. When an archive consists entirely of valid object files, the implementation shall  
 4986 format the archive so that it is usable as a library for link editing (see *c99*, *cc*, and *fort77*). When  
 4987 some of the archived files are not valid object files, the suitability of the archive for library use is  
 4988 XSI undefined. If an archive file consists entirely of printable files, the entire archive file is printable.

4989 When *ar* creates an archive file, it creates administrative information indicating whether a  
 4990 symbol table is present in the archive. When there is at least one object file that *ar* recognizes as  
 4991 such in the archive, an archive symbol table is created in the archive file and maintained by *ar*; it  
 4992 is used by the link editor to search the archive file. Whenever the *ar* utility is used to create or  
 4993 update the contents of such an archive, the symbol table is rebuilt. The *-s* option forces the  
 4994 symbol table to be rebuilt.

4995 All *file* operands can be path names. However, files within archives shall be named by a file  
 4996 name, which is the last component of the path name used when the file was entered into the  
 4997 archive. The comparison of *file* operands to the names of files in archives shall be performed by  
 4998 comparing the last component of the operand to the name of the archive file.

4999 It is unspecified whether multiple files in the archive may be identically named. In the case of  
 5000 XSI such files, however, each *file* and *posname* operand shall match only the first archive file having a  
 5001 name that is the same as the last component of the operand.

5002 **OPTIONS**

5003 The *ar* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,  
 5004 Utility Syntax Guidelines.

5005 The following options shall be supported:

5006 XSI **-a** Position new files in the archive after the file named by the *posname* operand.

5007 XSI **-b** Position new files in the archive before the file named by the *posname* operand.

5008 **-c** Suppress the diagnostic message that is written to standard error by default when  
 5009 the archive file *archive* is created.

5010 XSI **-C** Prevent extracted files from replacing like-named files in the file system. This  
 5011 option is useful when **-T** is also used, to prevent truncated file names from  
 5012 replacing files with the same prefix.



- 5013           **-d**           Delete one or more *files* from *archive*.
- 5014 XSI       **-i**           Position new files in the archive before the file named by the *posname* operand  
5015                   (equivalent to **-b**).
- 5016 XSI       **-m**           Move the named files. The **-a**, **-b**, or **-i** options with the *posname* operand indicate  
5017                   the position; otherwise, move the files to the end of the archive.
- 5018           **-p**           Write the contents of the *files* from *archive* to the standard output. If no *files* are  
5019                   specified, the contents of all files in the archive shall be written in the order of the  
5020                   archive.
- 5021 XSI       **-q**           Quickly append the named files to the end of the archive file. In this case *ar* does  
5022                   not check whether the added members are already in the archive. This is useful to  
5023                   bypass the searching otherwise done when creating a large archive piece by piece.
- 5024           **-r**           Replace or add *files* to *archive*. If the archive named by *archive* does not exist, a  
5025                   new archive file shall be created and a diagnostic message shall be written to  
5026                   standard error (unless the **-c** option is specified). If no *files* are specified and the  
5027                   *archive* exists, the results are undefined. Files that replace existing files shall not  
5028                   change the order of the archive. Files that do not replace existing files shall be  
5029 XSI           appended to the archive unless a **-a**, **-b**, or **-i** option specifies another position.
- 5030 XSI       **-s**           Force the regeneration of the archive symbol table even if *ar* is not invoked with an  
5031                   option that modifies the archive file contents. This option is useful to restore the  
5032                   archive symbol table after it has been stripped; see *strip*.
- 5033           **-t**           Write a table of contents of *archive* to the standard output. The files specified by the  
5034                   *file* operands shall be included in the written list. If no *file* operands are specified,  
5035                   all files in *archive* shall be included in the order of the archive.
- 5036 XSI       **-T**           Allow file name truncation of extracted files whose archive names are longer than  
5037                   the file system can support. By default, extracting a file with a name that is too  
5038                   long is an error; a diagnostic message is written and the file is not extracted.
- 5039           **-u**           Update older files. When used with the **-r** option, files within the archive are  
5040                   replaced only if the corresponding *file* has a modification time that is at least as  
5041                   new as the modification time of the file within the archive.
- 5042           **-v**           Give verbose output. When used with the option characters **-d**, **-r**, or **-x**, write a  
5043                   detailed file-by-file description of the archive creation and maintenance activity, as  
5044                   described in the STDOUT section.
- 5045                   When used with **-p**, write the name of the file to the standard output before  
5046                   writing the file itself to the standard output, as described in the STDOUT section.
- 5047                   When used with **-t**, include a long listing of information about the files within the  
5048                   archive, as described in the STDOUT section.
- 5049           **-x**           Extract the files named by the *file* operands from *archive*. The contents of the  
5050                   archive file shall not be changed. If no *file* operands are given, all files in the  
5051                   archive shall be extracted. The modification time of each file extracted shall be set  
5052                   to the time the file is extracted from the archive.

#### 5053 OPERANDS

5054           The following operands shall be supported:

5055           *archive*       A path name of the archive file.

5056 *file* A path name. Only the last component shall be used when comparing against the  
5057 names of files in the archive. If two or more *file* operands have the same last path  
5058 name component (basename), the results are unspecified. The implementation's  
5059 archive format shall not truncate valid file names of files added to or replaced in  
5060 the archive.

5061 XSI *posname* The name of a file in the archive file, used for relative positioning; see options **-m**  
5062 and **-r**.

5063 **STDIN**  
5064 Not used.

5065 **INPUT FILES**  
5066 The input file named by *archive* shall be a file in the format created by *ar -r*.

5067 **ENVIRONMENT VARIABLES**  
5068 The following environment variables shall affect the execution of *ar*:

5069 *LANG* Provide a default value for the internationalization variables that are unset or null.  
5070 If *LANG* is unset or null, the corresponding value from the implementation-  
5071 defined default locale shall be used. If any of the internationalization variables  
5072 contains an invalid setting, the utility shall behave as if none of the variables had  
5073 been defined.

5074 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
5075 internationalization variables.

5076 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
5077 characters (for example, single-byte as opposed to multi-byte characters in  
5078 arguments and input files).

5079 *LC\_MESSAGES*  
5080 Determine the locale that should be used to affect the format and contents of  
5081 diagnostic messages written to standard error.

5082 *LC\_TIME* Determine the format and content for date and time strings written by *ar -tv*.

5083 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

5084 *TMPDIR* Determine the path name that overrides the default directory for temporary files, if  
5085 any.

5086 **ASYNCHRONOUS EVENTS**  
5087 Default.

5088 **STDOUT**  
5089 If the **-d** option is used with the **-v** option, the standard output format shall be:  
5090 "d - %s\n", <*file*>  
5091 where *file* is the operand specified on the command line.  
5092 If the **-p** option is used with the **-v** option, *ar* shall precede the contents of each file with:  
5093 "\n<%s>\n\n", <*file*>  
5094 where *file* is the operand specified on the command line, if *file* operands were specified, and the  
5095 name of the file in the archive if they were not.  
5096 If the **-r** option is used with the **-v** option:

- 5097           • If *file* is already in the archive, the standard output format shall be:
- 5098           "r - %s\n", <*file*>
- 5099           where <*file*> is the operand specified on the command line.
- 5100           • If *file* is not already in the archive, the standard output format shall be:
- 5101           "a - %s\n", <*file*>
- 5102           where <*file*> is the operand specified on the command line.

### 5103 **Notes to Reviewers**

5104           *This section with side shading will not appear in the final copy. - Ed.*

5105           D3, XCU, ERN 48 suggests changing the above to "where <*file*> is the member name found to be  
5106           in conflict". If the command line contains a path name which is not a simple file name (that is,  
5107           contains a slash), does it print the member name (which seems what's intended) or the actual  
5108           text from the command line (which is what's said)? This will eventually need to be an  
5109           interpretation against .2b.

5110           If the **-t** option is used, *ar* shall write the names of the files to the standard output in the format:

5111           "%s\n", <*file*>

5112           where *file* is the operand specified on the command line, if *file* operands were specified, or the  
5113           name of the file in the archive if they were not.

5114           If the **-t** option is used with the **-v** option, the standard output format shall be:

5115           "%s %u/%u %u %s %d %d:%d %d %s\n", <*member mode*>, <*user ID*>,  
5116           <*group ID*>, <*number of bytes in member*>,  
5117           <*abbreviated month*>, <*day-of-month*>, <*hour*>,  
5118           <*minute*>, <*year*>, <*file*>

5119           where:

5120           <*file*>           Shall be the operand specified on the command line, if *file* operands were specified,  
5121           or the name of the file in the archive if they were not.

5122           <*member*

5123                           *Shall be formatted the same as the <file mode> string defined in the STDOUT section of*  
5124                           *ls, except that the first character, the <entry type>, is not used; the string represents*  
5125                           *the file mode of the archive member at the time it was added to or replaced in the*  
5126                           *archive.*

5127           The following represent the last-modification time of a file when it was most recently added to  
5128           or replaced in the archive:

5129           <*abbreviated month*>

5130                           Equivalent to the *%b* format in *date*.

5131           <*day-of-month*>

5132                           Equivalent to the *%e* format in *date*.

5133           <*hour*>           Equivalent to the *%H* format in *date*.

5134           <*minute*>       Equivalent to the *%M* format in *date*.

5135           <*year*>           Equivalent to the *%Y* format in *date*.

5136 When *LC\_TIME* does not specify the POSIX locale, a different format and order of presentation  
5137 of these fields relative to each other may be used in a format appropriate in the specified locale.

5138 If the *-x* option is used with the *-v* option, the standard output format shall be:

5139 "x - %s\n", <file>

5140 where *file* is the operand specified on the command line, if *file* operands were specified, or the  
5141 name of the file in the archive if they were not.

#### 5142 **STDERR**

5143 Used only for diagnostic messages. The diagnostic message about creating a new archive when  
5144 *-c* is not specified shall not modify the exit status.

#### 5145 **OUTPUT FILES**

5146 Archives are files with unspecified formats.

#### 5147 **EXTENDED DESCRIPTION**

5148 None.

#### 5149 **EXIT STATUS**

5150 The following exit values shall be returned:

5151 0 Successful completion.

5152 >0 An error occurred.

#### 5153 **CONSEQUENCES OF ERRORS**

5154 Default.

#### 5155 **APPLICATION USAGE**

5156 None.

#### 5157 **EXAMPLES**

5158 None.

#### 5159 **RATIONALE**

5160 The archive format is not described. It is recognized that there are several known *ar* formats,  
5161 which are not compatible. The *ar* utility is included, however, to allow creation of archives that  
5162 are intended for use only on one machine. The archive file is specified as a file, and it can be  
5163 moved as a file. This does allow an archive to be moved from one machine to another machine  
5164 that uses the same implementation of *ar*.

5165 Utilities such as *pax* (and its forebears *tar* and *cpio*) also provide portable “archives”. This is a not  
5166 a duplication; the *ar* utility is included to provide an interface primarily for *make* and the  
5167 compilers, based on a historical model.

5168 In historical implementations, the *-q* option (available on XSI-conforming systems) is known to  
5169 execute quickly because *ar* does not check on whether the added members are already in the  
5170 archive. This is useful to bypass the searching otherwise done when creating a large archive  
5171 piece-by-piece. These remarks may but need not remain true for a brand new implementation of  
5172 this utility; hence, these remarks have been moved into the RATIONALE.

5173 BSD implementations historically required applications to provide the *-s* option whenever the  
5174 archive was supposed to contain a symbol table. As in this volume of IEEE Std. 1003.1-200x,  
5175 System V historically creates or updates an archive symbol table whenever an object file is  
5176 removed from, added to, or updated in the archive.

5177 The OPERANDS section requires what might seem to be true without specifying it: the archive  
5178 cannot truncate the file names below {NAME\_MAX}. Some historical implementations do so,  
5179 however, causing unexpected results for the application. Therefore, this volume of

5180 IEEE Std. 1003.1-200x makes the requirement explicit to avoid misunderstandings.

5181 According to the System V documentation, the options `-dmpqrtx` are not required to begin with  
5182 a hyphen ('-'). This volume of IEEE Std. 1003.1-200x requires that a conforming application  
5183 use the leading hyphen.

5184 The archive format used by the 4.4 BSD implementation is documented in this RATIONALE as  
5185 an example:

5186 A file created by *ar* begins with the "magic" string "`!<arch>\n`". The rest of the archive is  
5187 made up of objects, each of which is composed of a header for a file, a possible file name, and  
5188 the file contents. The header is portable between machine architectures, and, if the file  
5189 contents are printable, the archive is itself printable.

5190 The header is made up of six ASCII fields, followed by a two-character trailer. The fields are  
5191 the object name (16 characters), the file last modification time (12 characters), the user and  
5192 group IDs (each 6 characters), the file mode (8 characters), and the file size (10 characters). All  
5193 numeric fields are in decimal, except for the file mode, which is in octal.

5194 The modification time is the file *st\_mtime* field. The user and group IDs are the file *st\_uid* and  
5195 *st\_gid* fields. The file mode is the file *st\_mode* field. The file size is the file *st\_size* field. The  
5196 two-byte trailer is the string "`<newline>`".

5197 Only the name field has any provision for overflow. If any file name is more than 16  
5198 characters in length or contains an embedded space, the string "`#1/`" followed by the ASCII  
5199 length of the name is written in the name field. The file size (stored in the archive header) is  
5200 incremented by the length of the name. The name is then written immediately following the  
5201 archive header.

5202 Any unused characters in any of these fields are written as `<space>` characters. If any fields  
5203 are their particular maximum number of characters in length, there is no separation between  
5204 the fields.

5205 Objects in the archive are always an even number of bytes long; files that are an odd number  
5206 of bytes long are padded with a `<newline>` character, although the size in the header does  
5207 not reflect this.

5208 The *ar* utility description requires that (when all its members are valid object files) *ar* produce an  
5209 object code library, which the linkage editor can use to extract object modules. If the linkage  
5210 editor needs a symbol table to permit random access to the archive, *ar* must provide it; however,  
5211 *ar* does not require a symbol table.

5212 The BSD `-o` option was omitted. It is a rare portable application that uses *ar* to extract object  
5213 code from a library with concern for its modification time, since this can only be of importance  
5214 to *make*. Hence, since this functionality is not deemed important for applications portability, the  
5215 modification time of the extracted files is set to the current time.

5216 There is at least one known implementation (for a small computer) that can accommodate only  
5217 object files for that system, disallowing mixed object and other files. The ability to handle any  
5218 type of file is not only historical practice for most implementations, but is also a reasonable  
5219 expectation.

5220 Consideration was given to changing the output format of *ar -tv* to the same format as the  
5221 output of *ls -l*. This would have made parsing the output of *ar* the same as that of *ls*. This was  
5222 rejected in part because the current *ar* format is commonly used and changes would break  
5223 historical usage. Second, *ar* gives the user ID and group ID in numeric format separated by a  
5224 slash. Changing this to be the user name and group name would not be correct if the archive  
5225 were moved to a machine that contained a different user database. Since *ar* cannot know

- 5226 whether the archive file was generated on the same machine, it cannot tell what to report.
- 5227 The text on the `-ur` option combination is historical practice—since one file name can easily  
5228 represent two different files (for example, `/a/foo` and `/b/foo`), it is reasonable to replace the  
5229 member in the archive even when the modification time in the archive is identical to that in the  
5230 file system.
- 5231 **FUTURE DIRECTIONS**
- 5232 None.
- 5233 **SEE ALSO**
- 5234 *c99*, *pax*, *strip* the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 13, Headers, |  
5235 `<unistd.h>` description of `{POSIX_NO_TRUNC}`
- 5236 **CHANGE HISTORY**
- 5237 First released in Issue 2.
- 5238 **Issue 4**
- 5239 Aligned with the ISO/IEC 9945-2:1993 standard.
- 5240 The `-C` and `-T` options are added.
- 5241 **Issue 5**
- 5242 FUTURE DIRECTIONS section added.
- 5243 **Issue 6**
- 5244 This utility is now marked as part of the Software Development Utilities option.
- 5245 The `STDOUT` description is changed for the `-v` option to align with the IEEE P1003.2b draft  
5246 standard.
- 5247 The normative text is reworded to avoid use of the term “must” for application requirements.

5248 **NAME**

5249           asa — interpret carriage-control characters

5250 **SYNOPSIS**5251 FR       asa [ *file* ... ]

5252

5253 **DESCRIPTION**5254           The *asa* utility shall write its input files to standard output, mapping carriage-control characters from the text files to line-printer control sequences in an implementation-defined manner.

5256           The first character of every line shall be removed from the input, and the following actions are performed:

5258           If the character removed is:

5259           &lt;space&gt; The rest of the line is output without change.

5260           0        A &lt;newline&gt; character is output, then the rest of the input line.

5261           1        One or more implementation-defined characters that causes an advance to the next page shall be output, followed by the rest of the input line.

5263           +        The &lt;newline&gt; character of the previous line shall be replaced with one or more implementation-defined characters that causes printing to return to column position 1, followed by the rest of the input line. If the '+' is the first character in the input, it shall have the same effect as the &lt;space&gt; character.

5267           The action of the *asa* utility is unspecified upon encountering any character other than those listed above as the first character in a line.5269 **OPTIONS**

5270           None.

5271 **OPERANDS**5272           *file*        A path name of a text file used for input. If no *file* operands are specified, the standard input shall be used.5274 **STDIN**5275           The standard input is used only if no *file* operands are specified; see the INPUT FILES section.5276 **INPUT FILES**

5277           The input files shall be text files.

5278 **ENVIRONMENT VARIABLES**5279           The following environment variables shall affect the execution of *asa*:5280           *LANG*        Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-defined default locale shall be used. If any of the internationalization variables contains an invalid setting, the utility shall behave as if none of the variables had been defined.5285           *LC\_ALL*       If set to a non-empty string value, override the values of all the other internationalization variables.5287           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

5290 *LC\_MESSAGES*  
5291 Determine the locale that should be used to affect the format and contents of  
5292 diagnostic messages written to standard error.

5293 XSI *NLS\_PATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

5294 **ASYNCHRONOUS EVENTS**  
5295 Default.

5296 **STDOUT**  
5297 The standard output shall be the text from the input file modified as described in the  
5298 DESCRIPTION section.

5299 **STDERR**  
5300 None.

5301 **OUTPUT FILES**  
5302 None.

5303 **EXTENDED DESCRIPTION**  
5304 None.

5305 **EXIT STATUS**  
5306 The following exit values shall be returned:  
5307 0 All input files were output successfully.  
5308 >0 An error occurred.

5309 **CONSEQUENCES OF ERRORS**  
5310 Default.

5311 **APPLICATION USAGE**  
5312 None.

5313 **EXAMPLES**  
5314 1. The following command:  
5315 *asa file*  
5316 permits the viewing of *file* (created by a program using FORTRAN-style carriage control  
5317 characters) on a terminal.  
5318 2. The following command:  
5319 *a.out | asa | lp*  
5320 formats the FORTRAN output of **a.out** and directs it to the printer.

5321 **RATIONALE**  
5322 The *asa* utility is needed to map “standard” FORTRAN 77 output into a form acceptable to  
5323 contemporary printers. Usually, *asa* is used to pipe data to the *lp* utility; see *lp*.  
5324 This utility is generally used only by FORTRAN programs. The standard developers decided to  
5325 retain *asa* to avoid breaking the historical large base of FORTRAN applications that put  
5326 carriage-control characters in their output files. There is no requirement that a system have a  
5327 FORTRAN compiler in order to run applications that need *asa*.  
5328 Historical implementations have used an ASCII <form-feed> character in response to a 1 and an  
5329 ASCII <carriage-return> in response to a '+'. It is suggested that implementations treat  
5330 characters other than 0, 1, and '+' as <space> in the absence of any compelling reason to do  
5331 otherwise. However, the action is listed here as “unspecified”, permitting an implementation to



5332 provide extensions to access fast multiple-line slewing and channel seeking in a non-portable  
5333 manner.

5334 **FUTURE DIRECTIONS**

5335 None.

5336 **SEE ALSO**

5337 *fort77, lp*

5338 **CHANGE HISTORY**

5339 First released in Issue 4.

5340 **Issue 6**

5341 This utility is now marked as part of the FORTRAN Runtime Utilities option.

5342 The normative text is reworded to avoid use of the term “must” for application requirements.

## 5343 NAME

5344 at — execute commands at a later time

## 5345 SYNOPSIS

5346 UP at [-m][-f *file*][-q *queuename*] -t *time\_arg*5347 at [-m][-f *file*][-q *queuename*] *timespec* ...5348 at -r *at\_job\_id* ...5349 at -l -q *queuename*5350 at -l [*at\_job\_id* ...]

5351

## 5352 DESCRIPTION

5353 The *at* utility shall read commands from standard input and group them together as an *at-job*, to  
5354 be executed at a later time.5355 The *at-job* shall be executed in a separate invocation of the shell, running in a separate process  
5356 group with no controlling terminal, except that the environment variables, current working  
5357 directory, file creation mask, and other implementation-defined execution-time attributes in  
5358 effect when the *at* utility is executed shall be retained and used when the *at-job* is executed.5359 When the *at-job* is submitted, the *at\_job\_id* and scheduled time shall be written to standard error.  
5360 The *at\_job\_id* is an identifier that shall be a string consisting solely of alphanumeric characters  
5361 and the period character. The *at\_job\_id* shall be assigned by the system when the job is scheduled  
5362 such that it uniquely identifies a particular job.5363 User notification and the processing of the job's standard output and standard error are  
5364 described under the **-m** option.5365 XSI Users are permitted to use *at* if their name appears in the file **/usr/lib/cron/at.allow**. If that file  
5366 does not exist, the file **/usr/lib/cron/at.deny** is checked to determine whether the user should be  
5367 denied access to *at*. If neither file exists, only a process with the appropriate privileges is  
5368 allowed to submit a job. If only **at.deny** exists and is empty, global usage is permitted. The  
5369 **at.allow** and **at.deny** files consist of one user name per line.

## 5370 OPTIONS

5371 The *at* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,  
5372 Utility Syntax Guidelines.

5373 The following options shall be supported:

5374 **-f file** Specify the path name of a file to be used as the source of the *at-job*, instead of  
5375 standard input.5376 **-l** (The letter ell.) Report all jobs scheduled for the invoking user if no *at\_job\_id*  
5377 operands are specified. If *at\_job\_ids* are specified, report only information for these  
5378 jobs. The output shall be written to standard output.5379 **-m** Send mail to the invoking user after the *at-job* has run, announcing its completion.  
5380 Standard output and standard error produced by the *at-job* shall be mailed to the  
5381 user as well, unless redirected elsewhere. Mail shall be sent even if the job  
5382 produces no output.5383 If **-m** is not used, the job's standard output and standard error shall be provided to  
5384 the user by means of mail, unless they are redirected elsewhere; if there is no such  
5385 output to provide, the implementation need not notify the user of the job's  
5386 completion.

- 5387           **-q** *queuename*  
 5388                   Specify in which queue to schedule a job for submission. When used with the **-l**  
 5389                   option, limit the search to that particular queue. By default, at-jobs shall be  
 5390                   scheduled in queue *a*. In contrast, queue *b* shall be reserved for batch jobs; see  
 5391                   *batch*. The meanings of all other *queuenames* are implementation-defined. If **-q** is  
 5392                   specified along with either of the **-t** *time\_arg* or *timespec* arguments, the results are  
 5393                   unspecified.
- 5394           **-r**           Remove the jobs with the specified *at\_job\_id* operands that were previously  
 5395                   scheduled by the *at* utility.
- 5396           **-t** *time\_arg*   Submit the job to be run at the time specified by the *time* option-argument, which  
 5397                   the application shall ensure has the format as specified by the *touch -t time* utility.

5398 **OPERANDS**

5399           The following operands shall be supported:

5400           **at\_job\_id**       The name reported by a previous invocation of the *at* utility at the time the job was  
 5401                   scheduled.

5402           **timespec**       Submit the job to be run at the date and time specified. All of the *timespec* operands  
 5403                   are interpreted as if they were separated by <space> characters and concatenated,  
 5404                   and shall be parsed as described in the grammar at the end of this section. The date  
 5405                   and time shall be interpreted as being in the timezone of the user (as determined  
 5406                   by the *TZ* variable), unless a timezone name appears as part of *time*, below.

5407                   In the POSIX locale, the following describes the three parts of the time  
 5408                   specification string. All of the values from the *LC\_TIME* categories in the POSIX  
 5409                   locale shall be recognized in a case-insensitive manner.

5410           **time**           The time can be specified as one, two, or four digits. One-digit and  
 5411                   two-digit numbers shall be taken to be hours; four-digit numbers to  
 5412                   be hours and minutes. The time can alternatively be specified as two  
 5413                   numbers separated by a colon, meaning *hour:minute*. An AM/PM  
 5414                   indication (one of the values from the **am\_pm** keywords in the  
 5415                   *LC\_TIME* locale category) can follow the time; otherwise, a 24-hour  
 5416                   clock time shall be understood. A timezone name can also follow to  
 5417                   further qualify the time. The acceptable timezone names are  
 5418                   implementation-defined, except that they shall be case-insensitive  
 5419                   and the string **utc** is supported to indicate the time is in Coordinated  
 5420                   Universal Time. In the POSIX locale, the *time* field can also be one of  
 5421                   the following tokens:

5422                   **midnight**       Indicates the time 12:00 am (00:00).

5423                   **noon**           Indicates the time 12:00 pm.

5424                   **now**           Indicates the current day and time. Invoking *at <now>*  
 5425                   shall submit an at-job for potentially immediate  
 5426                   execution (that is, subject only to unspecified  
 5427                   scheduling delays).

5428           **date**           An optional *date* can be specified as either a month name (one of the  
 5429                   values from the **mon** or **abmon** keywords in the *LC\_TIME* locale  
 5430                   category) followed by a day number (and possibly year number  
 5431                   preceded by a comma), or a day of the week (one of the values from  
 5432                   the **day** or **abday** keywords in the *LC\_TIME* locale category). In the  
 5433                   POSIX locale, two special days shall be recognized:

5434                   **today**       Indicates the current day.

5435                   **tomorrow**   Indicates the day following the current day.

5436                   If no *date* is given, **today** shall be assumed if the given time is greater  
5437                   than the current time, and **tomorrow** shall be assumed if it is less. If  
5438                   the given month is less than the current month (and no year is given),  
5439                   next year shall be assumed.

5440                   *increment*    The optional *increment* shall be a number preceded by a plus sign  
5441                   ('+') and suffixed by one of the following: **minutes, hours, days,**  
5442                   **weeks, months, or years.** (The singular forms shall be also  
5443                   accepted.) The keyword **next** shall be equivalent to an increment  
5444                   number of +1. For example, the following are equivalent commands:

5445                               at 2pm + 1 week  
5446                               at 2pm next week

5447                   The following grammar describes the precise format of *timespec* in the POSIX locale. The general  
5448                   conventions for this style of grammar are described in Section 1.10 (on page 2223). This formal  
5449                   syntax shall take precedence over the preceding text syntax description. The longest possible  
5450                   token or delimiter shall be recognized at a given point. When used in a *timespec*, white space  
5451                   shall also delimit tokens.

```
5452 %token hr24clock_hr_min
5453 %token hr24clock_hour
5454 /*
5455 A hr24clock_hr_min is a one, two, or four-digit number. A one-digit
5456 or two-digit number constitutes a hr24clock_hour. A hr24clock_hour
5457 may be any of the single digits 0-9, or may be double digits, ranging
5458 from 00-23. If a hr24clock_hr_min is a four digit number, the
5459 first two digits shall be a valid hr24clock_hour, while the last two
5460 represent the number of minutes, from 00-59.
5461 */
```

```
5462 %token wallclock_hr_min
5463 %token wallclock_hour
5464 /*
5465 A wallclock_hr_min is a one, two-digit, or four-digit number.
5466 A one-digit or two-digit number constitutes a wallclock_hour.
5467 A wallclock_hour may be any of the single digits 1-9, or may
5468 be double digits, ranging from 01-12. If a wallclock_hr_min
5469 is a four-digit number, the first two digits shall be a valid
5470 wallclock_hour, while the last two represent the number of
5471 minutes, from 00-59.
5472 */
```

```
5473 %token minute
5474 /*
5475 A minute is a one or two-digit number whose values can be 0-9
5476 or 00-59.
5477 */
```

```
5478 %token day_number
5479 /*
5480 A day_number is a number in the range appropriate for the particular
5481 month and year specified by month_name and year_number, respectively.
```

```

5482 If no year_number is given, the current year is assumed if the given
5483 date and time are later this year. If no year_number is given and
5484 the date and time have already occurred this year and the month is
5485 not the current month, next year is the assumed year.
5486 */

5487 %token year_number
5488 /*
5489 A year_number is a four-digit number representing the year A.D., in
5490 which the at_job is to be run.
5491 */

5492 %token inc_number
5493 /*
5494 The inc_number is the number of times the succeeding increment
5495 period is to be added to the specified date and time.
5496 */

5497 %token timezone_name
5498 /*
5499 The name of an optional timezone suffix to the time field, in an
5500 implementation-defined format.
5501 */

5502 %token month_name
5503 /*
5504 One of the values from the mon or abmon keywords in the LC_TIME
5505 locale category.
5506 */

5507 %token day_of_week
5508 /*
5509 One of the values from the day or abday keywords in the LC_TIME
5510 locale category.
5511 */

5512 %token am_pm
5513 /*
5514 One of the values from the am_pm keyword in the LC_TIME locale
5515 category.
5516 */

5517 %start timespec
5518 %%
5519 timespec : time
5520 | time date
5521 | time increment
5522 | time date increment
5523 | nowspec
5524 ;

5525 nowspec : "now"
5526 | "now" increment
5527 ;

5528 time : hr24clock_hr_min
5529 | hr24clock_hr_min timezone_name

```

```

5530 | hr24clock_hour ":" minute
5531 | hr24clock_hour ":" minute timezone_name
5532 | wallclock_hr_min am_pm
5533 | wallclock_hr_min am_pm timezone_name
5534 | wallclock_hour ":" minute am_pm
5535 | wallclock_hour ":" minute am_pm timezone_name
5536 | "noon"
5537 | "midnight"
5538 ;

5539 date : month_name day_number
5540 | month_name day_number "," year_number
5541 | day_of_week
5542 | "today"
5543 | "tomorrow"
5544 ;

5545 increment : "+" inc_number inc_period
5546 | "next" inc_period
5547 ;

5548 inc_period : "minute" | "minutes"
5549 | "hour" | "hours"
5550 | "day" | "days"
5551 | "week" | "weeks"
5552 | "month" | "months"
5553 | "year" | "years"
5554 ;

```

#### 5555 STDIN

5556 The standard input shall be a text file consisting of commands acceptable to the shell command  
 5557 language described in Chapter 2 (on page 2235). The standard input shall only be used if no `-f`  
 5558 *file* option is specified.

#### 5559 INPUT FILES

5560 See the STDIN section.

5561 XSI The text files `/usr/lib/cron/at.allow` and `/usr/lib/cron/at.deny` contain user names, one per line, of  
 5562 users who are, respectively, authorized or denied access to the *at* and *batch* utilities.

#### 5563 ENVIRONMENT VARIABLES

5564 The following environment variables shall affect the execution of *at*:

5565 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 5566 If *LANG* is unset or null, the corresponding value from the implementation-  
 5567 defined default locale shall be used. If any of the internationalization variables  
 5568 contains an invalid setting, the utility shall behave as if none of the variables had  
 5569 been defined.

5570 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 5571 internationalization variables.

5572 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 5573 characters (for example, single-byte as opposed to multi-byte characters in  
 5574 arguments and input files).

5575 **LC\_MESSAGES**

5576 Determine the locale that should be used to affect the format and contents of

- 5577 diagnostic messages written to standard error and informative messages written to  
5578 standard output.
- 5579 XSI **NLS\_PATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 5580 **LC\_TIME** Determine the format and contents for date and time strings written and accepted  
5581 by *at*.
- 5582 **SHELL** Determine a name of a command interpreter to be used to invoke the at-job. If the  
5583 variable is unset or null, *sh* shall be used. If it is set to a value other than a name for  
5584 *sh*, the implementation shall do one of the following: use that shell; use *sh*; use the  
5585 login shell from the user database; or any of the preceding accompanied by a  
5586 warning diagnostic about which was chosen.
- 5587 **TZ** Determine the timezone. The job shall be submitted for execution at the time  
5588 specified by *timespec* or *-t time* relative to the timezone specified by the *TZ*  
5589 variable. If *timespec* specifies a timezone, it shall override *TZ*. If *timespec* does not  
5590 specify a timezone and *TZ* is unset or null, an unspecified default timezone shall  
5591 be used.
- 5592 **ASYNCHRONOUS EVENTS**
- 5593 Default.
- 5594 **STDOUT**
- 5595 When standard input is a terminal, prompts of unspecified format for each line of the user input  
5596 described in the STDIN section may be written to standard output.
- 5597 In the POSIX locale, the following shall be written to the standard output for each job when jobs  
5598 are listed in response to the *-l* option:
- 5599 "%s\t%s\n", *at\_job\_id*, <*date*>
- 5600 where *date* shall be equivalent in format to the output of:
- 5601 date +"%a %b %e %T %Y"
- 5602 The date and time written shall be adjusted so that they appear in the timezone of the user (as  
5603 determined by the *TZ* variable).
- 5604 **STDERR**
- 5605 In the POSIX locale, the following shall be written to standard error when a job has been  
5606 successfully submitted:
- 5607 "job %s at %s\n", *at\_job\_id*, <*date*>
- 5608 where *date* has the same format as is described in the STDOUT section. Neither this, nor warning  
5609 messages concerning the selection of the command interpreter, shall be considered a diagnostic  
5610 that changes the exit status.
- 5611 Diagnostic messages, if any, shall be written to standard error.
- 5612 **OUTPUT FILES**
- 5613 None.
- 5614 **EXTENDED DESCRIPTION**
- 5615 None.
- 5616 **EXIT STATUS**
- 5617 The following exit values shall be returned:
- 5618 0 The *at* utility successfully submitted, removed, or listed a job or jobs.

5619 >0 An error occurred.

## 5620 CONSEQUENCES OF ERRORS

5621 The job shall not be scheduled, removed, or listed.

## 5622 APPLICATION USAGE

5623 The format of the *at* command line shown here is guaranteed only for the POSIX locale. Other  
5624 cultures may be supported with substantially different interfaces, although implementations are  
5625 encouraged to provide comparable levels of functionality.

5626 Since the commands run in a separate shell invocation, running in a separate process group with  
5627 no controlling terminal, open file descriptors, traps, and priority inherited from the invoking  
5628 environment are lost.

5629 Some implementations do not allow substitution of different shells using *SHELL*. System V  
5630 systems, for example, have used the login shell value for the user in */etc/passwd*. To select  
5631 reliably another command interpreter, the user must include it as part of the script, such as:

```
5632 $ at 1800
5633 myshell myscript
5634 job ... at ...
5635 $
```

## 5636 EXAMPLES

5637 1. This sequence can be used at a terminal:

```
5638 at -m 0730 tomorrow
5639 sort < file >outfile
5640 EOT
```

5641 2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a  
5642 command procedure (the sequence of output redirection specifications is significant):

```
5643 at now + 1 hour <<!
5644 diff file1 file2 2>&1 >outfile | mailx mygroup
5645 !
```

5646 3. To have a job reschedule itself, *at* can be invoked from within the *at*-job. For example, this  
5647 daily processing script named **my.daily** runs every day (although *crontab* is a more  
5648 appropriate vehicle for such work):

```
5649 # my.daily runs every day
5650 daily processing
5651 at now tomorrow < my.daily
```

5652 4. The spacing of the three portions of the POSIX locale *timespec* is quite flexible as long as  
5653 there are no ambiguities. Examples of various times and operand presentation include:

```
5654 at 0815am Jan 24
5655 at 8 :15amjan24
5656 at now "+ 1day"
5657 at 5 pm FRIday
5658 at '17
5659 utc+
5660 30minutes'
```



## 5661 RATIONALE

5662 The *at* utility reads from standard input the commands to be executed at a later time. It may be  
5663 useful to redirect standard output and standard error within the specified commands.

5664 The **-t** *time* option was added as a new capability to support an internationalized way of  
5665 specifying a time for execution of the submitted job.

5666 Early proposals added a “jobname” concept as a way of giving submitted jobs names that are  
5667 meaningful to the user submitting them. The historical, system-specified *at\_job\_id* gives no  
5668 indication of what the job is. Upon further reflection, it was decided that the benefit of this was  
5669 not worth the change in historical interface. It is anticipated that considerably more  
5670 sophisticated ways of controlling background or batch work will be the subject of a future  
5671 version of this volume of IEEE Std. 1003.1-200x.

5672 The **-q** option historically has been an undocumented option, used mainly by the *batch* utility.

5673 The System V **-m** option was added to provide a method for informing users that an at-job had  
5674 completed. Otherwise, users are only informed when output to standard error or standard  
5675 output are not redirected.

5676 The behavior of *at* *<now>* was changed in an early proposal from being unspecified to  
5677 submitting a job for potentially immediate execution. Historical BSD *at* implementations  
5678 support this. Historical System V implementations give an error in that case, but a change to the  
5679 System V versions should have no backwards compatibility ramifications.

5680 On BSD-based systems, a **-u** *user* option has allowed those with appropriate privileges to access  
5681 the work of other users. Since this is primarily a system administration feature and is not  
5682 universally implemented, it has been omitted. Similarly, a specification for the output format for  
5683 user with appropriate privileges viewing the queues of other users has been omitted.

5684 The **-f** *file* option from System V is used instead of the BSD method of using the last operand as  
5685 the path name. The BSD method is ambiguous—does:

5686 `at 1200 friday`

5687 mean the same thing if there is a file named **friday** in the current directory?

5688 The *at\_job\_id* is composed of a limited character set in historical practice, and it is mandated here  
5689 to invalidate systems that might try using characters that require shell quoting or that could not  
5690 be easily parsed by shell scripts.

5691 The *at* utility varies between System V and BSD systems in the way timezones are used. On  
5692 System V systems, the *TZ* variable affects the at-job submission times and the times displayed  
5693 for the user. On BSD systems, *TZ* is not taken into account. The BSD behavior is easily achieved  
5694 with the current specification. If the user wishes to have the timezone default to that of the  
5695 system, they merely need to issue the *at* command immediately following an unsetting or null  
5696 assignment to *TZ*. For example:

5697 `TZ= at noon ...`

5698 gives the desired BSD result.

5699 While the *yacc*-like grammar specified in the OPERANDS section is lexically unambiguous with  
5700 respect to the digit strings, a lexical analyzer would probably be written to look for and return  
5701 digit strings in those cases. The parser could then check whether the digit string returned is a  
5702 valid *day\_number*, *year\_number*, and so on, based on the context.

5703 **FUTURE DIRECTIONS**

5704 None.

5705 **SEE ALSO**5706 *batch, crontab*5707 **CHANGE HISTORY**

5708 First released in Issue 2.

5709 **Issue 4**

5710 Aligned with the ISO/IEC 9945-2:1993 standard.

5711 **Issue 6**

5712 This utility is now marked as part of the User Portability Utilities option.

5713 The following new requirements on POSIX implementations derive from alignment with the  
5714 Single UNIX Specification:5715 • If **-m** is not used, the job's standard output and standard error are provided to the user by  
5716 mail.5717 The effects of using the **-q** and **-t** options as defined in the IEEE P1003.2b draft standard are  
5718 specified.

5719 The normative text is reworded to avoid use of the term “must” for application requirements.

5720 **NAME**

5721 awk — pattern scanning and processing language

5722 **SYNOPSIS**5723 awk [-F *ERE*][-v *assignment*] ... *program* [*argument* ...]5724 awk [-F *ERE*] -f *progfile* ... [-v *assignment*] ... [*argument* ...]5725 **DESCRIPTION**

5726 The *awk* utility shall execute programs written in the *awk* programming language, which is  
 5727 specialized for textual data manipulation. An *awk* program is a sequence of patterns and  
 5728 corresponding actions. When input is read that matches a pattern, the action associated with  
 5729 that pattern is carried out.

5730 Input shall be interpreted as a sequence of records. By default, a record is a line, but this can be  
 5731 changed by using the **RS** built-in variable. Each record of input shall be matched in turn against  
 5732 each pattern in the program. For each pattern matched, the associated action shall be executed.

5733 The *awk* utility shall interpret each input record as a sequence of fields where, by default, a field  
 5734 is a string of non-<blank> characters. This default white-space field delimiter can be changed by  
 5735 using the **FS** built-in variable or the **-F *ERE***. The *awk* utility shall denote the first field in a  
 5736 record \$1, the second \$2, and so on. The symbol \$0 shall refer to the entire record; setting any  
 5737 other field causes the re-evaluation of \$0. Assigning to \$0 shall reset the values of all other fields  
 5738 and the **NF** built-in variable.

5739 **OPTIONS**

5740 The *awk* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 5741 12.2, Utility Syntax Guidelines.

5742 The following options shall be supported:

5743 **-F *ERE*** Define the input field separator to be the extended regular expression *ERE*, before  
 5744 any input is read; see **Regular Expressions** (on page 2375).

5745 **-f *progfile*** Specify the path name of the file *progfile* containing an *awk* program. If multiple  
 5746 instances of this option are specified, the concatenation of the files specified as  
 5747 *progfile* in the order specified shall be the *awk* program. The *awk* program can  
 5748 alternatively be specified in the command line as a single argument.

5749 **-v *assignment***

5750 The application shall ensure that the *assignment* argument is in the same form as an  
 5751 *assignment* operand. The specified variable assignment shall occur prior to  
 5752 executing the *awk* program, including the actions associated with **BEGIN** patterns  
 5753 (if any). Multiple occurrences of this option can be specified.

5754 **OPERANDS**

5755 The following operands shall be supported:

5756 *program* If no **-f** option is specified, the first operand to *awk* shall be the text of the *awk*  
 5757 program. The application shall supply the *program* operand as a single argument to  
 5758 *awk*. If the text does not end in a <newline> character, *awk* shall interpret the text  
 5759 as if it did.

5760 *argument* Either of the following two types of *argument* can be intermixed:

5761 *file* A path name of a file that contains the input to be read, which is  
 5762 matched against the set of patterns in the program. If no *file* operands  
 5763 are specified, or if a *file* operand is '-', the standard input shall be  
 5764 used.

5765           *assignment*   An operand that begins with an underscore or alphabetic character  
 5766 from the portable character set (see the table in the Base Definitions  
 5767 volume of IEEE Std. 1003.1-200x, Section 6.1, Portable Character Set),  
 5768 followed by a sequence of underscores, digits, and alphabetic characters  
 5769 from the portable character set, followed by the '=' character, shall  
 5770 specify a variable assignment rather than a path name. The  
 5771 characters before the '=' represent the name of an *awk* variable; if  
 5772 that name is an *awk* reserved word (see **Grammar** (on page 2384)) the  
 5773 behavior is undefined. The characters following the equal sign shall  
 5774 be interpreted as if they appeared in the *awk* program preceded and  
 5775 followed by a double-quote ('"') character, as a **STRING** token (see  
 5776 **Grammar** (on page 2384)), except that if the last character is an  
 5777 unescaped backslash, it shall be interpreted as a literal backslash  
 5778 rather than as the first character of the sequence "\ ". The variable  
 5779 shall be assigned the value of that **STRING** token and, if  
 5780 appropriate, shall be considered a *numeric string* (see **Expressions in**  
 5781 **awk** (on page 2370)), the variable shall also be assigned its numeric  
 5782 value. Each such variable assignment shall occur just prior to the  
 5783 processing of the following *file*, if any. Thus, an assignment before  
 5784 the first *file* argument shall be executed after the **BEGIN** actions (if  
 5785 any), while an assignment after the last *file* argument shall occur  
 5786 before the **END** actions (if any). If there are no *file* arguments,  
 5787 assignments shall be executed before processing the standard input.

#### 5788 **STDIN**

5789           The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-';  
 5790 see the INPUT FILES section. If the *awk* program contains no actions and no patterns, but is  
 5791 otherwise a valid *awk* program, standard input and any *file* operands shall not be read and *awk*  
 5792 shall exit with a return status of zero.

#### 5793 **INPUT FILES**

5794           Input files to the *awk* program from any of the following sources shall be text files:

- 5795           • Any *file* operands or their equivalents, achieved by modifying the *awk* variables **ARGV** and
- 5796           **ARGC**
- 5797           • Standard input in the absence of any *file* operands
- 5798           • Arguments to the **getline** function

5799           Whether the variable **RS** is set to a value other than a <newline> character or not, for these files,  
 5800 implementations shall support records terminated with the specified separator up to  
 5801 {**LINE\_MAX**} bytes and may support longer records.

5802           If **-f progfile** is specified, the application shall ensure that the files named by each of the *progfile*  
 5803 option-arguments are text files containing an *awk* program.

#### 5804 **ENVIRONMENT VARIABLES**

5805           The following environment variables shall affect the execution of *awk*:

5806           **LANG**       Provide a default value for the internationalization variables that are unset or null.  
 5807           If **LANG** is unset or null, the corresponding value from the implementation-  
 5808           defined default locale shall be used. If any of the internationalization variables  
 5809           contains an invalid setting, the utility shall behave as if none of the variables had  
 5810           been defined.

- 5811 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
5812 internationalization variables.
- 5813 *LC\_COLLATE*  
5814 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
5815 character collating elements within regular expressions and in comparisons of  
5816 string values.
- 5817 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
5818 characters (for example, single-byte as opposed to multi-byte characters in  
5819 arguments and input files), the behavior of character classes within regular  
5820 expressions, the identification of characters as letters, and the mapping of  
5821 uppercase and lowercase characters for the **toupper** and **tolower** functions.
- 5822 *LC\_MESSAGES*  
5823 Determine the locale that should be used to affect the format and contents of  
5824 diagnostic messages written to standard error.
- 5825 *LC\_NUMERIC*  
5826 Determine the radix character used when interpreting numeric input, performing  
5827 conversions between numeric and string values, and formatting numeric output.  
5828 Regardless of locale, the period character (the decimal-point character of the  
5829 POSIX locale) is the decimal-point character recognized in processing *awk*  
5830 programs (including assignments in command line arguments).
- 5831 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 5832 *PATH* Determine the search path when looking for commands executed by *system(expr)*,  
5833 or input and output pipes; see the Base Definitions volume of  
5834 IEEE Std. 1003.1-200x, Chapter 8, Environment Variables.
- 5835 In addition, all environment variables shall be visible via the *awk* variable **ENVIRON**.
- 5836 **ASYNCHRONOUS EVENTS**  
5837 Default.
- 5838 **STDOUT**  
5839 The nature of the output files depends on the *awk* program.
- 5840 **STDERR**  
5841 Used only for diagnostic messages.
- 5842 **OUTPUT FILES**  
5843 The nature of the output files depends on the *awk* program.
- 5844 **EXTENDED DESCRIPTION**
- 5845 **Overall Program Structure**  
5846 An *awk* program is composed of pairs of the form:  
5847 *pattern* { *action* }  
5848 Either the pattern or the action (including the enclosing brace characters) can be omitted.  
5849 A missing pattern shall match any record of input, and a missing action shall be equivalent to:  
5850 { *print* }  
5851 Execution of the *awk* program shall start by first executing the actions associated with all **BEGIN**  
5852 patterns in the order they occur in the program. Then each *file* operand (or standard input if no

5853 files were specified) shall be processed in turn by reading data from the file until a record  
 5854 separator is seen (<newline> by default). Before the first reference to a field in the record is  
 5855 evaluated, the record shall be split into fields, according to the rules in **Regular Expressions** (on  
 5856 page 2375), using the value of **FS** that was current at the time the record was read. Each pattern  
 5857 in the program then shall be evaluated in the order of occurrence, and the action associated with  
 5858 each pattern that matches the current record executed. The action for a matching pattern shall be  
 5859 executed before evaluating subsequent patterns. Finally, the actions associated with all **END**  
 5860 patterns shall be executed in the order they occur in the program.

### 5861 Expressions in awk

5862 Expressions describe computations used in *patterns* and *actions*. In the following table, valid  
 5863 expression operations are given in groups from highest precedence first to lowest precedence  
 5864 last, with equal-precedence operators grouped between horizontal lines. In expression  
 5865 evaluation, where the grammar is formally ambiguous, higher precedence operators shall be  
 5866 evaluated before lower precedence operators. In this table *expr*, *expr1*, *expr2*, and *expr3* represent  
 5867 any expression, while *lvalue* represents any entity that can be assigned to (that is, on the left side  
 5868 of an assignment operator). The precise syntax of expressions is given in **Grammar** (on page  
 5869 2384).

5870 **Table 4-1** Expressions in Decreasing Precedence in *awk*

| Syntax                     | Name                     | Type of Result      | Associativity |
|----------------------------|--------------------------|---------------------|---------------|
| ( <i>expr</i> )            | Grouping                 | Type of <i>expr</i> | N/A           |
| $\$$ <i>expr</i>           | Field reference          | String              | N/A           |
| ++ <i>lvalue</i>           | Pre-increment            | Numeric             | N/A           |
| -- <i>lvalue</i>           | Pre-decrement            | Numeric             | N/A           |
| <i>lvalue</i> ++           | Post-increment           | Numeric             | N/A           |
| <i>lvalue</i> --           | Post-decrement           | Numeric             | N/A           |
| <i>expr</i> ^ <i>expr</i>  | Exponentiation           | Numeric             | Right         |
| ! <i>expr</i>              | Logical not              | Numeric             | N/A           |
| + <i>expr</i>              | Unary plus               | Numeric             | N/A           |
| - <i>expr</i>              | Unary minus              | Numeric             | N/A           |
| <i>expr</i> * <i>expr</i>  | Multiplication           | Numeric             | Left          |
| <i>expr</i> / <i>expr</i>  | Division                 | Numeric             | Left          |
| <i>expr</i> % <i>expr</i>  | Modulus                  | Numeric             | Left          |
| <i>expr</i> + <i>expr</i>  | Addition                 | Numeric             | Left          |
| <i>expr</i> - <i>expr</i>  | Subtraction              | Numeric             | Left          |
| <i>expr</i> <i>expr</i>    | String concatenation     | String              | Left          |
| <i>expr</i> < <i>expr</i>  | Less than                | Numeric             | None          |
| <i>expr</i> <= <i>expr</i> | Less than or equal to    | Numeric             | None          |
| <i>expr</i> != <i>expr</i> | Not equal to             | Numeric             | None          |
| <i>expr</i> == <i>expr</i> | Equal to                 | Numeric             | None          |
| <i>expr</i> > <i>expr</i>  | Greater than             | Numeric             | None          |
| <i>expr</i> >= <i>expr</i> | Greater than or equal to | Numeric             | None          |
| <i>expr</i> ~ <i>expr</i>  | ERE match                | Numeric             | None          |
| <i>expr</i> !~ <i>expr</i> | ERE non-match            | Numeric             | None          |

5897  
5898  
5899  
5900  
5901  
5902  
5903  
5904  
5905  
5906  
5907  
5908  
5909  
5910  
5911  
5912

| Syntax                             | Name                             | Type of Result                                | Associativity |
|------------------------------------|----------------------------------|-----------------------------------------------|---------------|
| <code>expr in array</code>         | Array membership                 | Numeric                                       | Left          |
| <code>( index ) in array</code>    | Multi-dimension array membership | Numeric                                       | Left          |
| <code>expr &amp;&amp; expr</code>  | Logical AND                      | Numeric                                       | Left          |
| <code>expr    expr</code>          | Logical OR                       | Numeric                                       | Left          |
| <code>expr1 ? expr2 : expr3</code> | Conditional expression           | Type of selected <i>expr2</i> or <i>expr3</i> | Right         |
| <code>lvalue ^= expr</code>        | Exponentiation assignment        | Numeric                                       | Right         |
| <code>lvalue %= expr</code>        | Modulus assignment               | Numeric                                       | Right         |
| <code>lvalue *= expr</code>        | Multiplication assignment        | Numeric                                       | Right         |
| <code>lvalue /= expr</code>        | Division assignment              | Numeric                                       | Right         |
| <code>lvalue += expr</code>        | Addition assignment              | Numeric                                       | Right         |
| <code>lvalue -= expr</code>        | Subtraction assignment           | Numeric                                       | Right         |
| <code>lvalue = expr</code>         | Assignment                       | Type of <i>expr</i>                           | Right         |

5913  
5914  
5915  
5916

Each expression shall have either a string value, a numeric value, or both. Except as stated for specific contexts, the value of an expression shall be implicitly converted to the type needed for the context in which it is used. A string value shall be converted to a numeric value by the equivalent of the following calls to functions defined by the ISO C standard:

5917  
5918

```
setlocale(LC_NUMERIC, "");
numeric_value = atof(string_value);
```

5919  
5920  
5921  
5922  
5923  
5924  
5925  
5926  
5927  
5928

A numeric value that is exactly equal to the value of an integer shall be converted to a string by the equivalent of a call to the **sprintf** function (see **String Functions** (on page 2381)) with the string "%d" as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. Any other numeric value shall be converted to a string by the equivalent of a call to the **sprintf** function with the value of the variable **CONVFMT** as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. The result of the conversion is unspecified if the value of **CONVFMT** is not a floating-point format specification. This volume of IEEE Std. 1003.1-200x specifies no explicit conversions between numbers and strings. An application can force an expression to be treated as a number by adding zero to it, or can force it to be treated as a string by concatenating the null string (" ") to it.

5929

A string value shall be considered a *numeric string* if it comes from one of the following:

5930  
5931  
5932  
5933  
5934  
5935  
5936  
5937

1. Field variables
2. Input from the *getline()* function
3. **FILENAME**
4. **ARGV** array elements
5. **ENVIRON** array elements
6. Array elements created by the *split()* function
7. A command line variable assignment
8. Variable assignment from another numeric string variable

5938  
5939  
5940

and after all the following conversions have been applied, the resulting string would lexically be recognized as a **NUMBER** token as described by the lexical conventions in **Grammar** (on page 2384):

- 5941       • All leading and trailing <blank>s are discarded
- 5942       • If the first non-<blank> character is '+' or '-', it is discarded
- 5943       • Changing each occurrence of the decimal point character from the current locale to a period
- 5944       If a '-' character is ignored in the preceding description, the numeric value of the *numeric string*
- 5945       shall be the negation of the numeric value of the recognized **NUMBER** token. Otherwise, the
- 5946       numeric value of the *numeric string* shall be the numeric value of the recognized **NUMBER**
- 5947       token. Whether or not a string is a *numeric string* shall be relevant only in contexts where that
- 5948       term is used in this section.
- 5949       When an expression is used in a Boolean context, if it has a numeric value, a value of zero shall
- 5950       be treated as false and any other value shall be treated as true. Otherwise, a string value of the
- 5951       null string shall be treated as false and any other value shall be treated as true. A Boolean
- 5952       context shall be one of the following:
- 5953       • The first subexpression of a conditional expression
- 5954       • An expression operated on by logical NOT, logical AND, or logical OR
- 5955       • The second expression of a **for** statement
- 5956       • The expression of an **if** statement
- 5957       • The expression of the **while** clause in either a **while** or **do...while** statement
- 5958       • An expression used as a pattern (as in Overall Program Structure)
- 5959       All arithmetic shall follow the semantics of floating-point arithmetic as specified by the ISO C
- 5960       standard.
- 5961       The value of the expression:
- 5962       `expr1 ^ expr2`
- 5963       shall be equivalent to the value returned by the ISO C standard function call:
- 5964       `pow(expr1, expr2)`
- 5965       The expression:
- 5966       `lvalue ^= expr`
- 5967       shall be equivalent to the ISO C standard expression:
- 5968       `lvalue = pow(lvalue, expr)`
- 5969       except that *lvalue* shall be evaluated only once. The value of the expression:
- 5970       `expr1 % expr2`
- 5971       shall be equivalent to the value returned by the ISO C standard function call:
- 5972       `fmod(expr1, expr2)`
- 5973       The expression:
- 5974       `lvalue %= expr`
- 5975       shall be equivalent to the ISO C standard expression:
- 5976       `lvalue = fmod(lvalue, expr)`
- 5977       except that *lvalue* shall be evaluated only once.



5978 Variables and fields shall be set by the assignment statement:

5979 *lvalue* = *expression*

5980 and the type of *expression* shall determine the resulting variable type. The assignment includes  
 5981 the arithmetic assignments (" += ", " -= ", " \*= ", " /= ", " % = ", " ^ = ", " ++ ", " — ") all of which  
 5982 shall produce a numeric result. The left-hand side of an assignment and the target of increment  
 5983 and decrement operators can be one of a variable, an array with index, or a field selector.

5984 The *awk* language supplies arrays that are used for storing numbers or strings. Arrays need not  
 5985 be declared. They shall initially be empty, and their sizes shall change dynamically. The  
 5986 subscripts, or element identifiers, are strings, providing a type of associative array capability. An  
 5987 array name followed by a subscript within square brackets can be used as an *lvalue* and thus as  
 5988 an expression, as described in the grammar; see **Grammar** (on page 2384). Unsubscripted array  
 5989 names can be used in only the following contexts:

- 5990 • A parameter in a function definition or function call
- 5991 • The **NAME** token following any use of the keyword **in** as specified in the grammar (see  
 5992 **Grammar** (on page 2384)); if the name used in this context is not an array name, the behavior  
 5993 is undefined

5994 A valid array *index* shall consist of one or more comma-separated expressions, similar to the way  
 5995 in which multi-dimensional arrays are indexed in some programming languages. Because *awk*  
 5996 arrays are really one-dimensional, such a comma-separated list shall be converted to a single  
 5997 string by concatenating the string values of the separate expressions, each separated from the  
 5998 other by the value of the **SUBSEP** variable. Thus, the following two index operations shall be  
 5999 equivalent:

6000 *var*[*expr1*, *expr2*, ... *exprn*]

6001 *var*[*expr1* **SUBSEP** *expr2* **SUBSEP** ... **SUBSEP** *exprn*]

6002 The application shall ensure that a multi-dimensioned *index* used with the **in** operator is  
 6003 parenthesized. The **in** operator, which tests for the existence of a particular array element, shall  
 6004 not cause that element to exist. Any other reference to a nonexistent array element shall  
 6005 automatically create it.

6006 Comparisons (with the '<', '<=', '!=', '==', '>', and '>=' operators) shall be made  
 6007 numerically if both operands are numeric, if one is numeric and the other has a string value that  
 6008 is a numeric string, or if one is numeric and the other has the uninitialized value. Otherwise,  
 6009 operands shall be converted to strings as required and a string comparison shall be made using  
 6010 the locale-specific collation sequence. The value of the comparison expression shall be 1 if the  
 6011 relation is true, or 0 if the relation is false.

## 6012 **Variables and Special Variables**

6013 Variables can be used in an *awk* program by referencing them. With the exception of function  
 6014 parameters (see **User-Defined Functions** (on page 2383)), they are not explicitly declared.  
 6015 Function parameter names shall be local to the function; all other variable names shall be global.  
 6016 The same name shall not be used as both a function parameter name and as the name of a  
 6017 function or a special *awk* variable. The same name shall not be used both as a variable name with  
 6018 global scope and as the name of a function. The same name shall not be used within the same  
 6019 scope both as a scalar variable and as an array. Uninitialized variables, including scalar  
 6020 variables, array elements, and field variables, shall have an uninitialized value. An uninitialized  
 6021 value shall have both a numeric value of zero and a string value of the empty string. Evaluation  
 6022 of variables with an uninitialized value, to either string or numeric, shall be determined by the  
 6023 context in which they are used.

6024 Field variables shall be designated by a '\$' followed by a number or numerical expression. The  
6025 effect of the field number *expression* evaluating to anything other than a non-negative integer is  
6026 unspecified; uninitialized variables or string values need not be converted to numeric values in  
6027 this context. New field variables can be created by assigning a value to them. References to  
6028 nonexistent fields (that is, fields after \$NF), shall evaluate to the uninitialized value. Such  
6029 references shall not create new fields. However, assigning to a nonexistent field (for example,  
6030 \$(NF+2)=5) shall increase the value of NF; create any intervening fields with the uninitialized  
6031 value; and cause the value of \$0 to be recomputed, with the fields being separated by the value  
6032 of OFS. Each field variable shall have a string value or an uninitialized value when created.  
6033 Field variables shall have the uninitialized value when created from \$0 using FS and the variable  
6034 does not contain any characters. If appropriate, the field variable shall be considered a numeric  
6035 string (see **Expressions in awk** (on page 2370)).

6036 Implementations shall support the following other special variables that are set by *awk*:

6037 **ARGC** The number of elements in the **ARGV** array.

6038 **ARGV** An array of command line arguments, excluding options and the *program*  
6039 argument, numbered from zero to **ARGC**-1.

6040 The arguments in **ARGV** can be modified or added to; **ARGC** can be altered. As  
6041 each input file ends, *awk* shall treat the next non-null element of **ARGV**, up to the  
6042 current value of **ARGC**-1, inclusive, as the name of the next input file. Thus,  
6043 setting an element of **ARGV** to null means that it shall not be treated as an input  
6044 file. The name '-' indicates the standard input. If an argument matches the  
6045 format of an *assignment* operand, this argument shall be treated as an assignment  
6046 rather than a *file* argument.

6047 **CONVFMT** The **printf** format for converting numbers to strings (except for output statements,  
6048 where **OFMT** is used); "% . 6g" by default.

6049 **ENVIRON** The variable **ENVIRON** is an array representing the value of the environment, as  
6050 described in the *exec* functions defined in the System Interfaces volume of  
6051 IEEE Std. 1003.1-200x. The indices of the array shall be strings consisting of the  
6052 names of the environment variables, and the value of each array element is a string  
6053 consisting of the value of that variable. If appropriate, the environment variable  
6054 shall be considered a *numeric string* (see **Expressions in awk** (on page 2370)), the  
6055 array element shall also have its numeric value.

6056 In all cases where the behavior of *awk* is affected by environment variables  
6057 (including the environment of any commands that *awk* executes via the **system**  
6058 function or via pipeline redirections with the **print** statement, the **printf** statement,  
6059 or the **getline** function), the environment used shall be the environment at the time  
6060 *awk* began executing; it is implementation-defined whether any modification of  
6061 **ENVIRON** affects this environment.

6062 **FILENAME** A path name of the current input file. Inside a **BEGIN** action the value is  
6063 undefined. Inside an **END** action the value is the name of the last input file  
6064 processed.

6065 **FNR** The ordinal number of the current record in the current file. Inside a **BEGIN** action  
6066 the value is zero. Inside an **END** action the value is the number of the last record  
6067 processed in the last file processed.

6068 **FS** Input field separator regular expression; a <space> character by default.

6069 **NF** The number of fields in the current record. Inside a **BEGIN** action, the use of **NF** is  
6070 undefined unless a **getline** function without a *var* argument is executed

6071 previously. Inside an **END** action, **NF** retains the value it had for the last record  
6072 read, unless a subsequent redirected, **getline** function without a *var* argument is  
6073 performed prior to entering the **END** action.

6074 **NR** The ordinal number of the current record from the start of input. Inside a **BEGIN**  
6075 action the value is zero. Inside an **END** action the value is the number of the last  
6076 record processed.

6077 **OFMT** The **printf** format for converting numbers to strings in output statements (see  
6078 **Output Statements** (on page 2379)); "%.*g*" by default. The result of the  
6079 conversion is unspecified if the value of **OFMT** is not a floating-point format  
6080 specification.

6081 **OFS** The **print** statement output field separation; <space> by default.

6082 **ORS** The **print** statement output record separator; a <newline> character by default.

6083 **RLENGTH** The length of the string matched by the **match** function.

6084 **RS** The first character of the string value of **RS** is the input record separator; a  
6085 <newline> character by default. If **RS** contains more than one character, the results  
6086 are unspecified. If **RS** is null, then records are separated by sequences of one or  
6087 more blank lines, leading or trailing blank lines do not result in empty records at  
6088 the beginning or end of the input, and a <newline> character is always a field  
6089 separator, no matter what the value of **FS** is.

6090 **RSTART** The starting position of the string matched by the **match** function, numbering from  
6091 1. This is always equivalent to the return value of the **match** function.

6092 **SUBSEP** The subscript separator string for multi-dimensional arrays; the default value is  
6093 implementation-defined.

6094 **Regular Expressions**

6095 The *awk* utility shall make use of the extended regular expression notation (see the Base  
6096 Definitions volume of IEEE Std. 1003.1-200x, Section 9.4, Extended Regular Expressions) except  
6097 that it shall allow the use of C-language conventions for escaping special characters within the  
6098 EREs, as specified in the table in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 5,  
6099 File Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v') and the following  
6100 table; these escape sequences shall be recognized both inside and outside bracket expressions.  
6101 Note that records need not be separated by <newline> characters and string constants can  
6102 contain <newline> characters, so even the "\n" sequence is valid in *awk* EREs. Using a slash  
6103 character within an ERE requires the escaping shown in the following table.

6104

Table 4-2 Escape Sequences in *awk*

6105

6106

6107

6108

6109

6110

6111

6112

6113

6114

6115

6116

6117

6118

6119

6120

6121

6122

6123

6124

6125

| Escape Sequence | Description                                                                                                                                                                                                                           | Meaning                                                                                                                                                                                                                                                                                                                                                               |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \ "             | Backslash quotation-mark                                                                                                                                                                                                              | Quotation-mark character                                                                                                                                                                                                                                                                                                                                              |
| \ /             | Backslash slash                                                                                                                                                                                                                       | Slash character                                                                                                                                                                                                                                                                                                                                                       |
| \ ddd           | A backslash character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.                 | The character whose encoding is represented by the one, two, or three-digit octal integer. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is implementation-defined. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading '\ ' for each byte. |
| \ c             | A backslash character followed by any character not described in this table or in the table in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v') | Undefined                                                                                                                                                                                                                                                                                                                                                             |

6126

6127

6128

6129

6130

6131

6132

6133

6134

6135

6136

6137

6138

6139

6140

A regular expression can be matched against a specific field or string by using one of the two regular expression matching operators, '~' and '!~'. These operators shall interpret their right-hand operand as a regular expression and their left-hand operand as a string. If the regular expression matches the string, the '~' expression shall evaluate to a value of 1, and the '!~' expression shall evaluate to a value of 0. (The regular expression matching operation is as defined by the term *matched* in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 9.1, Regular Expression Definitions, where a match occurs on any part of the string unless the regular expression is limited with the circumflex or dollar sign special characters.) If the regular expression does not match the string, the '~' expression shall evaluate to a value of 0, and the '!~' expression shall evaluate to a value of 1. If the right-hand operand is any expression other than the lexical token **ERE**, the string value of the expression shall be interpreted as an extended regular expression, including the escape conventions described above. Note that these same escape conventions shall also be applied in determining the value of a string literal (the lexical token **STRING**), and thus shall be applied a second time when a string literal is used in this context.

6141

6142

6143

When an **ERE** token appears as an expression in any context other than as the right-hand of the '~' or '!~' operator or as one of the built-in function arguments described below, the value of the resulting expression shall be the equivalent of:

6144

```
$0 ~ /ere/
```

6145

6146

6147

6148

The *ere* argument to the **gsub**, **match**, **sub** functions, and the *fs* argument to the **split** function (see **String Functions** (on page 2381)) shall be interpreted as extended regular expressions. These can be either **ERE** tokens or arbitrary expressions, and shall be interpreted in the same manner as the right-hand side of the '~' or '!~' operator.

6149

6150

6151

An extended regular expression can be used to separate fields by using the **-F ERE** option or by assigning a string containing the expression to the built-in variable **FS**. The default value of the **FS** variable shall be a single <space> character. The following describes **FS** behavior:

- 6152 1. If **FS** is a null string, the behavior is unspecified.
- 6153 2. If **FS** is a single character:
- 6154 a. If **FS** is the <space> character, skip leading and trailing <blank> characters; fields
- 6155 shall be delimited by sets of one or more <blank> characters.
- 6156 b. Otherwise, if **FS** is any other character *c*, fields shall be delimited by each single
- 6157 occurrence of *c*.
- 6158 3. Otherwise, the string value of **FS** shall be considered to be an extended regular expression.
- 6159 Each occurrence of a sequence matching the extended regular expression shall delimit
- 6160 fields.

6161 Except for the '*~*' and '!~' operators, and in the **gsub**, **match**, **split**, and **sub** built-in functions,

6162 ERE matching shall be based on input records; that is, record separator characters (the first

6163 character of the value of the variable **RS**, <newline> by default) cannot be embedded in the

6164 expression, and no expression shall match the record separator character. If the record separator

6165 is not <newline>, <newline> characters embedded in the expression can be matched. For the

6166 '*~*' and '!~' operators, and in those four built-in functions, ERE matching shall be based on

6167 text strings; that is, any character (including <newline> and the record separator) can be

6168 embedded in the pattern, and an appropriate pattern shall match any character. However, in all

6169 *awk* ERE matching, the use of one or more NUL characters in the pattern, input record, or text

6170 string produces undefined results.

## 6171 **Patterns**

6172 A *pattern* is any valid *expression*, a range specified by two expressions separated by comma, or

6173 one of the two special patterns **BEGIN** or **END**.

## 6174 **Special Patterns**

6175 The *awk* utility shall recognize two special patterns, **BEGIN** and **END**. Each **BEGIN** pattern

6176 shall be matched once and its associated action executed before the first record of input is read

6177 (except possibly by use of the **getline** function—see **Input/Output and General Functions** (on

6178 page 2382)—in a prior **BEGIN** action) and before command line assignment is done. Each **END**

6179 pattern shall be matched once and its associated action executed after the last record of input has

6180 been read. These two patterns shall have associated actions.

6181 **BEGIN** and **END** shall not combine with other patterns. Multiple **BEGIN** and **END** patterns

6182 shall be allowed. The actions associated with the **BEGIN** patterns shall be executed in the order

6183 specified in the program, as are the **END** actions. An **END** pattern can precede a **BEGIN** pattern

6184 in a program.

6185 If an *awk* program consists of only actions with the pattern **BEGIN**, and the **BEGIN** action

6186 contains no **getline** function, *awk* shall exit without reading its input when the last statement in

6187 the last **BEGIN** action is executed. If an *awk* program consists of only actions with the pattern

6188 **END** or only actions with the patterns **BEGIN** and **END**, the input shall be read before the

6189 statements in the **END** actions are executed.

6190 **Expression Patterns**

6191 An expression pattern shall be evaluated as if it were an expression in a Boolean context. If the  
6192 result is true, the pattern shall be considered to match, and the associated action (if any) shall be  
6193 executed. If the result is false, the action shall not be executed.

6194 **Pattern Ranges**

6195 A pattern range consists of two expressions separated by a comma; in this case, the action shall  
6196 be performed for all records between a match of the first expression and the following match of  
6197 the second expression, inclusive. At this point, the pattern range can be repeated starting at  
6198 input records subsequent to the end of the matched range.

6199 **Actions**

6200 An action is a sequence of statements as shown in the grammar in **Grammar** (on page 2384).  
6201 Any single statement can be replaced by a statement list enclosed in braces. The application shall  
6202 ensure that statements in a statement list are separated by <newline> characters or semicolons,  
6203 and are executed sequentially in the order that they appear.

6204 The *expression* acting as the conditional in an **if** statement shall be evaluated and if it is non-zero  
6205 or non-null, the following *statement* shall be executed; otherwise, if **else** is present, the statement  
6206 following the **else** shall be executed.

6207 The **if**, **while**, **do...while**, **for**, **break**, and **continue** statements are based on the ISO C standard,  
6208 except that the Boolean expressions shall be treated as described in **Expressions in awk** (on page  
6209 2370), and except in the case of:

6210 `for (variable in array)`

6211 which shall iterate, assigning each *index of array* to *variable* in an unspecified order. The results of  
6212 adding new elements to *array* within such a **for** loop are undefined. If a **break** or **continue**  
6213 statement occurs outside of a loop, the behavior is undefined.

6214 The **delete** statement shall remove an individual array element. Thus, the following code deletes  
6215 an entire array:

```
6216 for (index in array)
6217 delete array[index]
```

6218 The **next** statement shall cause all further processing of the current input record to be  
6219 abandoned. The behavior is undefined if a **next** statement appears or is invoked in a **BEGIN** or  
6220 **END** action.

6221 The **exit** statement shall invoke all **END** actions in the order in which they occur in the program  
6222 source and then terminate the program without reading further input. An **exit** statement inside  
6223 an **END** action shall terminate the program without further execution of **END** actions. If an  
6224 expression is specified in an **exit** statement, its numeric value shall be the exit status of *awk*,  
6225 unless subsequent errors are encountered or a subsequent **exit** statement with an expression is  
6226 executed.

6227 **Output Statements**

6228 Both **print** and **printf** statements shall write to standard output by default. The output shall be  
 6229 written to the location specified by *output\_redirection* if one is supplied, as follows:

```
6230 > expression
6231 >> expression
6232 | expression
```

6233 In all cases, the *expression* shall be evaluated to produce a string that is used as a path name into  
 6234 which to write (for '*>*' or "*>>*") or as a command to be executed (for '*|*'). Using the first two  
 6235 forms, if the file of that name is not currently open, it shall be opened, creating it if necessary and  
 6236 using the first form, truncating the file. The output then shall be appended to the file. As long as  
 6237 the file remains open, subsequent calls in which *expression* evaluates to the same string value  
 6238 shall simply append output to the file. The file remains open until the **close** function (see  
 6239 **Input/Output and General Functions** (on page 2382)) is called with an expression that evaluates  
 6240 to the same string value.

6241 The third form shall write output onto a stream piped to the input of a command. The stream  
 6242 shall be created if no stream is currently open with the value of *expression* as its command name.  
 6243 The stream created shall be equivalent to one created by a call to the *popen()* function defined in  
 6244 the System Interfaces volume of IEEE Std. 1003.1-200x with the value of *expression* as the  
 6245 *command* argument and a value of *w* as the *mode* argument. As long as the stream remains open,  
 6246 subsequent calls in which *expression* evaluates to the same string value shall write output to the  
 6247 existing stream. The stream shall remain open until the **close** function (see **Input/Output and**  
 6248 **General Functions** (on page 2382)) is called with an expression that evaluates to the same string  
 6249 value. At that time, the stream shall be closed as if by a call to the *pclose()* function defined in  
 6250 the System Interfaces volume of IEEE Std. 1003.1-200x.

6251 As described in detail by the grammar in **Grammar** (on page 2384), these output statements shall  
 6252 take a comma-separated list of *expressions* referred to in the grammar by the non-terminal  
 6253 symbols **expr\_list**, **print\_expr\_list**, or **print\_expr\_list\_opt**. This list is referred to here as the  
 6254 *expression list*, and each member is referred to as an *expression argument*.

6255 The **print** statement shall write the value of each expression argument onto the indicated output  
 6256 stream separated by the current output field separator (see variable **OFS** above), and terminated  
 6257 by the output record separator (see variable **ORS** above). All expression arguments shall be  
 6258 taken as strings, being converted if necessary; this conversion shall be as described in  
 6259 **Expressions in awk** (on page 2370), with the exception that the **printf** format in **OFMT** shall be  
 6260 used instead of the value in **CONVFMT**. An empty expression list shall stand for the whole  
 6261 input record (\$0).

6262 The **printf** statement shall produce output based on a notation similar to the File Format  
 6263 Notation used to describe file formats in this volume of IEEE Std. 1003.1-200x (see the Base  
 6264 Definitions volume of IEEE Std. 1003.1-200x, Chapter 5, File Format Notation). Output shall be  
 6265 produced as specified with the first expression argument as the string *format* and subsequent  
 6266 expression arguments as the strings *arg1* to *argn*, inclusive, with the following exceptions:

- 6267 1. The *format* shall be an actual character string rather than a graphical representation.  
 6268 Therefore, it cannot contain empty character positions. The *<space>* character in the *format*  
 6269 string, in any context other than a *flag* of a conversion specification, shall be treated as an  
 6270 ordinary character that is copied to the output.
- 6271 2. If the character set contains a '*Δ*' character and that character appears in the *format* string,  
 6272 it shall be treated as an ordinary character that is copied to the output.

- 6273 3. The *escape sequences* beginning with a backslash character shall be treated as sequences of  
 6274 ordinary characters that are copied to the output. Note that these same sequences shall be  
 6275 interpreted lexically by *awk* when they appear in literal strings, but they shall not be  
 6276 treated specially by the **printf** statement.
- 6277 4. A *field width* or *precision* can be specified as the '\*' character instead of a digit string. In  
 6278 this case the next argument from the expression list shall be fetched and its numeric value  
 6279 taken as the field width or precision.
- 6280 5. The implementation shall not precede or follow output from the *d* or *u* conversion  
 6281 specifications with <blank> characters not specified by the *format* string.
- 6282 6. The implementation shall not precede output from the *o* conversion specification with  
 6283 leading zeros not specified by the *format* string.
- 6284 7. For the *c* conversion specification: if the argument has a numeric value, the character  
 6285 whose encoding is that value shall be output. If the value is zero or is not the encoding of  
 6286 any character in the character set, the behavior is undefined. If the argument does not have  
 6287 a numeric value, the first character of the string value shall be output; if the string does not  
 6288 contain any characters, the behavior is undefined.
- 6289 8. For each conversion specification that consumes an argument, the next expression  
 6290 argument shall be evaluated. With the exception of the *c* conversion, the value shall be  
 6291 converted (according to the rules specified in **Expressions in awk** (on page 2370)) to the  
 6292 appropriate type for the conversion specification.
- 6293 9. If there are insufficient expression arguments to satisfy all the conversion specifications in  
 6294 the *format* string, the behavior is undefined.
- 6295 10. If any character sequence in the *format* string begins with a '%' character, but does not  
 6296 form a valid conversion specification, the behavior is unspecified.

6297 Both **print** and **printf** can output at least {LINE\_MAX} bytes.

## 6298 **Functions**

6299 The *awk* language has a variety of built-in functions: arithmetic, string, input/output, and  
 6300 general.

### 6301 **Arithmetic Functions**

6302 The arithmetic functions, except for **int**, shall be based on the ISO C standard. The behavior is  
 6303 undefined in cases where the ISO C standard specifies that an error be returned or that the  
 6304 behavior is undefined. Although the grammar (see **Grammar** (on page 2384)) permits built-in  
 6305 functions to appear with no arguments or parentheses, unless the argument or parentheses are  
 6306 indicated as optional in the following list (by displaying them within the "[ ]" brackets), such  
 6307 use is undefined.

- 6308 **atan2**(*y,x*) Return arctangent of  $y/x$  in radians in the range  $-\{\pi\}$  to  $\{\}$ .
- 6309 **cos**(*x*) Return cosine of *x*, where *x* is in radians.
- 6310 **sin**(*x*) Return sine of *x*, where *x* is in radians.
- 6311 **exp**(*x*) Return the exponential function of *x*.
- 6312 **log**(*x*) Return the natural logarithm of *x*.
- 6313 **sqrt**(*x*) Return the square root of *x*.



- 6314        **int**(*x*)        Truncate its argument to an integer. It shall be truncated toward 0 when  $x > 0$ .
- 6315        **rand**()        Return a random number  $n$ , such that  $0 \leq n < 1$ .
- 6316        **srand**([*expr*]) Set the seed value for *rand* to *expr* or use the time of day if *expr* is omitted. The  
6317        previous seed value shall be returned.

### 6318        **String Functions**

6319        The string functions in the following list shall be supported. Although the grammar (see  
6320        **Grammar** (on page 2384)) permits built-in functions to appear with no arguments or  
6321        parentheses, unless the argument or parentheses are indicated as optional in the following list  
6322        (by displaying them within the "[ ]" brackets), such use is undefined.

#### 6323        **gsub**(*ere*, *repl*[, *in*])

6324        Behave like **sub** (see below), except that it shall replace all occurrences of the  
6325        regular expression (like the *ed* utility global substitute) in \$0 or in the *in* argument,  
6326        when specified.

6327        **index**(*s*, *t*)    Return the position, in characters, numbering from 1, in string *s* where string *t* first  
6328        occurs, or zero if it does not occur at all.

6329        **length**([*(s)*]) Return the length, in characters, of its argument taken as a string, or of the whole  
6330        record, \$0, if there is no argument.

6331        **match**(*s*, *ere*) Return the position, in characters, numbering from 1, in string *s* where the  
6332        extended regular expression *ere* occurs, or zero if it does not occur at all. RSTART  
6333        shall be set to the starting position (which is the same as the returned value), zero  
6334        if no match is found; RLENGTH shall be set to the length of the matched string, -1  
6335        if no match is found.

#### 6336        **split**(*s*, *a*[, *fs* ])

6337        Split the string *s* into array elements *a*[1], *a*[2], ..., *a*[*n*], and return *n*. All elements  
6338        of the array shall be deleted before the split is performed. The separation shall be  
6339        done with the ERE *fs* or with the field separator **FS** if *fs* is not given. Each array  
6340        element shall have a string value when created and, if appropriate, the array  
6341        element shall be considered a numeric string (see **Expressions in awk** (on page  
6342        2370)). The effect of a null string as the value of *fs* is unspecified.

#### 6343        **sprintf**(*fmt*, *expr*, *expr*, ...)

6344        Format the expressions according to the **printf** format given by *fmt* and return the  
6345        resulting string.

#### 6346        **sub**(*ere*, *repl*[, *in* ])

6347        Substitute the string *repl* in place of the first instance of the extended regular  
6348        expression *ERE* in string *in* and return the number of substitutions. An ampersand  
6349        ('&') appearing in the string *repl* shall be replaced by the string from *in* that  
6350        matches the ERE. An ampersand preceded with a backslash ('\&') shall be  
6351        interpreted as the literal ampersand character. Any other occurrence of a backslash  
6352        (for example, preceding any other character) shall be treated as a literal backslash  
6353        character. Note that if *repl* is a string literal (the lexical token **STRING**; see  
6354        **Grammar** (on page 2384)), the handling of the ampersand character occurs after  
6355        any lexical processing, including any lexical backslash escape sequence processing.  
6356        If *in* is specified and it is not an *lvalue* (see **Expressions in awk** (on page 2370)), the  
6357        behavior is undefined. If *in* is omitted, *awk* shall use the current record (\$0) in its  
6358        place.

- 6359       **substr**(*s*, *m* [, *n* ] )  
 6360           Return the at most *n*-character substring of *s* that begins at position *m*, numbering  
 6361           from 1. If *n* is missing, or if *n* specifies more characters than are left in the string,  
 6362           the length of the substring shall be limited by the length of the string *s*.
- 6363       **tolower**(*s*)    Return a string based on the string *s*. Each character in *s* that is an uppercase letter  
 6364           specified to have a **tolower** mapping by the *LC\_CTYPE* category of the current  
 6365           locale shall be replaced in the returned string by the lowercase letter specified by  
 6366           the mapping. Other characters in *s* shall be unchanged in the returned string.
- 6367       **toupper**(*s*)    Return a string based on the string *s*. Each character in *s* that is a lowercase letter  
 6368           specified to have a **toupper** mapping by the *LC\_CTYPE* category of the current  
 6369           locale is replaced in the returned string by the uppercase letter specified by the  
 6370           mapping. Other characters in *s* are unchanged in the returned string.

6371       All of the preceding functions that take *ERE* as a parameter expect a pattern or a string valued  
 6372       expression that is a regular expression as defined in **Regular Expressions** (on page 2375).

### 6373       **Input/Output and General Functions**

6374       The input/output and general functions are:

- 6375       **close**(*expression*)  
 6376           Close the file or pipe opened by a **print** or **printf** statement or a call to **getline** with  
 6377           the same string-valued *expression*. The limit on the number of open *expression*  
 6378           arguments is implementation-defined. If the close was successful, the function  
 6379           shall return zero; otherwise, it shall return non-zero.

- 6380       *expression* / **getline** [*var*]  
 6381           Read a record of input from a stream piped from the output of a command. The  
 6382           stream shall be created if no stream is currently open with the value of *expression* as  
 6383           its command name. The stream created shall be equivalent to one created by a call  
 6384           to the *popen*() function with the value of *expression* as the *command* argument and a  
 6385           value of *r* as the *mode* argument. As long as the stream remains open, subsequent  
 6386           calls in which *expression* evaluates to the same string value shall read subsequent  
 6387           records from the stream. The stream shall remain open until the **close** function is  
 6388           called with an *expression* that evaluates to the same string value. At that time, the  
 6389           stream shall be closed as if by a call to the *pclose*() function. If *var* is missing, \$0 and  
 6390           **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered  
 6391           a numeric string (see **Expressions in awk** (on page 2370)).

6392       The **getline** operator can form ambiguous constructs when there are  
 6393       unparenthesized operators (including concatenate) to the left of the ' | ' (to the  
 6394       beginning of the expression containing **getline**). In the context of the '\$'  
 6395       operator, ' | ' shall behave as if it had a lower precedence than '\$'. The result of  
 6396       evaluating other operators is unspecified, and portable applications shall  
 6397       parenthesize properly all such usages.

- 6398       **getline**       Set \$0 to the next input record from the current input file. This form of **getline** shall  
 6399       set the **NF**, **NR**, and **FNR** variables.

- 6400       **getline var**    Set variable *var* to the next input record from the current input file and, if  
 6401       appropriate, *var* shall be considered a numeric string (see **Expressions in awk** (on  
 6402       page 2370)). This form of **getline** shall set the **FNR** and **NR** variables.

- 6403       **getline** [*var*] < *expression*  
 6404           Read the next record of input from a named file. The *expression* shall be evaluated

6405 to produce a string that is used as a path name. If the file of that name is not  
 6406 currently open, it shall be opened. As long as the stream remains open, subsequent  
 6407 calls in which *expression* evaluates to the same string value shall read subsequent  
 6408 records from the file. The file shall remain open until the **close** function is called  
 6409 with an expression that evaluates to the same string value. If *var* is missing, \$0 and  
 6410 **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered  
 6411 a numeric string (see **Expressions in awk** (on page 2370)).

6412 The **getline** operator can form ambiguous constructs when there are  
 6413 unparenthesized binary operators (including concatenate) to the right of the '<'  
 6414 (up to the end of the expression containing the **getline**). The result of evaluating  
 6415 such a construct is unspecified, and portable applications shall parenthesize  
 6416 properly all such usages.

6417 **system**(*expression*)

6418 Execute the command given by *expression* in a manner equivalent to the *system*()  
 6419 function defined in the System Interfaces volume of IEEE Std. 1003.1-200x and  
 6420 return the exit status of the command.

6421 All forms of **getline** shall return 1 for successful input, zero for end-of-file, and -1 for an error.

6422 Where strings are used as the name of a file or pipeline, the application shall ensure that the  
 6423 strings are textually identical. The terminology “same string value” implies that “equivalent  
 6424 strings”, even those that differ only by <space> characters, represent different files.

## 6425 User-Defined Functions

6426 The *awk* language also provides user-defined functions. Such functions can be defined as:

```
6427 function name([parameter, ...]) { statements }
```

6428 A function can be referred to anywhere in an *awk* program; in particular, its use can precede its  
 6429 definition. The scope of a function is global.

6430 Function parameters, if present, can be either scalars or arrays; the behavior is undefined if an  
 6431 array name is passed as a parameter that the function uses as a scalar, or if a scalar expression is  
 6432 passed as a parameter that the function uses as an array. Function parameters shall be passed by  
 6433 value if scalar and by reference if array name.

6434 The number of parameters in the function definition need not match the number of parameters  
 6435 in the function call. Excess formal parameters can be used as local variables. If fewer arguments  
 6436 are supplied in a function call than are in the function definition, the extra parameters that are  
 6437 used in the function body as scalars shall evaluate to the uninitialized value until they are  
 6438 otherwise initialized, and the extra parameters that are used in the function body as arrays shall  
 6439 be treated as uninitialized arrays where each element evaluates to the uninitialized value until  
 6440 otherwise initialized.

6441 When invoking a function, no white space can be placed between the function name and the  
 6442 opening parenthesis. Function calls can be nested and recursive calls can be made upon  
 6443 functions. Upon return from any nested or recursive function call, the values of all of the calling  
 6444 function's parameters shall be unchanged, except for array parameters passed by reference. The  
 6445 **return** statement can be used to return a value. If a **return** statement appears outside of a  
 6446 function definition, the behavior is undefined.

6447 In the function definition, <newline> characters shall be optional before the opening brace and  
 6448 after the closing brace. Function definitions can appear anywhere in the program where a  
 6449 *pattern-action* pair is allowed.

```

6450 Grammar
6451 The grammar in this section and the lexical conventions in the following section shall together
6452 describe the syntax for awk programs. The general conventions for this style of grammar are
6453 described in Section 1.10 (on page 2223). A valid program can be represented as the non-
6454 terminal symbol program in the grammar. This formal syntax shall take precedence over the
6455 preceding text syntax description.
6456 %token NAME NUMBER STRING ERE
6457 %token FUNC_NAME /* Name followed by '(' without white space. */
6458 /* Keywords */
6459 %token Begin End
6460 /* 'BEGIN' 'END' */
6461 %token Break Continue Delete Do Else
6462 /* 'break' 'continue' 'delete' 'do' 'else' */
6463 %token Exit For Function If In
6464 /* 'exit' 'for' 'function' 'if' 'in' */
6465 %token Next Print Printf Return While
6466 /* 'next' 'print' 'printf' 'return' 'while' */
6467 /* Reserved function names */
6468 %token BUILTIN_FUNC_NAME
6469 /* One token for the following:
6470 * atan2 cos sin exp log sqrt int rand srand
6471 * gsub index length match split sprintf sub
6472 * substr tolower toupper close system
6473 */
6474 %token GETLINE
6475 /* Syntactically different from other built-ins. */
6476 /* Two-character tokens. */
6477 %token ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN POW_ASSIGN
6478 /* '+=' '-=' '*=' '/=' '%=' '^=' */
6479 %token OR AND NO_MATCH EQ LE GE NE INCR DECR APPEND
6480 /* '|'| '&&' '!~' '==' '<=' '>=' '!=' '++' '—' '>>' */
6481 /* One-character tokens. */
6482 %token '{' '}' '(' ')' '[' ']' ',' ';' NEWLINE
6483 %token '+' '-' '*' '%' '^' '!' '>' '<' '|' '?' ':' '~' '$' '='
6484 %start program
6485 %%
6486 program : item_list
6487 | actionless_item_list
6488 ;
6489 item_list : newline_opt
6490 | actionless_item_list item terminator
6491 | item_list item terminator
6492 | item_list action terminator
6493 ;

```

```

6494 actionless_item_list : item_list pattern terminator
6495 | actionless_item_list pattern terminator
6496 ;

6497 item : pattern action
6498 | Function NAME '(' param_list_opt ')'
6499 | newline_opt action
6500 | Function FUNC_NAME '(' param_list_opt ')'
6501 | newline_opt action
6502 ;

6503 param_list_opt : /* empty */
6504 | param_list
6505 ;

6506 param_list : NAME
6507 | param_list ',' NAME
6508 ;

6509 pattern : Begin
6510 | End
6511 | expr
6512 | expr ',' newline_opt expr
6513 ;

6514 action : '{' newline_opt '}'
6515 | '{' newline_opt terminated_statement_list '}'
6516 | '{' newline_opt unterminated_statement_list '}'
6517 ;

6518 terminator : terminator ';'
6519 | terminator NEWLINE
6520 | ';'
6521 | NEWLINE
6522 ;

6523 terminated_statement_list : terminated_statement
6524 | terminated_statement_list terminated_statement
6525 ;

6526 unterminated_statement_list : unterminated_statement
6527 | terminated_statement_list unterminated_statement
6528 ;

6529 terminated_statement : action newline_opt
6530 | If '(' expr ')' newline_opt terminated_statement
6531 | If '(' expr ')' newline_opt terminated_statement
6532 | Else newline_opt terminated_statement
6533 | While '(' expr ')' newline_opt terminated_statement
6534 | For '(' simple_statement_opt ';'
6535 | expr_opt ';' simple_statement_opt ')' newline_opt
6536 | terminated_statement
6537 | For '(' NAME In NAME ')' newline_opt
6538 | terminated_statement
6539 | ';' newline_opt
6540 | terminatable_statement NEWLINE newline_opt
6541 | terminatable_statement ';' newline_opt

```

```

6542 ;
6543 unterminated_statement : terminatable_statement
6544 | If '(' expr ')' newline_opt unterminated_statement
6545 | If '(' expr ')' newline_opt terminated_statement
6546 | Else newline_opt unterminated_statement
6547 | While '(' expr ')' newline_opt unterminated_statement
6548 | For '(' simple_statement_opt ';'
6549 | expr_opt ';' simple_statement_opt ')' newline_opt
6550 | unterminated_statement
6551 | For '(' NAME In NAME ')' newline_opt
6552 | unterminated_statement
6553 ;
6554 terminatable_statement : simple_statement
6555 | Break
6556 | Continue
6557 | Next
6558 | Exit expr_opt
6559 | Return expr_opt
6560 | Do newline_opt terminated_statement While '(' expr ')'
6561 ;
6562 simple_statement_opt : /* empty */
6563 | simple_statement
6564 ;
6565 simple_statement : Delete NAME '[' expr_list ']'
6566 | expr
6567 | print_statement
6568 ;
6569 print_statement : simple_print_statement
6570 | simple_print_statement output_redirection
6571 ;
6572 simple_print_statement : Print print_expr_list_opt
6573 | Print '(' multiple_expr_list ')'
6574 | Printf print_expr_list
6575 | Printf '(' multiple_expr_list ')'
6576 ;
6577 output_redirection : '>' expr
6578 | APPEND expr
6579 | '|' expr
6580 ;
6581 expr_list_opt : /* empty */
6582 | expr_list
6583 ;
6584 expr_list : expr
6585 | multiple_expr_list
6586 ;
6587 multiple_expr_list : expr ',' newline_opt expr
6588 | multiple_expr_list ',' newline_opt expr

```

```

6589 ;
6590 expr_opt : /* empty */
6591 | expr
6592 ;
6593 expr : unary_expr
6594 | non_unary_expr
6595 ;
6596 unary_expr : '+' expr
6597 | '-' expr
6598 | unary_expr '^' expr
6599 | unary_expr '*' expr
6600 | unary_expr '/' expr
6601 | unary_expr '%' expr
6602 | unary_expr '+' expr
6603 | unary_expr '-' expr
6604 | unary_expr non_unary_expr
6605 | unary_expr '<' expr
6606 | unary_expr LE expr
6607 | unary_expr NE expr
6608 | unary_expr EQ expr
6609 | unary_expr '>' expr
6610 | unary_expr GE expr
6611 | unary_expr '~' expr
6612 | unary_expr NO_MATCH expr
6613 | unary_expr In NAME
6614 | unary_expr AND newline_opt expr
6615 | unary_expr OR newline_opt expr
6616 | unary_expr '?' expr ':' expr
6617 | unary_input_function
6618 ;
6619 non_unary_expr : '(' expr ')'
6620 | '!' expr
6621 | non_unary_expr '^' expr
6622 | non_unary_expr '*' expr
6623 | non_unary_expr '/' expr
6624 | non_unary_expr '%' expr
6625 | non_unary_expr '+' expr
6626 | non_unary_expr '-' expr
6627 | non_unary_expr non_unary_expr
6628 | non_unary_expr '<' expr
6629 | non_unary_expr LE expr
6630 | non_unary_expr NE expr
6631 | non_unary_expr EQ expr
6632 | non_unary_expr '>' expr
6633 | non_unary_expr GE expr
6634 | non_unary_expr '~' expr
6635 | non_unary_expr NO_MATCH expr
6636 | non_unary_expr In NAME
6637 | '(' multiple_expr_list ')' In NAME
6638 | non_unary_expr AND newline_opt expr

```

```

6639 | non_unary_expr OR newline_opt expr
6640 | non_unary_expr '?' expr ':' expr
6641 | NUMBER
6642 | STRING
6643 | lvalue
6644 | ERE
6645 | lvalue INCR
6646 | lvalue DECR
6647 | INCR lvalue
6648 | DECR lvalue
6649 | lvalue POW_ASSIGN expr
6650 | lvalue MOD_ASSIGN expr
6651 | lvalue MUL_ASSIGN expr
6652 | lvalue DIV_ASSIGN expr
6653 | lvalue ADD_ASSIGN expr
6654 | lvalue SUB_ASSIGN expr
6655 | lvalue '=' expr
6656 | FUNC_NAME '(' expr_list_opt ')'
6657 | /* no white space allowed before '(' */
6658 | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
6659 | BUILTIN_FUNC_NAME
6660 | non_unary_input_function
6661 ;

6662 print_expr_list_opt : /* empty */
6663 | print_expr_list
6664 ;

6665 print_expr_list : print_expr
6666 | print_expr_list ',' newline_opt print_expr
6667 ;

6668 print_expr : unary_print_expr
6669 | non_unary_print_expr
6670 ;

6671 unary_print_expr : '+' print_expr
6672 | '-' print_expr
6673 | unary_print_expr '^' print_expr
6674 | unary_print_expr '*' print_expr
6675 | unary_print_expr '/' print_expr
6676 | unary_print_expr '%' print_expr
6677 | unary_print_expr '+' print_expr
6678 | unary_print_expr '-' print_expr
6679 | unary_print_expr non_unary_print_expr
6680 | unary_print_expr '~' print_expr
6681 | unary_print_expr NO_MATCH print_expr
6682 | unary_print_expr In NAME
6683 | unary_print_expr AND newline_opt print_expr
6684 | unary_print_expr OR newline_opt print_expr
6685 | unary_print_expr '?' print_expr ':' print_expr
6686 ;

6687 non_unary_print_expr : '(' expr ')'
6688 | '!' print_expr

```



```

6689 | non_unary_print_expr '^' print_expr
6690 | non_unary_print_expr '*' print_expr
6691 | non_unary_print_expr '/' print_expr
6692 | non_unary_print_expr '%' print_expr
6693 | non_unary_print_expr '+' print_expr
6694 | non_unary_print_expr '-' print_expr
6695 | non_unary_print_expr non_unary_print_expr
6696 | non_unary_print_expr '~' print_expr
6697 | non_unary_print_expr NO_MATCH print_expr
6698 | non_unary_print_expr In NAME
6699 | '(' multiple_expr_list ')' In NAME
6700 | non_unary_print_expr AND newline_opt print_expr
6701 | non_unary_print_expr OR newline_opt print_expr
6702 | non_unary_print_expr '?' print_expr ':' print_expr
6703 | NUMBER
6704 | STRING
6705 | lvalue
6706 | ERE
6707 | lvalue INCR
6708 | lvalue DECR
6709 | INCR lvalue
6710 | DECR lvalue
6711 | lvalue POW_ASSIGN print_expr
6712 | lvalue MOD_ASSIGN print_expr
6713 | lvalue MUL_ASSIGN print_expr
6714 | lvalue DIV_ASSIGN print_expr
6715 | lvalue ADD_ASSIGN print_expr
6716 | lvalue SUB_ASSIGN print_expr
6717 | lvalue '=' print_expr
6718 | FUNC_NAME '(' expr_list_opt ')'
6719 | /* no white space allowed before '(' */
6720 | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
6721 | BUILTIN_FUNC_NAME
6722 ;

6723 lvalue : NAME
6724 | NAME '[' expr_list ']'
6725 | '$' expr
6726 ;

6727 non_unary_input_function : simple_get
6728 | simple_get '<' expr
6729 | non_unary_expr '|' simple_get
6730 ;

6731 unary_input_function : unary_expr '|' simple_get
6732 ;

6733 simple_get : GETLINE
6734 | GETLINE lvalue
6735 ;

6736 newline_opt : /* empty */
6737 | newline_opt NEWLINE
6738 ;

```

6739 This grammar has several ambiguities that shall be resolved as follows:

- 6740 • Operator precedence and associativity shall be as described in Table 4-1 (on page 2370).
- 6741 • In case of ambiguity, an **else** shall be associated with the most immediately preceding **if** that
- 6742 would satisfy the grammar.
- 6743 • In some contexts, a slash ( / ) that is used to surround an ERE could also be the division
- 6744 operator. This shall be resolved in such a way that wherever the division operator could
- 6745 appear, a slash is assumed to be the division operator. (There is no unary division operator.)

6746 One convention that might not be obvious from the formal grammar is where <newline>

6747 characters are acceptable. There are several obvious placements such as terminating a statement,

6748 and a backslash can be used to escape <newline> characters between any lexical tokens. In

6749 addition, <newline> characters without backslashes can follow a comma, an open brace, logical

6750 AND operator ("&&"), logical OR operator (" | "), the **do** keyword, the **else** keyword, and the

6751 closing parenthesis of an **if**, **for**, or **while** statement. For example:

```
6752 { print $1,
6753 $2 }
```

### 6754 Lexical Conventions

6755 The lexical conventions for *awk* programs, with respect to the preceding grammar, shall be as

6756 follows:

- 6757 1. Except as noted, *awk* shall recognize the longest possible token or delimiter beginning at a
- 6758 given point.
- 6759 2. A comment shall consist of any characters beginning with the number sign character and
- 6760 terminated by, but excluding the next occurrence of, a <newline> character. Comments
- 6761 shall have no effect, except to delimit lexical tokens.
- 6762 3. The <newline> character shall be recognized as the token **NEWLINE**.
- 6763 4. A backslash character immediately followed by a <newline> character shall have no effect.
- 6764 5. The token **STRING** shall represent a string constant. A string constant shall begin with the
- 6765 character ' " '. Within a string constant, a backslash character shall be considered to begin
- 6766 an escape sequence as specified in the table in the Base Definitions volume of
- 6767 IEEE Std. 1003.1-200x, Chapter 5, File Format Notation ( '\ ', '\a', '\b', '\f', '\n',
- 6768 '\r', '\t', '\v' ). In addition, the escape sequences in Table 4-2 (on page 2376) shall be
- 6769 recognized. A <newline> character shall not occur within a string constant. A string
- 6770 constant shall be terminated by the first unescaped occurrence of the character ' " ' after
- 6771 the one that begins the string constant. The value of the string shall be the sequence of all
- 6772 unescaped characters and values of escape sequences between, but not including, the two
- 6773 delimiting ' " ' characters.
- 6774 6. The token **ERE** represents an extended regular expression constant. An ERE constant shall
- 6775 begin with the slash character. Within an ERE constant, a backslash character shall be
- 6776 considered to begin an escape sequence as specified in the table in the Base Definitions
- 6777 volume of IEEE Std. 1003.1-200x, Chapter 5, File Format Notation. In addition, the escape
- 6778 sequences in Table 4-2 (on page 2376) shall be recognized. The application shall ensure that
- 6779 a <newline> character does not occur within an ERE constant. An ERE constant shall be
- 6780 terminated by the first unescaped occurrence of the slash character after the one that
- 6781 begins the ERE constant. The extended regular expression represented by the ERE constant
- 6782 shall be the sequence of all unescaped characters and values of escape sequences between,
- 6783 but not including, the two delimiting slash characters.

6784 7. A <blank> character shall have no effect, except to delimit lexical tokens or within  
6785 **STRING** or **ERE** tokens.

6786 8. The token **NUMBER** shall represent a numeric constant. Its form and numeric value shall  
6787 be equivalent to either of the tokens **floating-constant** or **integer-constant** as specified by  
6788 the ISO C standard, with the following exceptions:

6789 a. An integer constant cannot begin with 0x or include the hexadecimal digits 'a', 'b',  
6790 'c', 'd', 'e', 'f', 'A', 'B', 'C', 'D', 'E', or 'F'.

6791 b. The value of an integer constant beginning with 0 shall be taken in decimal rather  
6792 than octal.

6793 c. An integer constant cannot include a suffix ('u', 'U', 'l', or 'L').

6794 d. A floating constant cannot include a suffix ('f', 'F', 'l', or 'L').

6795 If the value is too large or too small to be representable, the behavior is undefined.

6796 9. A sequence of underscores, digits, and alphabetic characters from the portable character set (see the  
6797 Base Definitions volume of IEEE Std. 1003.1-200x, Section 6.1, Portable Character Set),  
6798 beginning with an underscore or alphabetic, shall be considered a word.

6799 10. The following words are keywords that shall be recognized as individual tokens; the name  
6800 of the token is the same as the keyword:

|      |                 |               |             |                 |              |               |
|------|-----------------|---------------|-------------|-----------------|--------------|---------------|
| 6801 | <b>BEGIN</b>    | <b>delete</b> | <b>END</b>  | <b>function</b> | <b>in</b>    | <b>printf</b> |
| 6802 | <b>break</b>    | <b>do</b>     | <b>exit</b> | <b>getline</b>  | <b>next</b>  | <b>return</b> |
| 6803 | <b>continue</b> | <b>else</b>   | <b>for</b>  | <b>if</b>       | <b>print</b> | <b>while</b>  |

6804 11. The following words are names of built-in functions and shall be recognized as the token  
6805 **BUILTIN\_FUNC\_NAME**:

|      |              |               |              |                |                |                |
|------|--------------|---------------|--------------|----------------|----------------|----------------|
| 6806 | <b>atan2</b> | <b>gsub</b>   | <b>log</b>   | <b>split</b>   | <b>sub</b>     | <b>toupper</b> |
| 6807 | <b>close</b> | <b>index</b>  | <b>match</b> | <b>sprintf</b> | <b>substr</b>  |                |
| 6808 | <b>cos</b>   | <b>int</b>    | <b>rand</b>  | <b>sqrt</b>    | <b>system</b>  |                |
| 6809 | <b>exp</b>   | <b>length</b> | <b>sin</b>   | <b>srand</b>   | <b>tolower</b> |                |

6810 The above-listed keywords and names of built-in functions are considered reserved words.

6811 12. The token **NAME** shall consist of a word that is not a keyword or a name of a built-in  
6812 function and is not followed immediately (without any delimiters) by the ' ( ' character.

6813 13. The token **FUNC\_NAME** shall consist of a word that is not a keyword or a name of a  
6814 built-in function, followed immediately (without any delimiters) by the ' ( ' character. The  
6815 ' ( ' character shall not be included as part of the token.

6816 14. The following two-character sequences shall be recognized as the named tokens:

| Token Name             | Sequence | Token Name      | Sequence |
|------------------------|----------|-----------------|----------|
| 6818 <b>ADD_ASSIGN</b> | +=       | <b>NO_MATCH</b> | !~       |
| 6819 <b>SUB_ASSIGN</b> | -=       | <b>EQ</b>       | ==       |
| 6820 <b>MUL_ASSIGN</b> | *=       | <b>LE</b>       | <=       |
| 6821 <b>DIV_ASSIGN</b> | /=       | <b>GE</b>       | >=       |
| 6822 <b>MOD_ASSIGN</b> | %=       | <b>NE</b>       | !=       |
| 6823 <b>POW_ASSIGN</b> | ^=       | <b>INCR</b>     | ++       |
| 6824 <b>OR</b>         |          | <b>DECR</b>     | --       |
| 6825 <b>AND</b>        | &&       | <b>APPEND</b>   | >>       |

6826 15. The following single characters shall be recognized as tokens whose names are the  
6827 character:

6828 <newline> { } ( ) [ ] , ; + - \* % ^ ! > < | ? : ~ \$ =

6829 There is a lexical ambiguity between the token **ERE** and the tokens **'/'** and **DIV\_ASSIGN**.  
6830 When an input sequence begins with a slash character in any syntactic context where the token  
6831 **'/'** or **DIV\_ASSIGN** could appear as the next token in a valid program, the longer of those two  
6832 tokens that can be recognized shall be recognized. In any other syntactic context where the token  
6833 **ERE** could appear as the next token in a valid program, the token **ERE** shall be recognized.

#### 6834 EXIT STATUS

6835 The following exit values shall be returned:

6836 0 All input files were processed successfully.

6837 >0 An error occurred.

6838 The exit status can be altered within the program by using an **exit** expression.

#### 6839 CONSEQUENCES OF ERRORS

6840 If any *file* operand is specified and the named file cannot be accessed, *awk* shall write a  
6841 diagnostic message to standard error and terminate without any further action.

6842 If the program specified by either the *program* operand or a *progfile* operand is not a valid *awk*  
6843 program (as specified in the EXTENDED DESCRIPTION section), the behavior is undefined.

#### 6844 APPLICATION USAGE

6845 The **index**, **length**, **match**, and **substr** functions should not be confused with similar functions in  
6846 the ISO C standard; the *awk* versions deal with characters, while the ISO C standard deals with  
6847 bytes.

6848 Because the concatenation operation is represented by adjacent expressions rather than an  
6849 explicit operator, it is often necessary to use parentheses to enforce the proper evaluation  
6850 precedence.

#### 6851 EXAMPLES

6852 The *awk* program specified in the command line is most easily specified within single-quotes (for  
6853 example, *'program'*) for applications using *sh*, because *awk* programs commonly contain  
6854 characters that are special to the shell, including double-quotes. In the cases where an *awk*  
6855 program contains single-quote characters, it is usually easiest to specify most of the program as  
6856 strings within single-quotes concatenated by the shell with quoted single-quote characters. For  
6857 example:

```
6858 awk '/\''/ { print "quote:", $0 }'
```

6859 prints all lines from the standard input containing a single-quote character, prefixed with *quote:*.

6860 The following are examples of simple *awk* programs:

6861 1. Write to the standard output all input lines for which field 3 is greater than 5:

```
6862 $3 > 5
```

6863 2. Write every tenth line:

```
6864 (NR % 10) == 0
```

6865 3. Write any line with a substring matching the regular expression:

```
6866 /(G|D)(2[0-9][[:alpha:]]*)/
```

- 6867 4. Print any line with a substring containing a 'G' or 'D', followed by a sequence of digits  
6868 and characters. This example uses character classes **digit** and **alpha** to match language-  
6869 independent digit and alphabetic characters respectively:
- ```
6870 /(G|D)([[:digit:][:alpha:]]*)/
```
- 6871 5. Write any line in which the second field matches the regular expression and the fourth
6872 field does not:
- ```
6873 $2 ~ /xyz/ && $4 !~ /xyz/
```
- 6874 6. Write any line in which the second field contains a backslash:
- ```
6875 $2 ~ /\\/
```
- 6876 7. Write any line in which the second field contains a backslash. Note that backslash escapes
6877 are interpreted twice, once in lexical processing of the string and once in processing the
6878 regular expression:
- ```
6879 $2 ~ "\\\""
```
- 6880 8. Write the second to the last and the last field in each line. Separate the fields by a colon:
- ```
6881 {OFS=":";print $(NF-1), $NF}
```
- 6882 9. Write the line number and number of fields in each line. The three strings representing the
6883 line number, the colon, and the number of fields are concatenated and that string is written
6884 to standard output:
- ```
6885 {print NR ":" NF}
```
- 6886 10. Write lines longer than 72 characters:
- ```
6887 length($0) > 72
```
- 6888 11. Write first two fields in opposite order separated by the **OFS**:
- ```
6889 { print $2, $1 }
```
- 6890 12. Same, with input fields separated by comma or <space> and <tab> characters, or both:
- ```
6891 BEGIN { FS = ",[ \t]*|[ \t]+" }
6892 { print $2, $1 }
```
- 6893 13. Add up first column, print sum, and average:
- ```
6894 {s += $1 }
6895 END {print "sum is ", s, " average is", s/NR}
```
- 6896 14. Write fields in reverse order, one per line (many lines out for each line in):
- ```
6897 { for (i = NF; i > 0; --i) print $i }
```
- 6898 15. Write all lines between occurrences of the strings **start** and **stop**:
- ```
6899 /start/, /stop/
```
- 6900 16. Write all lines whose first field is different from the previous one:
- ```
6901 $1 != prev { print; prev = $1 }
```
- 6902 17. Simulate *echo*:
- ```
6903 BEGIN {
6904 for (i = 1; i < ARGV; ++i)
6905 printf("%s%s", ARGV[i], i==ARGV-1?"\n":" ")
```

```

6906 }
6907 18. Write the path prefixes contained in the PATH environment variable, one per line:
6908 BEGIN {
6909 n = split (ENVIRON["PATH"], path, ":")
6910 for (i = 1; i <= n; ++i)
6911 print path[i]
6912 }
6913 19. If there is a file named input containing page headers of the form:
6914 Page #
6915 and a file named program that contains:
6916 /Page/ { $2 = n++; }
6917 { print }
6918 then the command line:
6919 awk -f program n=5 input
6920 prints the file input, filling in page numbers starting at 5.

```

#### 6921 RATIONALE

6922 The ISO POSIX-2 standard description is based on the new *awk*, “*nawk*”, (see the referenced *The* |  
6923 *AWK Programming Language*), which introduced a number of new features to the historical *awk*:

- 6924 1. New keywords: **delete**, **do**, **functin**, **return**
- 6925 2. New built-in functions: **atan2**, **close**, **cos**, **gsub**, **match**, **rand**, **sin**, **srand**, **sub**, **system**
- 6926 3. New predefined variables: **FNR**, **ARGC**, **ARGV**, **RSTART**, **RLENGTH**, **SUBSEP**
- 6927 4. New expression operators: **?**, **:**, **..**, **^**
- 6928 5. The **FS** variable and the third argument to **split**, now treated as extended regular  
6929 expressions.
- 6930 6. The operator precedence, changed to more closely match the C language. Two examples  
6931 of code that operate differently are:

```

6932 while (n /= 10 > 1) ...
6933 if (!"wk" ~ /bwk/) ...

```

6934 Several features have been added based on newer implementations of *awk*:

- 6935 • Multiple instances of **-f progfile** are permitted
- 6936 • The new option **-v assignment**
- 6937 • The new predefined variable **ENVIRON**
- 6938 • New built-in functions **toupper**, and **tolower**
- 6939 • More formatting capabilities are added to **printf** to match the ISO C standard

6940 The overall *awk* syntax has always been based on the C language, with a few features from the  
6941 shell command language and other sources. Because of this, it is not completely compatible with  
6942 any other language, which has caused confusion for some users. It is not the intent of the  
6943 standard developers to address such issues. IEEE Std. 1003.1-200x has made a few relatively  
6944 minor changes toward making the language more compatible with the C language as specified  
6945 by the ISO C standard; most of these changes are based on similar changes in recent

6946 implementations, as described above. There remain several C-language conventions that are not  
6947 in *awk*. One of the notable ones is the comma operator, which is commonly used to specify  
6948 multiple expressions in the C language **for** statement. Also, there are various places where *awk*  
6949 is more restrictive than the C language regarding the type of expression that can be used in a given  
6950 context. These limitations are due to the different features that the *awk* language does provide.

6951 Regular expressions in *awk* have been extended somewhat from historical implementations to  
6952 make them a pure superset of extended regular expressions, as defined by IEEE Std. 1003.1-200x  
6953 (see the Base Definitions volume of IEEE Std. 1003.1-200x, Section 9.4, Extended Regular  
6954 Expressions). The main extensions are internationalization features and interval expressions.  
6955 Historical implementations of *awk* have long supported backslash escape sequences as an  
6956 extension to extended regular expressions, and this extension has been retained despite  
6957 inconsistency with other utilities. The number of escape sequences recognized in both extended  
6958 regular expressions and strings has varied (generally increasing with time) among  
6959 implementations. The set specified by IEEE Std. 1003.1-200x includes most sequences known to  
6960 be supported by popular implementations and by the ISO C standard. One sequence that is not  
6961 supported is hexadecimal value escapes beginning with `'\x'`. This would allow values  
6962 expressed in more than 9 bits to be used within *awk* as in the ISO C standard. However, because  
6963 this syntax has a non-deterministic length, it does not permit the subsequent character to be a  
6964 hexadecimal digit. This limitation can be dealt with in the C language by the use of lexical string  
6965 concatenation. In the *awk* language, concatenation could also be a solution for strings, but not for  
6966 extended regular expressions (either lexical ERE tokens or strings used dynamically as regular  
6967 expressions). Because of this limitation, the feature has not been added to IEEE Std. 1003.1-200x.

6968 When a string variable is used in a context where an extended regular expression normally  
6969 appears (where the lexical token ERE is used in the grammar) the string does not contain the  
6970 literal slashes.

6971 Some versions of *awk* allow the form:

```
6972 func name(args, ...) { statements }
```

6973 This has been deprecated by the authors of the language, who asked that it not be included in  
6974 IEEE Std. 1003.1-200x.

6975 Historical implementations of *awk* produce an error if a **next** statement is executed in a **BEGIN**  
6976 action, and cause *awk* to terminate if a **next** statement is executed in an **END** action. This  
6977 behavior has not been documented, and it was not believed that it was necessary to standardize  
6978 it.

6979 The specification of conversions between string and numeric values is much more detailed than  
6980 in the documentation of historical implementations or in the referenced *The AWK Programming*  
6981 *Language*. Although most of the behavior is designed to be intuitive, the details are necessary to  
6982 ensure compatible behavior from different implementations. This is especially important in  
6983 relational expressions since the types of the operands determine whether a string or numeric  
6984 comparison is performed. From the perspective of an application writer, it is usually sufficient to  
6985 expect intuitive behavior and to force conversions (by adding zero or concatenating a null  
6986 string) when the type of an expression does not obviously match what is needed. The intent has  
6987 been to specify historical practice in almost all cases. The one exception is that, in historical  
6988 implementations, variables and constants maintain both string and numeric values after their  
6989 original value is converted by any use. This means that referencing a variable or constant can  
6990 have unexpected side effects. For example, with historical implementations the following  
6991 program:

```
6992 {
6993 a = "+2"
```

```

6994 b = 2
6995 if (NR % 2)
6996 c = a + b
6997 if (a == b)
6998 print "numeric comparison"
6999 else
7000 print "string comparison"
7001 }

```

7002 would perform a numeric comparison (and output numeric comparison) for each odd-  
7003 numbered line, but perform a string comparison (and output string comparison) for each even-  
7004 numbered line. IEEE Std. 1003.1-200x ensures that comparisons will be numeric if necessary.  
7005 With historical implementations, the following program:

```

7006 BEGIN {
7007 OFMT = "%e"
7008 print 3.14
7009 OFMT = "%f"
7010 print 3.14
7011 }

```

7012 would output "3.140000e+00" twice, because in the second **print** statement the constant  
7013 "3.14" would have a string value from the previous conversion. IEEE Std. 1003.1-200x requires  
7014 that the output of the second **print** statement be "3.140000". The behavior of historical  
7015 implementations was seen as too unintuitive and unpredictable.

7016 It was pointed out that with the rules contained in early drafts, the following script would print  
7017 nothing:

```

7018 BEGIN {
7019 y[1.5] = 1
7020 OFMT = "%e"
7021 print y[1.5]
7022 }

```

7023 Therefore, a new variable, **CONVFMT**, was introduced. The **OFMT** variable is now restricted to  
7024 affecting output conversions of numbers to strings and **CONVFMT** is used for internal  
7025 conversions, such as comparisons or array indexing. The default value is the same as that for  
7026 **OFMT**, so unless a program changes **CONVFMT** (which no historical program would do), it  
7027 will receive the historical behavior associated with internal string conversions.

7028 The POSIX *awk* lexical and syntactic conventions are specified more formally than in other  
7029 sources. Again the intent has been to specify historical practice. One convention that may not be  
7030 obvious from the formal grammar as in other verbal descriptions is where <newline> characters  
7031 are acceptable. There are several obvious placements such as terminating a statement, and a  
7032 backslash can be used to escape <newline> characters between any lexical tokens. In addition,  
7033 <newline> characters without backslashes can follow a comma, an open brace, a logical AND  
7034 operator ("&&"), a logical OR operator ("||"), the **do** keyword, the **else** keyword, and the  
7035 closing parenthesis of an **if**, **for**, or **while** statement. For example:

```

7036 { print $1,
7037 $2 }

```

7038 The requirement that *awk* add a trailing <newline> character to the program argument text is to  
7039 simplify the grammar, making it match a text file in form. There is no way for an application or  
7040 test suite to determine whether a literal <newline> is added or whether *awk* simply acts as if it  
7041 did.



7042 IEEE Std. 1003.1-200x requires several changes from historical implementations in order to  
 7043 support internationalization. Probably the most subtle of these is the use of the decimal-point  
 7044 character, defined by the *LC\_NUMERIC* category of the locale, in representations of floating-  
 7045 point numbers. This locale-specific character is used in recognizing numeric input, in converting  
 7046 between strings and numeric values, and in formatting output. However, regardless of locale,  
 7047 the period character (the decimal-point character of the POSIX locale) is the decimal-point  
 7048 character recognized in processing *awk* programs (including assignments in command line  
 7049 arguments). This is essentially the same convention as the one used in the ISO C standard. The  
 7050 difference is that the C language includes the *setlocale()* function, which permits an application  
 7051 to modify its locale. Because of this capability, a C application begins executing with its locale  
 7052 set to the C locale, and only executes in the environment-specified locale after an explicit call to  
 7053 *setlocale()*. However, adding such an elaborate new feature to the *awk* language was seen as  
 7054 inappropriate for IEEE Std. 1003.1-200x. It is possible to execute an *awk* program explicitly in any  
 7055 desired locale by setting the environment in the shell.

7056 The undefined behavior resulting from NULs in extended regular expressions allows future  
 7057 extensions for the GNU *gawk* program to process binary data.

7058 The behavior in the case of invalid *awk* programs (including lexical, syntactic, and semantic  
 7059 errors) is undefined because it was considered overly limiting on implementations to specify. In  
 7060 most cases such errors can be expected to produce a diagnostic and a non-zero exit status.  
 7061 However, some implementations may choose to extend the language in ways that make use of  
 7062 certain invalid constructs. Other invalid constructs might be deemed worthy of a warning, but  
 7063 otherwise cause some reasonable behavior. Still other constructs may be very difficult to detect  
 7064 in some implementations. Also, different implementations might detect a given error during an  
 7065 initial parsing of the program (before reading any input files) while others might detect it when  
 7066 executing the program after reading some input. Implementors should be aware that diagnosing  
 7067 errors as early as possible and producing useful diagnostics can ease debugging of applications,  
 7068 and thus make an implementation more usable.

7069 The unspecified behavior from using multi-character **RS** values is to allow possible future  
 7070 extensions based on extended regular expressions used for record separators. Historical  
 7071 implementations take the first character of the string and ignore the others.

7072 Unspecified behavior when *split(string,array,<null>)* is used is to allow a proposed future  
 7073 extension that would split up a string into an array of individual characters.

7074 In the context of the **getline** function, equally good arguments for different precedences of the |  
 7075 and < operators can be made. Historical practice has been that:

```
7076 getline < "a" "b"
```

7077 is parsed as:

```
7078 (getline < "a") "b"
```

7079 although many would argue that the intent was that the file **ab** should be read. However:

```
7080 getline < "x" + 1
```

7081 parses as:

```
7082 getline < ("x" + 1)
```

7083 Similar problems occur with the | version of **getline**, particularly in combination with \$. For  
 7084 example:

```
7085 $"echo hi" | getline
```

7086 (This situation is particularly problematic when used in a **print** statement, where the `|getline`  
7087 part might be a redirection of the **print**.)

7088 Since in most cases such constructs are not (or at least should not) be used (because they have a  
7089 natural ambiguity for which there is no conventional parsing), the meaning of these constructs  
7090 has been made explicitly unspecified. (The effect is that a portable application that runs into the  
7091 problem must parenthesize to resolve the ambiguity.) There appeared to be few if any actual  
7092 uses of such constructs.

7093 Grammars can be written that would cause an error under these circumstances. Where  
7094 backwards compatibility is not a large consideration, implementors may wish to use such  
7095 grammars.

7096 Some historical implementations have allowed some built-in functions to be called without an  
7097 argument list, the result being a default argument list chosen in some “reasonable” way. Use of  
7098 **length** as a synonym for **length(\$0)** is the only one of these forms that is thought to be widely  
7099 known or widely used; this particular form is documented in various places (for example, most  
7100 historical *awk* reference pages, although not in the referenced *The AWK Programming Language*)  
7101 as legitimate practice. With this exception, default argument lists have always been  
7102 undocumented and vaguely defined, and it is not at all clear how (or if) they should be  
7103 generalized to user-defined functions. They add no useful functionality and preclude possible  
7104 future extensions that might need to name functions without calling them. Not standardizing  
7105 them seems the simplest course. The standard developers considered that **length** merited special  
7106 treatment, however, since it has been documented in the past and sees possibly substantial use  
7107 in historical programs. Accordingly, this usage has been made legitimate, but Issue 5 removed  
7108 the obsolescent marking for XSI-conforming implementations and many otherwise conforming  
7109 applications depend on this feature.

7110 In **sub** and **gsub**, if *repl* is a string literal (the lexical token **STRING**), then two consecutive  
7111 backslash characters should be used in the string to ensure a single backslash will precede the  
7112 ampersand when the resultant string is passed to the function. (For example, to specify one  
7113 literal ampersand in the replacement string, use **gsub(ERE, "\\&")**.)

7114 Historically the only special character in the *repl* argument of **sub** and **gsub** string functions was  
7115 the ampersand ('&') character and preceding it with the backslash character was used to turn  
7116 off its special meaning.

7117 The description in the ISO POSIX-2:1993 standard introduced behavior such that the backslash  
7118 character was another special character and it was unspecified whether there were any other  
7119 special characters. This description introduced several portability problems, some of which are  
7120 described below, and so it has been replaced with the more historical description. Some of the  
7121 problems include:

- 7122 • Historically, to create the replacement string, a script could use **gsub(ERE, "\\&")**, but with  
7123 the ISO POSIX-2:1993 standard wording, it was necessary to use **gsub(ERE, "\\&")**.  
7124 Backslash characters are doubled here because all string literals are subject to lexical analysis,  
7125 which would reduce each pair of backslash characters to a single backslash before being  
7126 passed to **gsub**.
- 7127 • Since it was unspecified what the special characters were, for portable scripts to guarantee  
7128 that characters are printed literally, each character had to be preceded with a backslash. (For  
7129 example, a portable script had to use **gsub(ERE, "\\h\\i")** to produce a replacement string  
7130 of "hi".)

7131 The description for comparisons in the ISO POSIX-2:1993 standard did not properly describe  
7132 historical practice because of the way numeric strings are compared as numbers. The current  
7133 rules cause the following code:

```

7134 if (0 == "000")
7135 print "strange, but true"
7136 else
7137 print "not true"

```

7138 to do a numeric comparison, causing the **if** to succeed. It should be intuitively obvious that this  
 7139 is incorrect behavior, and indeed, no historical implementation of *awk* actually behaves this way.

7140 To fix this problem, the definition of *numeric string* was enhanced to include only those values  
 7141 obtained from specific circumstances (mostly external sources) where it is not possible to  
 7142 determine unambiguously whether the value is intended to be a string or a numeric.

7143 Variables that are assigned to a numeric string shall also be treated as a numeric string. (For  
 7144 example, the notion of a numeric string can be propagated across assignments.) In comparisons,  
 7145 all variables having the uninitialized value are to be treated as a numeric operand evaluating to  
 7146 the numeric value zero.

7147 Uninitialized variables include all types of variables including scalars, array elements, and fields.  
 7148 The definition of an uninitialized value in **Variables and Special Variables** (on page 2373) is  
 7149 necessary to describe the value placed on uninitialized variables and on fields that are valid (for  
 7150 example, < \$NF) but have no characters in them and to describe how these variables are to be  
 7151 used in comparisons. A valid field, such as \$1, that has no characters in it can be obtained by  
 7152 from an input line of "\t\t" when FS='\t'. Historically, the comparison (\$1<10) was done  
 7153 numerically after evaluating \$1 to the value zero.

7154 The phrase "... also shall have the numeric value of the numeric string" was removed from  
 7155 several sections of the ISO POSIX-2:1993 standard because it specifies an unnecessary  
 7156 implementation detail. It is not necessary for IEEE Std. 1003.1-200x to specify that these objects  
 7157 be assigned two different values. It is only necessary to specify that these objects may evaluate  
 7158 to two different values depending on context.

7159 The description of numeric string processing is based on the behavior of the *atof()* function in  
 7160 the ISO C standard. While it is not a requirement for an implementation to use this function,  
 7161 many historical implementations of *awk* do. In the ISO C standard, floating-point constants use a  
 7162 period as a decimal point character for the language itself, independent of the current locale, but  
 7163 the *atof()* function and the associated *strtod()* function use the decimal point character of the  
 7164 current locale when converting strings to numeric values. Similarly in *awk*, floating point  
 7165 constants in an *awk* script use a period independent of the locale, but input strings use the  
 7166 decimal point character of the locale.

#### 7167 FUTURE DIRECTIONS

7168 None.

#### 7169 SEE ALSO

7170 *grep*, *lex*, *sed*, the System Interfaces volume of IEEE Std. 1003.1-200x, *atof()*, *setlocale()*, *strtod()*

#### 7171 CHANGE HISTORY

7172 First released in Issue 2.

#### 7173 Issue 4

7174 Aligned with the ISO/IEC 9945-2:1993 standard.

#### 7175 Issue 4, Version 2

7176 The EXAMPLES section is corrected as follows:

- 7177 • In Example 10, the braces are removed.
- 7178 • In Example 17, the invocation of **printf** is corrected.

7179 **Issue 5**

7180 FUTURE DIRECTIONS section added.

7181 **Issue 6**

7182 The *awk* utility is aligned with the IEEE P1003.2b draft standard.

7183 The normative text is reworded to avoid use of the term “must” for application requirements.

7184 **NAME**

7185            basename — return non-directory portion of a path name

7186 **SYNOPSIS**7187            basename *string* [*suffix*]7188 **DESCRIPTION**

7189            The *string* operand shall be treated as a path name, as defined in the Base Definitions volume of  
 7190            IEEE Std. 1003.1-200x, Section 3.268, Path Name. The string *string* shall be converted to the file  
 7191            name corresponding to the last path name component in *string* and then the suffix string *suffix*, if  
 7192            present, shall be removed. This shall be done by performing actions equivalent to the following  
 7193            steps in order:

- 7194            1. If *string* is a null string, it is unspecified whether the resulting string is ' . ' or a null string.  
 7195            In either case, skip steps 2 through 6.
- 7196            2. If *string* is "///", it is implementation-defined whether steps 3 to 6 are skipped or  
 7197            processed.
- 7198            3. If *string* consists entirely of slash characters, *string* shall be set to a single slash character. In  
 7199            this case, skip steps 4 to 6.
- 7200            4. If there are any trailing slash characters in *string*, they shall be removed.
- 7201            5. If there are any slash characters remaining in *string*, the prefix of *string* up to and including  
 7202            the last slash character in *string* shall be removed.
- 7203            6. If the *suffix* operand is present, is not identical to the characters remaining in *string*, and is  
 7204            identical to a suffix of the characters remaining in *string*, the suffix *suffix* shall be removed  
 7205            from *string*. Otherwise, *string* is modified by this step. It shall not be considered an error if  
 7206            *suffix* is not found in *string*.

7207            The resulting string shall be written to standard output.

7208 **OPTIONS**

7209            None.

7210 **OPERANDS**

7211            The following operands shall be supported:

7212            *string*        A string.7213            *suffix*       A string.7214 **STDIN**

7215            Not used.

7216 **INPUT FILES**

7217            None.

7218 **ENVIRONMENT VARIABLES**7219            The following environment variables shall affect the execution of *basename*:

7220            *LANG*        Provide a default value for the internationalization variables that are unset or null.  
 7221            If *LANG* is unset or null, the corresponding value from the implementation-  
 7222            defined default locale shall be used. If any of the internationalization variables  
 7223            contains an invalid setting, the utility shall behave as if none of the variables had  
 7224            been defined.

7225            *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
 7226            internationalization variables.

7227            *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 7228 characters (for example, single-byte as opposed to multi-byte characters in  
 7229 arguments).

7230            *LC\_MESSAGES*  
 7231                      Determine the locale that should be used to affect the format and contents of  
 7232 diagnostic messages written to standard error.

7233 XSI        *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

7234 **ASYNCHRONOUS EVENTS**  
 7235            Default.

7236 **STDOUT**  
 7237            The *basename* utility shall write a line to the standard output in the following format:  
 7238            "%s\n", <resulting string>

7239 **STDERR**  
 7240            Used only for diagnostic messages.

7241 **OUTPUT FILES**  
 7242            None.

7243 **EXTENDED DESCRIPTION**  
 7244            None.

7245 **EXIT STATUS**  
 7246            The following exit values shall be returned:  
 7247            0 Successful completion.  
 7248            >0 An error occurred.

7249 **CONSEQUENCES OF ERRORS**  
 7250            Default.

7251 **APPLICATION USAGE**  
 7252            The definition of *pathname* specifies implementation-defined behavior for path names starting  
 7253 with two slash characters. Therefore, applications shall not arbitrarily add slashes to the  
 7254 beginning of a path name unless they can ensure that there are more or less than two or are  
 7255 prepared to deal with the implementation-defined consequences.

7256 **EXAMPLES**  
 7257            If the string *string* is a valid path name:  
 7258            `$(basename "string")`  
 7259            produces a file name that could be used to open the file named by *string* in the directory  
 7260 returned by:  
 7261            `$(dirname "string")`  
 7262            If the string *string* is not a valid path name, the same algorithm is used, but the result need not be  
 7263 a valid file name. The *basename* utility is not expected to make any judgements about the validity  
 7264 of *string* as a path name; it just follows the specified algorithm to produce a result string.  
 7265            The following shell script compiles `/usr/src/cmd/cat.c` and moves the output to a file named `cat`  
 7266 in the current directory when invoked with the argument `/usr/src/cmd/cat` or with the argument  
 7267 `/usr/src/cmd/cat.c`:

7268           c99 \$(dirname "\$1")/\$(basename "\$1" .c).c  
7269           mv a.out \$(basename "\$1" .c)

7270 **RATIONALE**

7271           The behaviors of *basename* and *dirname* have been coordinated so that when *string* is a valid path  
7272           name:

7273           \$(basename "*string*")

7274           would be a valid file name for the file in the directory:

7275           \$(dirname "*string*")

7276           This would not work for the early proposal versions of these utilities due to the way it specified  
7277           handling of trailing slashes.

7278           Since the definition of *pathname* specifies implementation-defined behavior for path names  
7279           starting with two slash characters, this volume of IEEE Std. 1003.1-200x specifies similar  
7280           implementation-defined behavior for the *basename* and *dirname* utilities.

7281 **FUTURE DIRECTIONS**

7282           None.

7283 **SEE ALSO**

7284           *dirname*, Section 2.5 (on page 2241)

7285 **CHANGE HISTORY**

7286           First released in Issue 2.

7287 **Issue 4**

7288           Aligned with the ISO/IEC 9945-2:1993 standard.

7289 **Issue 6**

7290           IEEE PASC Interpretation 1003.2 #164 has been applied.

7291           The normative text is reworded to avoid use of the term “must” for application requirements.

7292 **NAME**

7293 batch — schedule commands to be executed in a batch queue

7294 **SYNOPSIS**

7295 UP *batch*

7296

7297 **DESCRIPTION**

7298 The *batch* utility shall read commands from standard input and schedule them for execution in a  
7299 batch queue. It shall be the equivalent of the command:

7300 at -q b -m now

7301 where queue *b* is a special *at* queue, specifically for batch jobs. Batch jobs shall be submitted to  
7302 the batch queue with no time constraints and shall be run by the system using algorithms, based  
7303 on unspecified factors, that may vary with each invocation of *batch*.

7304 XSI Users are permitted to use *batch* if their name appears in the file **/usr/lib/cron/at.allow**. If that file  
7305 does not exist, the file **/usr/lib/cron/at.deny** is checked to determine whether the user should be  
7306 denied access to *batch*. If neither file exists, only a process with the appropriate privileges is  
7307 allowed to submit a job. If only **at.deny** exists and is empty, global usage is permitted. The  
7308 **at.allow** and **at.deny** files consist of one user name per line.

7309 **OPTIONS**

7310 None.

7311 **OPERANDS**

7312 None.

7313 **STDIN**

7314 The standard input shall be a text file consisting of commands acceptable to the shell command  
7315 language described in Chapter 2 (on page 2235).

7316 **INPUT FILES**

7317 XSI The text files **/usr/lib/cron/at.allow** and **/usr/lib/cron/at.deny** contain user names, one per line, of  
7318 users who are, respectively, authorized or denied access to the *at* and *batch* utilities.

7319 **ENVIRONMENT VARIABLES**

7320 The following environment variables shall affect the execution of *batch*:

7321 **LANG** Provide a default value for the internationalization variables that are unset or null.  
7322 If **LANG** is unset or null, the corresponding value from the implementation-  
7323 defined default locale shall be used. If any of the internationalization variables  
7324 contains an invalid setting, the utility shall behave as if none of the variables had  
7325 been defined.

7326 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
7327 internationalization variables.

7328 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
7329 characters (for example, single-byte as opposed to multi-byte characters in  
7330 arguments and input files).

7331 **LC\_MESSAGES**

7332 Determine the locale that should be used to affect the format and contents of  
7333 diagnostic messages written to standard error and informative messages written to  
7334 standard output.

7335 **LC\_TIME** Determine the format and contents for date and time strings written by *batch*.



- 7336 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 7337 **SHELL** Determine the name of a command interpreter to be used to invoke the at-job. If  
7338 the variable is unset or null, *sh* shall be used. If it is set to a value other than a name  
7339 for *sh*, the implementation shall do one of the following: use that shell; use *sh*; use  
7340 the login shell from the user database; any of the preceding accompanied by a  
7341 warning diagnostic about which was chosen.
- 7342 **TZ** Determine the timezone. The job shall be submitted for execution at the time  
7343 specified by *timespec* or *-t time* relative to the timezone specified by the *TZ*  
7344 variable. If *timespec* specifies a timezone, it overrides *TZ*. If *timespec* does not  
7345 specify a timezone and *TZ* is unset or null, an unspecified default timezone shall  
7346 be used.
- 7347 **ASYNCHRONOUS EVENTS**
- 7348 Default.
- 7349 **STDOUT**
- 7350 When standard input is a terminal, prompts of unspecified format for each line of the user input  
7351 described in the STDIN section may be written to standard output.
- 7352 **STDERR**
- 7353 The following shall be written to standard error when a job has been successfully submitted:
- 7354 "job %s at %s\n", *at\_job\_id*, <date>
- 7355 where *date* shall be equivalent in format to the output of:
- 7356 date +"%a %b %e %T %Y"
- 7357 The date and time written shall be adjusted so that they appear in the timezone of the user (as  
7358 determined by the *TZ* variable).
- 7359 Neither this, nor warning messages concerning the selection of the command interpreter, are  
7360 considered a diagnostic that changes the exit status.
- 7361 Diagnostic messages, if any, shall be written to standard error.
- 7362 **OUTPUT FILES**
- 7363 None.
- 7364 **EXTENDED DESCRIPTION**
- 7365 None.
- 7366 **EXIT STATUS**
- 7367 The following exit values shall be returned:
- 7368 0 Successful completion.
- 7369 >0 An error occurred.
- 7370 **CONSEQUENCES OF ERRORS**
- 7371 The job shall not be scheduled.

7372 **APPLICATION USAGE**

7373 It may be useful to redirect standard output within the specified commands.

7374 **EXAMPLES**

7375 1. This sequence can be used at a terminal:

```
7376 batch
7377 sort < file >outfile
7378 EOT
```

7379 2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a  
7380 command procedure (the sequence of output redirection specifications is significant):7381 

```
batch <<! diff file1 file2 2>&1 >outfile | mailx mygroup !
```

7382 **RATIONALE**

7383 Early proposals described *batch* in a manner totally separated from *at*, even though the historical  
7384 model treated it almost as a synonym for *at -qb*. A number of features were added to list and  
7385 control batch work separately from those in *at*. Upon further reflection, it was decided that the  
7386 benefit of this did not merit the change to the historical interface.

7387 The *-m* option was included on the equivalent *at* command because it is historical practice to  
7388 mail results to the submitter, even if all job-produced output is redirected. As explained in the  
7389 RATIONALE for *at*, the **now** keyword submits the job for immediate execution (after scheduling  
7390 delays), despite some historical systems where *at now* would have been considered an error.

7391 **FUTURE DIRECTIONS**

7392 None.

7393 **SEE ALSO**7394 *at*7395 **CHANGE HISTORY**

7396 First released in Issue 2.

7397 **Issue 4**7398 Format reorganized and separated from the *at* description.

7399 Aligned with the ISO/IEC 9945-2:1993 standard.

7400 **Issue 6**

7401 This utility is now marked as part of the User Portability Utilities option.

7402 The NAME is changed to align with the IEEE P1003.2b draft standard.

7403 The normative text is reworded to avoid use of the term “must” for application requirements.

7404 **NAME**7405 `bc` — arbitrary-precision arithmetic language7406 **SYNOPSIS**7407 `bc [-l] [file ...]`7408 **DESCRIPTION**

7409 The `bc` utility shall implement an arbitrary precision calculator. It shall take input from any files  
 7410 given, then read from the standard input. If the standard input and standard output to `bc` are  
 7411 attached to a terminal, the invocation of `bc` shall be considered to be *interactive*, causing  
 7412 behavioral constraints described in the following sections.

7413 **OPTIONS**

7414 The `bc` utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,  
 7415 Utility Syntax Guidelines.

7416 The following option shall be supported:

7417 `-l` (The letter ell.) Define the math functions and initialize *scale* to 20, instead of the  
 7418 default zero; see the EXTENDED DESCRIPTION section.

7419 **OPERANDS**

7420 The following operand shall be supported:

7421 *file* A path name of a text file containing `bc` program statements. After all *files* have  
 7422 been read, `bc` shall read the standard input.

7423 **STDIN**

7424 See the INPUT FILES section.

7425 **INPUT FILES**

7426 Input files shall be text files containing a sequence of comments, statements, and function  
 7427 definitions that shall be executed as they are read.

7428 **ENVIRONMENT VARIABLES**7429 The following environment variables shall affect the execution of `bc`:

7430 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 7431 If *LANG* is unset or null, the corresponding value from the implementation-  
 7432 defined default locale shall be used. If any of the internationalization variables  
 7433 contains an invalid setting, the utility shall behave as if none of the variables had  
 7434 been defined.

7435 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 7436 internationalization variables.

7437 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 7438 characters (for example, single-byte as opposed to multi-byte characters in  
 7439 arguments and input files).

7440 *LC\_MESSAGES*

7441 Determine the locale that should be used to affect the format and contents of  
 7442 diagnostic messages written to standard error.

7443 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

7444 **ASYNCHRONOUS EVENTS**

7445 Default.

7446 **STDOUT**

7447 The output of the *bc* utility shall be controlled by the program read, and consist of zero or more  
 7448 lines containing the value of all executed expressions without assignments. The radix and  
 7449 precision of the output shall be controlled by the values of the **obase** and **scale** variables; see the  
 7450 EXTENDED DESCRIPTION section.

7451 **STDERR**

7452 Used only for diagnostic messages.

7453 **OUTPUT FILES**

7454 None.

7455 **EXTENDED DESCRIPTION**7456 **Grammar**

7457 The grammar in this section and the lexical conventions in the following section shall together  
 7458 describe the syntax for *bc* programs. The general conventions for this style of grammar are  
 7459 described in Section 1.10 (on page 2223). A valid program can be represented as the non-  
 7460 terminal symbol **program** in the grammar. This formal syntax shall take precedence over the text  
 7461 syntax description.

```

7462 %token EOF NEWLINE STRING LETTER NUMBER
7463 %token MUL_OP
7464 /* '*' , '/' , '%' */
7465 %token ASSIGN_OP
7466 /* '=' , '+=' , '-=' , '*=' , '/=' , '%=' , '^=' */
7467 %token REL_OP
7468 /* '==' , '<=' , '>=' , '!=' , '<' , '>' */
7469 %token INCR_DECR
7470 /* '++' , '--' */
7471 %token Define Break Quit Length
7472 /* 'define' , 'break' , 'quit' , 'length' */
7473 %token Return For If While Sqrt
7474 /* 'return' , 'for' , 'if' , 'while' , 'sqrt' */
7475 %token Scale Ibase Obase Auto
7476 /* 'scale' , 'ibase' , 'obase' , 'auto' */
7477 %start program
7478 %%
7479 program : EOF
7480 | input_item program
7481 ;
7482 input_item : semicolon_list NEWLINE
7483 | function
7484 ;
7485 semicolon_list : /* empty */
7486 | statement
7487 | semicolon_list ';' statement
7488 | semicolon_list ';'

```

```

7489 ;
7490 statement_list : /* empty */
7491 | statement
7492 | statement_list NEWLINE
7493 | statement_list NEWLINE statement
7494 | statement_list ';'
7495 | statement_list ';' statement
7496 ;
7497 statement : expression
7498 | STRING
7499 | Break
7500 | Quit
7501 | Return
7502 | Return '(' return_expression ')'
7503 | For '(' expression ';'
7504 relational_expression ';'
7505 expression ')' statement
7506 | If '(' relational_expression ')' statement
7507 | While '(' relational_expression ')' statement
7508 | '{' statement_list '}'
7509 ;
7510 function : Define LETTER '(' opt_parameter_list ')'
7511 | '{' NEWLINE opt_auto_define_list
7512 statement_list '}'
7513 ;
7514 opt_parameter_list : /* empty */
7515 | parameter_list
7516 ;
7517 parameter_list : LETTER
7518 | define_list ',' LETTER
7519 ;
7520 opt_auto_define_list : /* empty */
7521 | Auto define_list NEWLINE
7522 | Auto define_list ';'
7523 ;
7524 define_list : LETTER
7525 | LETTER '[' ']'
7526 | define_list ',' LETTER
7527 | define_list ',' LETTER '[' ']'
7528 ;
7529 opt_argument_list : /* empty */
7530 | argument_list
7531 ;
7532 argument_list : expression
7533 | LETTER '[' ']' ',' argument_list"
7534 ;

```

```

7535 relational_expression : expression
7536 | expression REL_OP expression
7537 ;
7538 return_expression : /* empty */
7539 | expression
7540 ;
7541 expression : named_expression
7542 | NUMBER
7543 | '(' expression ')'
7544 | LETTER '(' opt_argument_list ')'
7545 | '-' expression
7546 | expression '+' expression
7547 | expression '-' expression
7548 | expression MUL_OP expression
7549 | expression '^' expression
7550 | INCR_DECR named_expression
7551 | named_expression INCR_DECR
7552 | named_expression ASSIGN_OP expression
7553 | Length '(' expression ')'
7554 | Sqrt '(' expression ')'
7555 | Scale '(' expression ')'
7556 ;
7557 named_expression : LETTER
7558 | LETTER '[' expression ']'
7559 | Scale
7560 | Ibase
7561 | Obase
7562 ;

```

### 7563 Lexical Conventions in bc

7564 The lexical conventions for *bc* programs, with respect to the preceding grammar, shall be as  
7565 follows:

- 7566 1. Except as noted, *bc* shall recognize the longest possible token or delimiter beginning at a  
7567 given point.
- 7568 2. A comment shall consist of any characters beginning with the two adjacent characters  
7569 `"/*"` and terminated by the next occurrence of the two adjacent characters `"*/"`.  
7570 Comments shall have no effect except to delimit lexical tokens.
- 7571 3. The `<newline>` character shall be recognized as the token **NEWLINE**.
- 7572 4. The token **STRING** shall represent a string constant; it shall consist of any characters  
7573 beginning with the double-quote character (`'"`) and terminated by another occurrence of  
7574 the double-quote character. The value of the string is the sequence of all characters  
7575 between, but not including, the two double-quote characters. All characters shall be taken  
7576 literally from the input, and there is no way to specify a string containing a double-quote  
7577 character. The length of the value of each string shall be limited to `{BC_STRING_MAX}`  
7578 bytes.
- 7579 5. A `<blank>` character shall have no effect except as an ordinary character if it appears  
7580 within a **STRING** token, or to delimit a lexical token other than **STRING**.

- 7581 6. The combination of a backslash character immediately followed by a <newline> character  
 7582 shall have no effect other than to delimit lexical tokens with the following exceptions:
- 7583 • It shall be interpreted as the character sequence "\<newline>" in **STRING** tokens.
  - 7584 • It shall be ignored as part of a multi-line **NUMBER** token.
- 7585 7. The token **NUMBER** shall represent a numeric constant. It shall be recognized by the  
 7586 following grammar:
- ```

7587 NUMBER : integer
7588         | '.' integer
7589         | integer '.'
7590         | integer '.' integer
7591         ;

7592 integer : digit
7593         | integer digit
7594         ;

7595 digit   : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
7596         | 8 | 9 | A | B | C | D | E | F
7597         ;

```
- 7598 8. The value of a **NUMBER** token shall be interpreted as a numeral in the base specified by
 7599 the value of the internal register **ibase** (described below). Each of the **digit** characters shall
 7600 have the value from 0 to 15 in the order listed here, and the period character shall represent
 7601 the radix point. The behavior is undefined if digits greater than or equal to the value of
 7602 **ibase** appear in the token. However, note the exception for single-digit values being
 7603 assigned to **ibase** and **obase** themselves, in **Operations in bc** (on page 2412).
- 7604 9. The following keywords shall be recognized as tokens:
- | | | | | | |
|------|---------------|--------------|---------------|---------------|--------------|
| 7605 | auto | ibase | length | return | while |
| 7606 | break | if | obase | scale | |
| 7607 | define | for | quit | sqrt | |
- 7608 10. Any of the following characters occurring anywhere except within a keyword shall be
 7609 recognized as the token **LETTER**:
- ```

7610 a b c d e f g h i j k l m n o p q r s t u v w x y z

```
- 7611 11. The following single-character and two-character sequences shall be recognized as the  
 7612 token **ASSIGN\_OP**:
- ```

7613 = += -= *= /= %= ^=

```
- 7614 12. If an '=' character, as the beginning of a token, is followed by a '-' character with no
 7615 intervening delimiter, the behavior is undefined.
- 7616 13. The following single-characters shall be recognized as the token **MUL_OP**:
- ```

7617 * / %

```
- 7618 14. The following single-character and two-character sequences shall be recognized as the  
 7619 token **REL\_OP**:
- ```

7620 == <= >= != < >

```
- 7621 15. The following two-character sequences shall be recognized as the token **INCR_DECR**:

7622 ++ --

7623 16. The following single characters shall be recognized as tokens whose names are the
7624 character:

7625 <newline> () , + - ; [] ^ { }

7626 17. The token **EOF** is returned when the end of input is reached.

7627 **Operations in bc**

7628 There are three kinds of identifiers: ordinary identifiers, array identifiers, and function
7629 identifiers. All three types consist of single lowercase letters. Array identifiers shall be followed
7630 by square brackets ("[]"). An array subscript is required except in an argument or auto list.
7631 Arrays are singly dimensioned and can contain up to {BC_DIM_MAX} elements. Indexing shall
7632 begin at zero so an array is indexed from 0 to {BC_DIM_MAX}-1. Subscripts shall be truncated
7633 to integers. The application shall ensure that function identifiers are followed by parentheses,
7634 possibly enclosing arguments. The three types of identifiers do not conflict.

7635 The following table summarizes the rules for precedence and associativity of all operators.
7636 Operators on the same line shall have the same precedence; rows are in order of decreasing
7637 precedence.

7638 **Table 4-3 Operators in bc**

Operator	Associativity
++, --	N/A
unary -	N/A
^	Right to left
*, /, %	Left to right
+, binary -	Left to right
=, +=, -=, *=, /=, %=, ^=	Right to left
==, <=, >=, !=, <, >	None

7647 Each expression or named expression has a *scale*, which is the number of decimal digits that
7648 shall be maintained as the fractional portion of the expression.

7649 *Named expressions* are places where values are stored. Named expressions shall be valid on the
7650 left side of an assignment. The value of a named expression shall be the value stored in the place
7651 named. Simple identifiers and array elements are named expressions; they have an initial value
7652 of zero and an initial scale of zero.

7653 The internal registers **scale**, **ibase**, and **obase** are all named expressions. The scale of an
7654 expression consisting of the name of one of these registers shall be zero; values assigned to any
7655 of these registers are truncated to integers. The **scale** register shall contain a global value used in
7656 computing the scale of expressions (as described below). The value of the register **scale** is
7657 limited to $0 \leq \text{scale} \leq \{\text{BC_SCALE_MAX}\}$ and shall have a default value of zero. The **ibase** and
7658 **obase** registers are the input and output number radix, respectively. The value of **ibase** shall be
7659 limited to:

7660 $2 \leq \text{ibase} \leq 16$

7661 The value of **obase** shall be limited to:

7662 $2 \leq \text{obase} \leq \{\text{BC_BASE_MAX}\}$

7663 When either **ibase** or **obase** is assigned a single **digit** value from the list in **Lexical Conventions**
7664 **in bc** (on page 2410), the value shall be assumed in hexadecimal. (For example, **ibase=A** sets to

7665 base ten, regardless of the current **ibase** value.) Otherwise, the behavior is undefined when
 7666 digits greater than or equal to the value of **ibase** appear in the input. Both **ibase** and **obase** shall
 7667 have initial values of 10.

7668 Internal computations shall be conducted as if in decimal, regardless of the input and output
 7669 bases, to the specified number of decimal digits. When an exact result is not achieved, (for
 7670 example, **scale**=0; 3.2/1) the result shall be truncated.

7671 For all values of **obase** specified by this volume of IEEE Std. 1003.1-200x, *bc* shall output numeric
 7672 values by performing each of the following steps in order:

- 7673 1. If the value is less than zero, a hyphen ('-') character shall be output.
- 7674 2. One of the following is output, depending on the numerical value:
 - 7675 • If the absolute value of the numerical value is greater than or equal to one, the integer
 7676 portion of the value shall be output as a series of digits appropriate to **obase** (as
 7677 described below) most significant digit first. The most significant non-zero digit shall
 7678 be output next, followed by each successively less significant digit.
 - 7679 • If the absolute value of the numerical value is less than one but greater than zero and
 7680 the scale of the numerical value is greater than zero, it is unspecified whether the
 7681 character 0 is output.
 - 7682 • If the numerical value is zero, the character 0 shall be output.
- 7683 3. If the scale of the value is greater than zero and the numeric value is not zero, a period
 7684 character shall be output, followed by a series of digits appropriate to **obase** (as described
 7685 below) representing the most significant portion of the fractional part of the value. If *s*
 7686 represents the scale of the value being output, the number of digits output shall be *s* if
 7687 **obase** is 10, less than or equal to *s* if **obase** is greater than 10, or greater than or equal to *s* if
 7688 **obase** is less than 10. For **obase** values other than 10, this should be the number of digits
 7689 needed to represent a precision of 10^s .

7690 For **obase** values from 2 to 16, valid digits are the first **obase** of the single characters:

7691 0 1 2 3 4 5 6 7 8 9 A B C D E F

7692 which represent the values zero to 15, inclusive, respectively.

7693 For bases greater than 16, each digit shall be written as a separate multi-digit decimal number.
 7694 Each digit except the most significant fractional digit shall be preceded by a single <space>
 7695 character. For bases from 17 to 100, *bc* shall write two-digit decimal numbers; for bases from 101
 7696 to 1 000, three-digit decimal strings, and so on. For example, the decimal number 1 024 in base 25
 7697 would be written as:

7698 Δ01Δ15Δ24

7699 in base 125, as:

7700 Δ008Δ024

7701 Very large numbers shall be split across lines with 70 characters per line in the POSIX locale;
 7702 other locales may split at different character boundaries. Lines that are continued shall end with
 7703 a backslash ('\').

7704 A function call shall consist of a function name followed by parentheses containing a comma-
 7705 separated list of expressions, which are the function arguments. A whole array passed as an
 7706 argument shall be specified by the array name followed by empty square brackets. All function
 7707 arguments shall be passed by value. As a result, changes made to the formal parameters shall
 7708 have no effect on the actual arguments. If the function terminates by executing a **return**

7709 statement, the value of the function shall be the value of the expression in the parentheses of the
 7710 **return** statement or shall be zero if no expression is provided or if there is no **return** statement.

7711 The result of **sqrt**(*expression*) shall be the square root of the expression. The result shall be
 7712 truncated in the least significant decimal place. The scale of the result shall be the scale of the
 7713 expression or the value of **scale**, whichever is larger.

7714 The result of **length**(*expression*) shall be the total number of significant decimal digits in the
 7715 expression. The scale of the result shall be zero.

7716 The result of **scale**(*expression*) shall be the scale of the expression. The scale of the result shall be
 7717 zero.

7718 A numeric constant shall be an expression. The scale shall be the number of digits that follow the
 7719 radix point in the input representing the constant, or zero if no radix point appears.

7720 The sequence (*expression*) shall be an expression with the same value and scale as *expression*.
 7721 The parentheses can be used to alter the normal precedence.

7722 The semantics of the unary and binary operators are as follows:

7723 *-expression*
 7724 The result shall be the negative of the *expression*. The scale of the result shall be the scale of
 7725 *expression*.

7726 The unary increment and decrement operators shall not modify the scale of the named
 7727 expression upon which they operate. The scale of the result shall be the scale of that named
 7728 expression.

7729 *++named-expression*
 7730 The named expression shall be incremented by one. The result shall be the value of the
 7731 named expression after incrementing.

7732 *--named-expression*
 7733 The named expression shall be decremented by one. The result shall be the value of the
 7734 named expression after decrementing.

7735 *named-expression++*
 7736 The named expression shall be incremented by one. The result shall be the value of the
 7737 named expression before incrementing.

7738 *named-expression--*
 7739 The named expression shall be decremented by one. The result shall be the value of the
 7740 named expression before decrementing.

7741 The exponentiation operator, circumflex ('^ '), shall bind right to left.

7742 *expression^expression*
 7743 The result shall be the first *expression* raised to the power of the second *expression*. If the
 7744 second expression is not an integer, the behavior is undefined. If *a* is the scale of the left
 7745 expression and *b* is the absolute value of the right expression, the scale of the result shall be:
 7746 if $b \geq 0$ $\min(a * b, \max(\text{scale}, a))$ if $b < 0$ *scale*

7747 The multiplicative operators (' * ' , ' / ' , ' % ') shall bind left to right.

7748 *expression*expression*
 7749 The result shall be the product of the two expressions. If *a* and *b* are the scales of the two
 7750 expressions, then the scale of the result shall be:

7751 $\min(a+b, \max(\text{scale}, a, b))$

7752 *expression/expression*
 7753 The result shall be the quotient of the two expressions. The scale of the result shall be the
 7754 value of **scale**.

7755 *expression%expression*
 7756 For expressions *a* and *b*, *a%b* shall be evaluated equivalent to the steps:

- 7757 1. Compute *a/b* to current scale.
- 7758 2. Use the result to compute:

7759 $a - (a / b) * b$
 7760 to scale:
 7761 $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

7762 The scale of the result shall be:
 7763 $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

7764 When **scale** is zero, the '*%*' operator is the mathematical remainder operator.

7765 The additive operators ('+', '-') shall bind left to right.

7766 *expression+expression*
 7767 The result shall be the sum of the two expressions. The scale of the result shall be the
 7768 maximum of the scales of the expressions.

7769 *expression-expression*
 7770 The result shall be the difference of the two expressions. The scale of the result shall be the
 7771 maximum of the scales of the expressions.

7772 The assignment operators ('=', '+=', '-=', '*=', '/=', '%=', '^=') shall bind right to left.

7773 *named-expression=expression*
 7774 This expression results in assigning the value of the expression on the right to the named
 7775 expression on the left. The scale of both the named expression and the result shall be the
 7776 scale of *expression*.

7777 The compound assignment forms:

7778 *named-expression <operator>= expression*
 7779 shall be equivalent to:
 7780 *named-expression=named-expression <operator> expression*
 7781 except that the *named-expression* shall be evaluated only once.

7782 Unlike all other operators, the relational operators ('<', '>', '<=', '>=', '==', '!=') shall be
 7783 only valid as the object of an **if**, **while**, or inside a **for** statement.

7784 *expression1<expression2*
 7785 The relation shall be true if the value of *expression1* is strictly less than the value of
 7786 *expression2*.

7787 *expression1>expression2*
 7788 The relation shall be true if the value of *expression1* is strictly greater than the value of
 7789 *expression2*.

7790 *expression1* <= *expression2*
 7791 The relation shall be true if the value of *expression1* is less than or equal to the value of
 7792 *expression2*.

7793 *expression1* >= *expression2*
 7794 The relation shall be true if the value of *expression1* is greater than or equal to the value of
 7795 *expression2*.

7796 *expression1* = *expression2*
 7797 The relation shall be true if the values of *expression1* and *expression2* are equal.

7798 *expression1* != *expression2*
 7799 The relation shall be true if the values of *expression1* and *expression2* are unequal.

7800 There are only two storage classes in *bc*, global and automatic (local). Only identifiers that are
 7801 local to a function need be declared with the **auto** command. The arguments to a function shall
 7802 be local to the function. All other identifiers are assumed to be global and available to all
 7803 functions. All identifiers, global and local, have initial values of zero. Identifiers declared as auto
 7804 shall be allocated on entry to the function and released on returning from the function. They
 7805 therefore do not retain values between function calls. Auto arrays shall be specified by the array
 7806 name followed by empty square brackets. On entry to a function, the old values of the names
 7807 that appear as parameters and as automatic variables shall be pushed onto a stack. Until the
 7808 function returns, reference to these names shall refer only to the new values.

7809 References to any of these names from other functions that are called from this function also
 7810 refer to the new value until one of those functions uses the same name for a local variable.

7811 When a statement is an expression, unless the main operator is an assignment, execution of the
 7812 statement shall write the value of the expression followed by a <newline> character.

7813 When a statement is a string, execution of the statement shall write the value of the string.

7814 Statements separated by semicolons or <newline> characters shall be executed sequentially. In
 7815 an interactive invocation of *bc*, each time a <newline> character is read that satisfies the
 7816 grammatical production:

7817 `input_item : semicolon_list NEWLINE`

7818 the sequential list of statements making up the **semicolon_list** shall be executed immediately
 7819 and any output produced by that execution shall be written without any delay due to buffering.

7820 In an **if** statement (**if**(*relation*) *statement*), the *statement* shall be executed if the relation is true.

7821 The **while** statement (**while**(*relation*) *statement*) implements a loop in which the *relation* is tested;
 7822 each time the *relation* is true, the *statement* shall be executed and the *relation* retested. When the
 7823 *relation* is false, execution shall resume after *statement*.

7824 A **for** statement (**for**(*expression*; *relation*; *expression*) *statement*) shall be the same as:

7825 `first-expression`
 7826 `while (relation) {`
 7827 `statement`
 7828 `last-expression`
 7829 `}`

7830 The application shall ensure that all three expressions are present.

7831 The **break** statement shall cause termination of a **for** or **while** statement.

7832 The **auto** statement (**auto** *identifier* [*identifier*] ...) shall cause the values of the identifiers to be
 7833 pushed down. The identifiers can be ordinary identifiers or array identifiers. Array identifiers

7834 shall be specified by following the array name by empty square brackets. The application shall
7835 ensure that the **auto** statement is the first statement in a function definition.

7836 A **define** statement:

```
7837 define LETTER ( opt_parameter_list ) {
7838     opt_auto_define_list
7839     statement_list
7840 }
```

7841 defines a function named **LETTER**. If a function named **LETTER** was previously defined, the
7842 **define** statement shall replace the previous definition. The expression:

```
7843 LETTER ( opt_argument_list )
```

7844 shall invoke the function named **LETTER**. The behavior is undefined if the number of
7845 arguments in the invocation does not match the number of parameters in the definition.
7846 Functions shall be defined before they are invoked. A function shall be considered to be defined
7847 within its own body, so recursive calls are valid. The values of numeric constants within a
7848 function shall be interpreted in the base specified by the value of the **ibase** register when the
7849 function is invoked.

7850 The **return** statements (**return** and **return(expression)**) shall cause termination of a function,
7851 popping of its auto variables, and specification of the result of the function. The first form shall
7852 be equivalent to **return(0)**. The value and scale of the result returned by the function shall be the
7853 value and scale of the expression returned.

7854 The **quit** statement (**quit**) shall stop execution of a *bc* program at the point where the statement
7855 occurs in the input, even if it occurs in a function definition, or in an **if**, **for**, or **while** statement.

7856 The following functions shall be defined when the **-l** option is specified:

```
7857 s( expression )
7858     Sine of argument in radians.
```

```
7859 c( expression )
7860     Cosine of argument in radians.
```

```
7861 a( expression )
7862     Arctangent of argument.
```

```
7863 l( expression )
7864     Natural logarithm of argument.
```

```
7865 e( expression )
7866     Exponential function of argument.
```

```
7867 j( expression, expression )
7868     Bessel function of integer order.
```

7869 The scale of the result returned by these functions shall be the value of the **scale** register at the
7870 time the function is invoked. The value of the **scale** register after these functions have completed
7871 their execution shall be the same value it had upon invocation. The behavior is undefined if any
7872 of these functions is invoked with an argument outside the domain of the mathematical
7873 function.

7874 EXIT STATUS

7875 The following exit values shall be returned:

7876 0 All input files were processed successfully.

7877 *unspecified* An error occurred.

7878 CONSEQUENCES OF ERRORS

7879 If any *file* operand is specified and the named file cannot be accessed, *bc* shall write a diagnostic message to standard error and terminate without any further action.

7881 In an interactive invocation of *bc*, the utility should print an error message and recover following any error in the input. In a non-interactive invocation of *bc*, invalid input causes undefined behavior.

7884 APPLICATION USAGE

7885 Automatic variables in *bc* do not work in exactly the same way as in either C or PL/1.

7886 For historical reasons, the exit status from *bc* cannot be relied upon to indicate that an error has occurred. Returning zero after an error is possible. Therefore, *bc* should be used primarily by interactive users (who can react to error messages) or by application programs that can somehow validate the answers returned as not including error messages.

7890 The *bc* utility always uses the period (`.`) character to represent a radix point, regardless of any decimal-point character specified as part of the current locale. In languages like C or *awk*, the period character is used in program source, so it can be portable and unambiguous, while the locale-specific character is used in input and output. Because there is no distinction between source and input in *bc*, this arrangement would not be possible. Using the locale-specific character in *bc*'s input would introduce ambiguities into the language; consider the following example in a locale with a comma as the decimal-point character:

```
7897 define f(a,b) {
7898     ...
7899 }
7900 ...
7901 f(1,2,3)
```

7902 Because of such ambiguities, the period character is used in input. Having input follow different conventions from output would be confusing in either pipeline usage or interactive usage, so the period is also used in output.

7905 EXAMPLES

7906 In the shell, the following assigns an approximation of the first ten digits of ' π ' to the variable *x*:

```
7908 x=$(printf "%s\n" 'scale = 10; 104348/33215' | bc)
```

7909 The following *bc* program prints the same approximation of ' π ', with a label, to standard output:

```
7911 scale = 10
7912 "pi equals "
7913 104348 / 33215
```

7914 The following defines a function to compute an approximate value of the exponential function (note that such a function is predefined if the `-l` option is specified):

```
7916 scale = 20
7917 define e(x){
7918     auto a, b, c, i, s
7919     a = 1
7920     b = 1
7921     s = 1
```

```

7922         for (i = 1; 1 == 1; i++){
7923             a = a*x
7924             b = b*i
7925             c = a/b
7926             if (c == 0) {
7927                 return(s)
7928             }
7929             s = s+c
7930         }
7931     }

```

7932 The following prints approximate values of the exponential function of the first ten integers:

```

7933     for (i = 1; i <= 10; ++i) {
7934         e(i)
7935     }

```

7936 RATIONALE

7937 The *bc* utility is implemented historically as a front-end processor for *dc*; *dc* was not selected to
 7938 be part of this volume of IEEE Std. 1003.1-200x because *bc* was thought to have a more intuitive
 7939 programmatic interface. Current implementations that implement *bc* using *dc* are expected to be
 7940 compliant.

7941 The exit status for error conditions has been left unspecified for several reasons:

- 7942 • The *bc* utility is used in both interactive and non-interactive situations. Different exit codes
 7943 may be appropriate for the two uses.
- 7944 • It is unclear when a non-zero exit should be given; divide-by-zero, undefined functions, and
 7945 syntax errors are all possibilities.
- 7946 • It is not clear what utility the exit status has.
- 7947 • In the 4.3 BSD, System V, and Ninth Edition implementations, *bc* works in conjunction with
 7948 *dc*. The *dc* utility is the parent, *bc* is the child. This was done to cleanly terminate *bc* if *dc*
 7949 aborted.

7950 The decision to have *bc* exit upon encountering an inaccessible input file is based on the belief
 7951 that *bc file1 file2* is used most often when at least *file1* contains data/function
 7952 declarations/initializations. Having *bc* continue with prerequisite files missing is probably not
 7953 useful. There is no implication in the CONSEQUENCES OF ERRORS section that *bc* must check
 7954 all its files for accessibility before opening any of them.

7955 There was considerable debate on the appropriateness of the language accepted by *bc*. Several
 7956 reviewers preferred to see either a pure subset of the C language or some changes to make the
 7957 language more compatible with C. While the *bc* language has some obvious similarities to C, it
 7958 has never claimed to be compatible with any version of C. An interpreter for a subset of C might
 7959 be a very worthwhile utility, and it could potentially make *bc* obsolete. However, no such utility
 7960 is known in historical practice, and it was not within the scope of this volume of
 7961 IEEE Std. 1003.1-200x to define such a language and utility. If and when they are defined, it may
 7962 be appropriate to include them in a future version of this volume of IEEE Std. 1003.1-200x. This
 7963 left the following alternatives:

- 7964 1. Exclude any calculator language from this volume of IEEE Std. 1003.1-200x.

7965 The consensus of the standard developers was that a simple programmatic calculator
 7966 language is very useful for both applications and interactive users. The only arguments for
 7967 excluding any calculator were that it would become obsolete if and when a C-compatible

7968 one emerged, or that the absence would encourage the development of such a C-
 7969 compatible one. These arguments did not sufficiently address the needs of current
 7970 application writers.

7971 2. Standardize the historical *dc*, possibly with minor modifications.

7972 The consensus of the standard developers was that *dc* is a fundamentally less usable
 7973 language and that that would be far too severe a penalty for avoiding the issue of being
 7974 similar to but incompatible with C.

7975 3. Standardize the historical *bc*, possibly with minor modifications.

7976 This was the approach taken. Most of the proponents of changing the language would not
 7977 have been satisfied until most or all of the incompatibilities with C were resolved. Since
 7978 most of the changes considered most desirable would break historical applications and
 7979 require significant modification to historical implementations, almost no modifications
 7980 were made. The one significant modification that was made was the replacement of the
 7981 historical *bc* assignment operators "=", and so on, with the more modern "+=", and so
 7982 on. The older versions are considered to be fundamentally flawed because of the lexical
 7983 ambiguity in uses like $a=-1$.

7984 In order to permit implementations to deal with backwards compatibility as they see fit,
 7985 the behavior of this one ambiguous construct was made undefined. (At least three
 7986 implementations have been known to support this change already, so the degree of change
 7987 involved should not be great.)

7988 The '%' operator is the mathematical remainder operator when **scale** is zero. The behavior of
 7989 this operator for other values of **scale** is from historical implementations of *bc*, and has been
 7990 maintained for the sake of historical applications despite its non-intuitive nature.

7991 Historical implementations permit setting **ibase** and **obase** to a broader range of values. This
 7992 includes values less than 2, which were not seen as sufficiently useful to standardize. These
 7993 implementations do not interpret input properly for values of **ibase** that are greater than 16. This
 7994 is because numeric constants are recognized syntactically, rather than lexically, as described in
 7995 this volume of IEEE Std. 1003.1-200x. They are built from lexical tokens of single hexadecimal
 7996 digits and periods. Since <blank>s between tokens are not visible at the syntactic level, it is not
 7997 possible to recognize the multi-digit "digits" used in the higher bases properly. The ability to
 7998 recognize input in these bases was not considered useful enough to require modifying these
 7999 implementations. Note that the recognition of numeric constants at the syntactic level is not a
 8000 problem with conformance to this volume of IEEE Std. 1003.1-200x, as it does not impact the
 8001 behavior of portable applications (and correct *bc* programs). Historical implementations also
 8002 accept input with all of the digits '0'-'9' and 'A'-'F' regardless of the value of **ibase**; since
 8003 digits with value greater than or equal to **ibase** are not really appropriate, the behavior when
 8004 they appear is undefined, except for the common case of:

```
8005 ibase=8;
8006     /* Process in octal base. */
8007     ...
8008 ibase=A
8009     /* Restore decimal base. */
```

8010 In some historical implementations, if the expression to be written is an uninitialized array
 8011 element, a leading <space> character and/or up to four leading 0 characters may be output
 8012 before the character zero. This behavior is considered a bug; it is unlikely that any currently
 8013 portable application relies on:

- 8014 `echo 'b[3]' | bc`
- 8015 returning 0000 rather than 0.
- 8016 Exact calculation of the number of fractional digits to output for a given value in a base other
8017 than 10 can be computationally expensive. Historical implementations use a faster
8018 approximation, and this is permitted. Note that the requirements apply only to values of **obase**
8019 that this volume of IEEE Std. 1003.1-200x requires implementations to support (in particular, not
8020 to 1, 0, or negative bases, if an implementation supports them as an extension).
- 8021 Historical implementations of *bc* did not allow array parameters to be passed as the last
8022 parameter to a function. New implementations are encouraged to remove this restriction even
8023 though it is not required by the grammar.
- 8024 **FUTURE DIRECTIONS**
- 8025 None.
- 8026 **SEE ALSO**
- 8027 *awk*
- 8028 **CHANGE HISTORY**
- 8029 First released in Issue 4.
- 8030 **Issue 5**
- 8031 FUTURE DIRECTIONS section added.
- 8032 **Issue 6**
- 8033 Updated to align with the IEEE P1003.2b draft standard, which included resolution of several
8034 interpretations of the ISO POSIX-2: 1993 standard.
- 8035 The normative text is reworded to avoid use of the term “must” for application requirements.

8036 **NAME**

8037 bg — run jobs in the background

8038 **SYNOPSIS**8039 UP bg [*job_id* ...]

8040

8041 **DESCRIPTION**

8042 If job control is enabled (see the description of *set -m*), the *bg* utility shall resume suspended jobs
 8043 from the current environment (see Section 2.13 (on page 2273)) by running them as background
 8044 jobs. If the job specified by *job_id* is already a running background job, the *bg* utility shall have no
 8045 effect and shall exit successfully.

8046 Using *bg* to place a job into the background shall cause its process ID to become “known in the
 8047 current shell execution environment”, as if it had been started as an asynchronous list; see
 8048 Section 2.9.3.1 (on page 2259).

8049 **OPTIONS**

8050 None.

8051 **OPERANDS**

8052 The following operand shall be supported:

8053 *job_id* Specify the job to be resumed as a background job. If no *job_id* operand is given,
 8054 the most recently suspended job shall be used. The format of *job_id* is described in
 8055 the Base Definitions volume of IEEE Std. 1003.1-200x, Section 3.205, Job Control
 8056 Job ID.

8057 **STDIN**

8058 Not used.

8059 **INPUT FILES**

8060 None.

8061 **ENVIRONMENT VARIABLES**8062 The following environment variables shall affect the execution of *bg*:

8063 *LANG* Provide a default value for the internationalization variables that are unset or null.
 8064 If *LANG* is unset or null, the corresponding value from the implementation-
 8065 defined default locale shall be used. If any of the internationalization variables
 8066 contains an invalid setting, the utility shall behave as if none of the variables had
 8067 been defined.

8068 *LC_ALL* If set to a non-empty string value, override the values of all the other
 8069 internationalization variables.

8070 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 8071 characters (for example, single-byte as opposed to multi-byte characters in
 8072 arguments).

8073 *LC_MESSAGES*

8074 Determine the locale that should be used to affect the format and contents of
 8075 diagnostic messages written to standard error.

8076 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

8077 **ASYNCHRONOUS EVENTS**

8078 Default.

8079 **STDOUT**8080 The output of *bg* shall consist of a line in the format:8081 "[%d] %s\n", <*job-number*>, <*command*>

8082 where the fields are as follows:

8083 <*job-number*> A number that can be used to identify the job to the *wait*, *fg*, and *kill* utilities. Using
8084 these utilities, the job can be identified by prefixing the job number with '% '.8085 <*command*> The associated command that was given to the shell.8086 **STDERR**

8087 Used only for diagnostic messages.

8088 **OUTPUT FILES**

8089 None.

8090 **EXTENDED DESCRIPTION**

8091 None.

8092 **EXIT STATUS**

8093 The following exit values shall be returned:

8094 0 Successful completion.

8095 >0 An error occurred.

8096 **CONSEQUENCES OF ERRORS**8097 If job control is disabled, the *bg* utility shall exit with an error and no job shall be placed in the
8098 background.8099 **APPLICATION USAGE**8100 A job is generally suspended by typing the SUSP character (<control>-Z on most systems); see
8101 the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface. At
8102 that point, *bg* can put the job into the background. This is most effective when the job is
8103 expecting no terminal input and its output has been redirected to non-terminal files. A
8104 background job can be forced to stop when it has terminal output by issuing the command:8105 `stty tostop`

8106 A background job can be stopped with the command:

8107 `kill -s stop job ID`8108 The *bg* utility does not work as expected when it is operating in its own utility execution
8109 environment because that environment has no suspended jobs. In the following examples:8110 ... | xargs bg
8111 (bg)8112 each *bg* operates in a different environment and does not share its parent shell's understanding
8113 of jobs. For this reason, *bg* is generally implemented as a shell regular built-in.8114 **EXAMPLES**

8115 None.

8116 **RATIONALE**

8117 The extensions to the shell specified in this volume of IEEE Std. 1003.1-200x have mostly been
8118 based on features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs*
8119 are also based on the KornShell. The standard developers examined the characteristics of the C
8120 shell versions of these utilities and found that differences exist. Despite widespread use of the C
8121 shell, the KornShell versions were selected for this volume of IEEE Std. 1003.1-200x to maintain a
8122 degree of uniformity with the rest of the KornShell features selected (such as the very popular
8123 command line editing features).

8124 The *bg* utility is expected to wrap its output if the output exceeds the number of display
8125 columns.

8126 **FUTURE DIRECTIONS**

8127 None.

8128 **SEE ALSO**

8129 *fg*, *kill*, *jobs*, *wait*

8130 **CHANGE HISTORY**

8131 First released in Issue 4.

8132 **Issue 6**

8133 This utility is now marked as part of the User Portability Utilities option.

8134 The JC margin marker on the SYNOPSIS is removed since support for Job Control is mandatory
8135 in this issue. This is a FIPS requirement.

8136 NAME

8137 c99 — compile standard C programs

8138 SYNOPSIS

```
8139 CD c99 [-c][-D name[=value]]...[-E][-g][-I directory] ... [-L directory]
8140 ... [-o outfile][-O][-s][-U name]... operand ...
8141
```

8142 DESCRIPTION

8143 The *c99* utility is an interface to the standard C compilation system; it shall accept source code
 8144 conforming to the ISO C standard. The system conceptually consists of a compiler and link
 8145 editor. The files referenced by *operands* shall be compiled and linked to produce an executable
 8146 file. (It is unspecified whether the linking occurs entirely within the operation of *c99*; some
 8147 systems may produce objects that are not fully resolved until the file is executed.)

8148 If the `-c` option is specified, for all path name operands of the form *file.c*, the files:

8149 `$(basename pathname .c).o`

8150 shall be created as the result of successful compilation. If the `-c` option is not specified, it is
 8151 unspecified whether such `.o` files are created or deleted for the *file.c* operands.

8152 If there are no options that prevent link editing (such as `-c` or `-E`), and all operands compile and
 8153 link without error, the resulting executable file shall be written according to the `-o outfile` option
 8154 (if present) or to the file `a.out`.

8155 The executable file shall be created as specified in Section 1.7.1.4 (on page 2209), except that the
 8156 file permission bits shall be set to:

8157 `S_IRWXO | S_IRWXG | S_IRWXU`

8158 and the bits specified by the *umask* of the process shall be cleared.

8159 OPTIONS

8160 The *c99* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 8161 12.2, Utility Syntax Guidelines, except that:

- 8162 • The `-I library` operands have the format of options, but their position within a list of
 8163 operands affects the order in which libraries are searched.
- 8164 • The order of specifying the `-I` and `-L` options is significant.
- 8165 • Portable applications shall specify each option separately; that is, grouping option letters (for
 8166 example, `-cO`) need not be recognized by all implementations.

8167 The following options shall be supported:

8168 `-c` Suppress the link-edit phase of the compilation, and do not remove any object files
 8169 that are produced.

8170 `-g` Produce symbolic information in the object or executable files; the nature of this
 8171 information is unspecified, and may be modified by implementation-defined
 8172 interactions with other options.

8173 `-s` Produce object or executable files, or both, from which symbolic and other
 8174 information not required for proper execution using the *exec* family defined in the
 8175 System Interfaces volume of IEEE Std. 1003.1-200x, has been removed (stripped). If
 8176 both `-g` and `-s` options are present, the action taken is unspecified.

8177 `-o outfile` Use the path name *outfile*, instead of the default `a.out`, for the executable file
 8178 produced. If the `-o` option is present with `-c` or `-E`, the result is unspecified.

- 8179 **-D** *name*[=*value*]
8180 Define *name* as if by a C-language **#define** directive. If no =*value* is given, a value of
8181 1 shall be used. The **-D** option has lower precedence than the **-U** option. That is, if
8182 *name* is used in both a **-U** and a **-D** option, *name* shall be undefined regardless of
8183 the order of the options. Additional implementation-defined *names* may be
8184 provided by the compiler. Implementations shall support at least 2 048 bytes of **-D**
8185 definitions and 256 *names*.
- 8186 **-E** Copy C-language source files to standard output, expanding all preprocessor
8187 directives; no compilation shall be performed. If any operand is not a text file, the
8188 effects are unspecified.
- 8189 **-I** *directory* Change the algorithm for searching for headers whose names are not absolute path
8190 names to look in the directory named by the *directory* path name before looking in
8191 the usual places. Thus, headers whose names are enclosed in double-quotes (" ")
8192 shall be searched for first in the directory of the file with the **#include** line, then in
8193 directories named in **-I** options, and last in the usual places. For headers whose
8194 names are enclosed in angle brackets ("< >"), the header shall be searched for only
8195 in directories named in **-I** options and then in the usual places. Directories named
8196 in **-I** options shall be searched in the order specified. Implementations shall
8197 support at least ten instances of this option in a single *c99* command invocation.
- 8198 **-L** *directory* Change the algorithm of searching for the libraries named in the **-I** objects to look
8199 in the directory named by the *directory* path name before looking in the usual
8200 places. Directories named in **-L** options shall be searched in the order specified.
8201 Implementations shall support at least ten instances of this option in a single *c99*
8202 command invocation. If a directory specified by a **-L** option contains files named
8203 **libc.a**, **libm.a**, **libl.a**, or **liby.a**, the results are unspecified.
- 8204 **-O** Optimize. The nature of the optimization is unspecified.
- 8205 **-U** *name* Remove any initial definition of *name*.
- 8206 Multiple instances of the **-D**, **-I**, **-U**, and **-L** options can be specified.

8207 OPERANDS

8208 An *operand* is either in the form of a path name or the form **-I** *library*. The application shall
8209 ensure that at least one operand of the path name form is specified. The following operands shall
8210 be supported:

- 8211 *file.c* A C-language source file to be compiled and optionally linked. The application
8212 shall ensure that the operand is of this form if the **-c** option is used.
- 8213 *file.a* A library of object files typically produced by the *ar* utility, and passed directly to
8214 the link editor. Implementations may recognize implementation-defined suffixes
8215 other than **.a** as denoting object file libraries.
- 8216 *file.o* An object file produced by *c99* **-c** and passed directly to the link editor.
8217 Implementations may recognize implementation-defined suffixes other than **.o** as
8218 denoting object files.

8219 The processing of other files is implementation-defined.

8220 **-I** *library* (The letter ell.) Search the library named:

8221 *liblibrary.a*

8222 A library shall be searched when its name is encountered, so the placement of a **-I**
8223 operand is significant. Several standard libraries can be specified in this manner, as

8224 described in the EXTENDED DESCRIPTION section. Implementations may
8225 recognize implementation-defined suffixes other than *.a* as denoting libraries.

8226 **STDIN**

8227 Not used.

8228 **INPUT FILES**

8229 The input file shall be one of the following: a text file containing a C-language source program,
8230 an object file in the format produced by *c99 -c*, or a library of object files, in the format produced
8231 by archiving zero or more object files, using *ar*. Implementations may supply additional utilities
8232 that produce files in these formats. Additional input file formats are implementation-defined.

8233 **ENVIRONMENT VARIABLES**

8234 The following environment variables shall affect the execution of *c99*:

8235 *LANG* Provide a default value for the internationalization variables that are unset or null.
8236 If *LANG* is unset or null, the corresponding value from the implementation-
8237 defined default locale shall be used. If any of the internationalization variables
8238 contains an invalid setting, the utility shall behave as if none of the variables had
8239 been defined.

8240 *LC_ALL* If set to a non-empty string value, override the values of all the other
8241 internationalization variables.

8242 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
8243 characters (for example, single-byte as opposed to multi-byte characters in
8244 arguments and input files).

8245 *LC_MESSAGES*

8246 Determine the locale that should be used to affect the format and contents of
8247 diagnostic messages written to standard error.

8248 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

8249 *TMPDIR* Provide a path name that should override the default directory for temporary files,
8250 XSI if any. On XSI-conforming systems, provide a path name that shall override the
8251 default directory for temporary files, if any.

8252 **ASYNCHRONOUS EVENTS**

8253 Default.

8254 **STDOUT**

8255 If more than one *file* operand ending in *.c* (or possibly other unspecified suffixes) is given, for
8256 each such file:

8257 "%s:\n", *<file>*

8258 may be written. These messages, if written, shall precede the processing of each input file; they
8259 shall not be written to the standard output if they are written to the standard error, as described
8260 in the *STDERR* section.

8261 If the *-E* option is specified, the standard output shall be a text file that represents the results of
8262 the preprocessing stage of the language; it may contain extra information appropriate for
8263 subsequent compilation passes.

8264 **STDERR**

8265 Used only for diagnostic messages. If more than one *file* operand ending in *.c* (or possibly other
8266 unspecified suffixes) is given, for each such file:

8267 "%s:\n", <file>

8268 may be written to allow identification of the diagnostic and warning messages with the
8269 appropriate input file. These messages, if written, shall precede the processing of each input file;
8270 they shall not be written to the standard error if they are written to the standard output, as
8271 described in the STDOUT section.

8272 This utility may produce warning messages about certain conditions that do not warrant
8273 returning an error (non-zero) exit value.

8274 OUTPUT FILES

8275 Object files or executable files or both are produced in unspecified formats.

8276 EXTENDED DESCRIPTION

8277 Standard Libraries

8278 The *c99* utility shall recognize the following **-l** operands for standard libraries:

8279 **-l c** This operand shall make visible all library functions referenced in the System
8280 Interfaces volume of IEEE Std. 1003.1-200x, with the possible exception of those
8281 functions listed as residing in < aio.h>, < arpa/inet.h>, < math.h>, < mqueue.h>,
8282 < netdb.h>, < netinet/in.h>, < pthread.h>, < sched.h>, < semaphore.h>,
8283 < sys/socket.h>, *pthread_atfork()* in < unistd.h>, and those functions marked as an
8284 RT extension in < sys/mman.h> and < time.h>. This operand shall not be required
8285 to be present to cause a search of this library.

8286 **-l l** This operand shall make visible all functions required by the C-language output of
8287 *lex* that are not made available through the **-l c** operand.

8288 **-l pthread** This operand shall make visible all functions referenced in < pthread.h> and
8289 *pthread_atfork()* referenced in < unistd.h>. An implementation may search this
8290 library in the absence of this operand.

8291 **-l m** This operand shall make visible all functions referenced in < math.h>. An
8292 implementation may search this library in the absence of this operand.

8293 **-l rt** This operand shall make visible all functions referenced in < aio.h>, < mqueue.h>,
8294 < sched.h>, and < semaphore.h>, and those functions marked as an RT extension in
8295 < sys/mman.h> and < time.h>. An implementation may search this library in the
8296 absence of this operand.

8297 **-l xnet** This operand makes visible all functions referenced in < arpa/inet.h>, < netdb.h>,
8298 < netinet/in.h>, and < sys/socket.h>. An implementation may search this library in
8299 the absence of this operand.

8300 **-l y** This operand shall make visible all functions required by the C-language output of
8301 *yacc* that are not made available through the **-l c** operand.

8302 In the absence of options that inhibit invocation of the link editor, such as **-c** or **-E**, the *c99* utility
8303 shall cause the equivalent of a **-l c** operand to be passed to the link editor as the last **-l** operand,
8304 causing it to be searched after all other object files and libraries are loaded.

8305 It is unspecified whether the libraries **libc.a**, **libm.a**, **librt.a**, **libpthread.a**, **libl.a**, **liby.a**, or **libxnet**
8306 exist as regular files. The implementation may accept as **-l** operands names of objects that do
8307 not exist as regular files.

8308 **External Symbols**

8309 The C compiler and link editor shall support the significance of external symbols up to a length
8310 of at least 31 bytes; the action taken upon encountering symbols exceeding the implementation-
8311 defined maximum symbol length is unspecified.

8312 The compiler and link editor shall support a minimum of 511 external symbols per source or
8313 object file, and a minimum of 4095 external symbols in total. A diagnostic message shall be
8314 written to the standard output if the implementation-defined limit is exceeded; other actions are
8315 unspecified.

8316 **Programming Environments**

8317 All implementations shall support one of the following programming environments as a default.
8318 Implementations may support more than one of the following programming environments.
8319 Applications can use *sysconf()* or *getconf* to determine which programming environments are
8320 supported.

8321 **Table 4-4** Programming Environments: Type Sizes

8322	Programming Environment	Bits in	Bits in	Bits in	Bits in
8323	<i>getconf</i> Name	int	long	pointer	off_t
8324	_POSIX_V6_ILP32_OFF32	32	32	32	32
8325	_POSIX_V6_ILP32_OFFBIG	32	32	32	≥64
8326	_POSIX_V6_LP64_OFF64	32	64	64	64
8327	_POSIX_V6_LP64_OFFBIG	≥32	≥64	≥64	≥64

8328 **Notes to Reviewers**

8329 *This section with side shading will not appear in the final copy. - Ed.*

8330 The names of the macros above may be changed. This has been added to the issues list.

8331 Implementations provide configuration strings for C compiler flags, linker/loader flags, and
8332 libraries for each supported environment. When an application needs to use a specific
8333 programming environment rather than the implementation default programming environment
8334 while compiling, the application shall first verify that the implementation supports the desired
8335 environment. If the desired programming environment is supported, the application shall then
8336 invoke *c99* with the appropriate C compiler flags as the first options for the compile, the
8337 appropriate linker/loader flags after any other options but before any operands, and the
8338 appropriate libraries at the end of the operands.

8339 Portable applications shall not attempt to link together object files compiled for different
8340 programming models. Applications shall also be aware that binary data placed in shared
8341 memory or in files might not be recognized by applications built for other programming models.

8342 **Table 4-5** Programming Environments: c99 and cc Arguments

8343	Programming Environment <i>getconf</i> Name	Use	<i>c99</i> and <i>cc</i> Arguments <i>getconf</i> Name
8345	_POSIX_V6_ILP32_OFF32	C Compiler Flags	POSIX_V6_ILP32_OFF32_CFLAGS
8346		Linker/Loader Flags	POSIX_V6_ILP32_OFF32_LDFLAGS
8347		Libraries	POSIX_V6_ILP32_OFF32_LIBS
8348	_POSIX_V6_ILP32_OFFBIG	C Compiler Flags	POSIX_V6_ILP32_OFFBIG_CFLAGS
8349		Linker/Loader Flags	POSIX_V6_ILP32_OFFBIG_LDFLAGS
8350		Libraries	POSIX_V6_ILP32_OFFBIG_LIBS
8351	_POSIX_V6_LP64_OFF64	C Compiler Flags	POSIX_V6_LP64_OFF64_CFLAGS
8352		Linker/Loader Flags	POSIX_V6_LP64_OFF64_LDFLAGS
8353		Libraries	POSIX_V6_LP64_OFF64_LIBS
8354	_POSIX_V6_LPBIG_OFFBIG	C Compiler Flags	POSIX_V6_LPBIG_OFFBIG_CFLAGS
8355		Linker/Loader Flags	POSIX_V6_LPBIG_OFFBIG_LDFLAGS
8356		Libraries	POSIX_V6_LPBIG_OFFBIG_LIBS

8357 **Notes to Reviewers**8358 *This section with side shading will not appear in the final copy. - Ed.*

8359 The names of the macros above may be changed. This has been added to the issues list.

8360 **EXIT STATUS**

8361 The following exit values shall be returned:

8362 0 Successful compilation or link edit.

8363 >0 An error occurred.

8364 **CONSEQUENCES OF ERRORS**

8365 When *c99* encounters a compilation error that causes an object file not to be created, it shall write
 8366 a diagnostic to standard error and continue to compile other source code operands, but it shall
 8367 not perform the link phase and return a non-zero exit status. If the link edit is unsuccessful, a
 8368 diagnostic message shall be written to standard error and *c99* exits with a non-zero status. A
 8369 portable application shall rely on the exit status of *c99*, rather than on the existence or mode of
 8370 the executable file.

8371 **APPLICATION USAGE**

8372 Since the *c99* utility usually creates files in the current directory during the compilation process,
 8373 it is typically necessary to run the *c99* utility in a directory in which a file can be created.

8374 On systems providing POSIX Conformance (see the Base Definitions volume of
 8375 IEEE Std. 1003.1-200x, Chapter 2, Conformance), *c99* is required only with the the C-Language
 8376 Development option; XSI-conformant systems always provide *c99*.

8377 Some historical implementations have created *.o* files when *-c* is not specified and more than
 8378 one source file is given. Since this area is left unspecified, the application cannot rely on *.o* files
 8379 being created, but it also must be prepared for any related *.o* files that already exist being deleted
 8380 at the completion of the link edit.

8381 Some historical implementations have permitted *-L* options to be interspersed with *-I* operands
 8382 on the command line. For an application to compile consistently on systems that do not behave
 8383 like this, it is necessary for a portable application to supply all *-L* options before any of the *-I*
 8384 options.

8385 There is the possible implication that if a user supplies versions of the standard library functions
 8386 (before they would be encountered by an implicit `-l c` or explicit `-l m`), that those versions
 8387 would be used in place of the standard versions. There are various reasons this might not be
 8388 true (functions defined as macros, manipulations for clean name space, and so on), so the
 8389 existence of files named in the same manner as the standard libraries within the `-L` directories is
 8390 explicitly stated to produce unspecified behavior.

8391 All of the functions specified in the System Interfaces volume of IEEE Std. 1003.1-200x may be
 8392 made visible by implementations when the Standard C Library is searched. Portable applications
 8393 must explicitly request searching the other standard libraries when functions made visible by
 8394 those libraries are used.

8395 EXAMPLES

8396 1. The following usage example compiles `foo.c` and creates the executable file `foo`:

```
8397 c99 -o foo foo.c
```

8398 The following usage example compiles `foo.c` and creates the object file `foo.o`:

```
8399 c99 -c foo.c
```

8400 The following usage example compiles `foo.c` and creates the executable file `a.out`:

```
8401 c99 foo.c
```

8402 The following usage example compiles `foo.c`, links it with `bar.o`, and creates the executable
 8403 file `a.out`. It also creates and leaves `foo.o`:

```
8404 c99 foo.c bar.o
```

8405 2. The following example shows how an application using threads interfaces can test for
 8406 support of and use a programming environment supporting 32-bit `int`, `long`, and `pointer`
 8407 types and an `off_t` type using at least 64 bits:

```
8408 if [ $(getconf _POSIX_V6_ILP32_OFFBIG) != "-1" ]
8409 then
8410     c99 $(getconf POSIX_V6_ILP32_OFFBIG_CFLAGS) -D_XOPEN_SOURCE=600 \
8411         $(getconf POSIX_V6_ILP32_OFFBIG_LDFLAGS) foo.c -o foo \
8412         $(getconf POSIX_V6_ILP32_OFFBIG_LIBS) -l pthread
8413 else
8414     echo ILP32_OFFBIG programming environment not supported
8415     exit 1
8416 fi
```

8417 **Notes to Reviewers**

8418 *This section with side shading will not appear in the final copy. - Ed.*

8419 The names of the macros above may be changed. This has been added to the issues list.

8420 3. The following examples clarify the use and interactions of `-L` options and `-l` operands.

8421 Consider the case in which module `a.c` calls function `f()` in library `libQ.a`, and module `b.c`
8422 calls function `g()` in library `libp.a`. Assume that both libraries reside in `/a/b/c`. The
8423 command line to compile and link in the desired way is:

```
8424 c99 -L /a/b/c main.o a.c -l Q b.c -l p
```

8425 In this case the `-l Q` operand need only precede the first `-l p` operand, since both `libQ.a`
8426 and `libp.a` reside in the same directory.

8427 Multiple `-L` operands can be used when library name collisions occur. Building on the
8428 previous example, suppose that the user wants to use a new `libp.a`, in `/a/a/a`, but still wants
8429 `f()` from `/a/b/c/libQ.a`:

```
8430 c99 -L /a/a/a -L /a/b/c main.o a.c -l Q b.c -l p
```

8431 In this example, the linker searches the `-L` options in the order specified, and finds
8432 `/a/a/a/libp.a` before `/a/b/c/libp.a` when resolving references for `b.c`. The order of the `-l`
8433 operands is still important, however.

8434 RATIONALE

8435 The `c99` utility is based on the `c89` utility originally introduced in the ISO POSIX-2: 1993 standard. |

8436 FUTURE DIRECTIONS

8437 None.

8438 SEE ALSO

8439 `ar`, `getconf`, `make`, `nm`, `strip`, `umask`, the System Interfaces volume of IEEE Std. 1003.1-200x,
8440 `sysconf()`

8441 CHANGE HISTORY

8442 First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard. |

8443 **NAME**

8444 cal — print a calendar

8445 **SYNOPSIS**8446 xSI cal [[*month*] *year*]

8447

8448 **DESCRIPTION**

8449 The *cal* utility shall write a calendar to standard output using the Julian calendar for dates from
 8450 January 1, 1 through September 2, 1752 and the Gregorian calendar for dates from September 14,
 8451 1752 through December 31, 9999 as though the Gregorian calendar had been adopted on
 8452 September 14, 1752.

8453 **OPTIONS**

8454 None.

8455 **OPERANDS**

8456 The following operands shall be supported:

8457 *month* Specify the month to be displayed, represented as a decimal integer from 1
 8458 (January) to 12 (December). The default shall be the current month.

8459 *year* Specify the year for which the calendar is displayed, represented as a decimal
 8460 integer from 1 to 9999. The default shall be the current year.

8461 **STDIN**

8462 Not used.

8463 **INPUT FILES**

8464 None.

8465 **ENVIRONMENT VARIABLES**8466 The following environment variables shall affect the execution of *cal*:

8467 *LANG* Provide a default value for the internationalization variables that are unset or null.
 8468 If *LANG* is unset or null, the corresponding value from the implementation-
 8469 defined default locale shall be used. If any of the internationalization variables
 8470 contains an invalid setting, the utility shall behave as if none of the variables had
 8471 been defined.

8472 *LC_ALL* If set to a non-empty string value, override the values of all the other
 8473 internationalization variables.

8474 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 8475 characters (for example, single-byte as opposed to multi-byte characters in
 8476 arguments).

8477 *LC_MESSAGES*

8478 Determine the locale that should be used to affect the format and contents of
 8479 diagnostic messages written to standard error, and informative messages written
 8480 to standard output.

8481 *LC_TIME* Determine the format and contents of the calendar.

8482 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

8483 *TZ* Determine the timezone used to calculate the value of the current month.

8484 **ASYNCHRONOUS EVENTS**

8485 Default.

8486 **STDOUT**

8487 The standard output shall be used to display the calendar, in an unspecified format.

8488 **STDERR**

8489 Used only for diagnostic messages.

8490 **OUTPUT FILES**

8491 None.

8492 **EXTENDED DESCRIPTION**

8493 None.

8494 **EXIT STATUS**

8495 The following exit values shall be returned:

8496 0 Successful completion.

8497 >0 An error occurred.

8498 **CONSEQUENCES OF ERRORS**

8499 Default.

8500 **APPLICATION USAGE**

8501 Note that:

8502 cal 83

8503 refers to A.D. 83, not 1983.

8504 **EXAMPLES**

8505 None.

8506 **RATIONALE**

8507 None.

8508 **FUTURE DIRECTIONS**8509 A future revision of IEEE Std. 1003.1-200x may support locale-specific recognition of the date of
8510 adoption of the Gregorian calendar.8511 **SEE ALSO**

8512 None.

8513 **CHANGE HISTORY**

8514 First released in Issue 2.

8515 **Issue 4**

8516 Format reorganized.

8517 Internationalized environment variable support mandated.

8518 **Issue 6**8519 The DESCRIPTION is updated to allow for traditional behavior for years before the adoption of
8520 the Gregorian calendar.

8521 **NAME**8522 `cat` — concatenate and print files8523 **SYNOPSIS**8524 `cat [-u][file ...]`8525 **DESCRIPTION**8526 The `cat` utility reads files in sequence and writes their contents to the standard output in the
8527 same sequence.8528 **OPTIONS**8529 The `cat` utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
8530 12.2, Utility Syntax Guidelines.

8531 The following option shall be supported:

8532 **-u** Write bytes from the input file to the standard output without delay as each is
8533 read.8534 **OPERANDS**

8535 The following operand shall be supported:

8536 **file** A path name of an input file. If no *file* operands are specified, the standard input is
8537 used. If a *file* is '-', the `cat` utility shall read from the standard input at that point
8538 in the sequence. The `cat` utility shall not close and reopen standard input when it is
8539 referenced in this way, but shall accept multiple occurrences of '-' as a *file*
8540 operand.8541 **STDIN**8542 The standard input is used only if no *file* operands are specified, or if a *file* operand is '-'. See
8543 the INPUT FILES section.8544 **INPUT FILES**

8545 The input files can be any file type.

8546 **ENVIRONMENT VARIABLES**8547 The following environment variables shall affect the execution of `cat`:8548 **LANG** Provide a default value for the internationalization variables that are unset or null.
8549 If *LANG* is unset or null, the corresponding value from the implementation-
8550 defined default locale shall be used. If any of the internationalization variables
8551 contains an invalid setting, the utility shall behave as if none of the variables had
8552 been defined.8553 **LC_ALL** If set to a non-empty string value, override the values of all the other
8554 internationalization variables.8555 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
8556 characters (for example, single-byte as opposed to multi-byte characters in
8557 arguments).8558 **LC_MESSAGES**8559 Determine the locale that should be used to affect the format and contents of
8560 diagnostic messages written to standard error.8561 **XSI** **NLS_PATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

8562 **ASYNCHRONOUS EVENTS**

8563 Default.

8564 **STDOUT**

8565 The standard output shall contain the sequence of bytes read from the input files. Nothing else
8566 shall be written to the standard output.

8567 **STDERR**

8568 Used only for diagnostic messages.

8569 **OUTPUT FILES**

8570 None.

8571 **EXTENDED DESCRIPTION**

8572 None.

8573 **EXIT STATUS**

8574 The following exit values shall be returned:

8575 0 All input files were output successfully.

8576 >0 An error occurred.

8577 **CONSEQUENCES OF ERRORS**

8578 Default.

8579 **APPLICATION USAGE**

8580 The **-u** option has value in prototyping non-blocking reads from FIFOs. The intent is to support
8581 the following sequence:

8582 `mkfifo foo`8583 `cat -u foo > /dev/tty13 &`8584 `cat -u > foo`

8585 It is unspecified whether standard output is or is not buffered in the default case. This is
8586 sometimes of interest when standard output is associated with a terminal, since buffering may
8587 delay the output. The presence of the **-u** option guarantees that unbuffered I/O is available. It is
8588 implementation-defined whether the *cat* utility buffers output if the **-u** option is not specified.
8589 Traditionally, the **-u** option is implemented using the equivalent of the *setvbuf()* function
8590 defined in the System Interfaces volume of IEEE Std. 1003.1-200x.

8591 **EXAMPLES**

8592 The following command:

8593 `cat myfile`8594 writes the contents of the file **myfile** to standard output.

8595 The following command:

8596 `cat doc1 doc2 > doc.all`8597 concatenates the files **doc1** and **doc2** and writes the result to **doc.all**.

8598 Because of the shell language mechanism used to perform output redirection, a command such
8599 as this:

8600 `cat doc doc.end > doc`8601 causes the original data in **doc** to be lost.

8602 The command:

8603 `cat start - middle - end > file`

8604 when standard input is a terminal, gets two arbitrary pieces of input from the terminal with a
8605 single invocation of *cat*. Note, however, that if standard input is a regular file, this would be
8606 equivalent to the command:

8607 `cat start - middle /dev/null end > file`

8608 because the entire contents of the file would be consumed by *cat* the first time '-' was used as a
8609 *file* operand and an end-of-file condition would be detected immediately when '-' was
8610 referenced the second time.

8611 RATIONALE

8612 Historical versions of the *cat* utility include the options `-e`, `-t`, and `-v`, which permit the ends of
8613 lines, `<tab>`s, and invisible characters, respectively, to be rendered visible in the output. The
8614 standard developers omitted these options because they provide too fine a degree of control
8615 over what is made visible, and similar output can be obtained using a command such as:

8616 `sed -n -e 's/$/$/' -e l pathname`

8617 The `-s` option was omitted because it corresponds to different functions in BSD and System V-
8618 based systems. The BSD `-s` option to squeeze blank lines can be accomplished by the shell script
8619 shown in following example:

```
8620 sed -n '  
8621 # Write non-empty lines.  
8622 ./ {  
8623     p  
8624     d  
8625 }  
8626 # Write a single empty line, then look for more empty lines.  
8627 /^$/ p  
8628 # Get next line, discard the held <newline> (empty line),  
8629 # and look for more empty lines.  
8630 :Empty  
8631 /^$/ {  
8632     N  
8633     s/./.  
8634     b Empty  
8635 }  
8636 # Write the non-empty line before going back to search  
8637 # for the first in a set of empty lines.  
8638     p  
8639 '
```

8640 The System V `-s` option to silence error messages can be accomplished by redirecting the
8641 standard error. Note that the BSD documentation for *cat* uses the term "blank line" to mean the
8642 same as the POSIX "empty line": a line consisting only of a `<newline>`.

8643 The BSD `-n` option was omitted because similar functionality can be obtained from the `-n`
8644 option of the *pr* utility.

8645 FUTURE DIRECTIONS

8646 None.

8647 **SEE ALSO**

8648 *more*

8649 **CHANGE HISTORY**

8650 First released in Issue 2.

8651 **Issue 4**

8652 Aligned with the ISO/IEC 9945-2: 1993 standard.

8653 NAME

8654 cd — change the working directory

8655 SYNOPSIS

8656 cd [-L] [-P] [*directory*]

8657 cd -

8658 DESCRIPTION

8659 The *cd* utility shall change the working directory of the current shell execution environment (see
8660 Section 2.13 (on page 2273)) by executing the following steps in sequence. (In the following
8661 steps, the symbol **curpath** represents an intermediate value used to simplify the description of
8662 the algorithm used by *cd*. There is no requirement that **curpath** be made visible to the
8663 application.)

- 8664 1. If no *directory* operand is given and the *HOME* environment variable is empty or
8665 undefined, the default behavior is implementation-defined and no further steps shall be
8666 taken.
- 8667 2. If no *directory* operand is given and the *HOME* environment variable is set to a non-empty
8668 value, the *cd* utility shall behave as if the directory named in the *HOME* environment
8669 variable was specified as the *directory* operand.
- 8670 3. If the *directory* operand begins with a slash character, set **curpath** to the operand and
8671 proceed to step 7.
- 8672 4. If the first component of the *directory* operand is dot or dot-dot, proceed to step 6.
- 8673 5. Starting with the first path name in the colon-separated path names of *CDPATH* (see the
8674 ENVIRONMENT VARIABLES section) if the path name is non-null, test if the
8675 concatenation of that path name, a slash character, and the *directory* operand names a
8676 directory. If the path name is null, test if the concatenation of dot, a slash character, and the
8677 operand names a directory. In either case, if the resulting string names an existing
8678 directory, set **curpath** to that string and proceed to step 7. Otherwise, repeat this step with
8679 the next path name in *CDPATH* until all path names have been tested.
- 8680 6. Set **curpath** to the string formed by the concatenation of the value of *PWD* a slash
8681 character, and the operand.
- 8682 7. If the **-P** option is in effect, the *cd* utility shall perform actions equivalent to the *chdir*()
8683 function, called with **curpath** as the *path* argument. If these actions succeed, the *PWD*
8684 environment variable shall be set to an absolute path name for the current working
8685 directory and shall not contain file name components that, in the context of path name
8686 resolution, refer to a file of type symbolic link. If there is insufficient permission on the new
8687 directory, or on any parent of that directory, to determine the current working directory,
8688 the value of the *PWD* environment variable is unspecified. If the actions equivalent to
8689 *chdir*() fail for any reason, the *cd* utility shall display an appropriate error message and not
8690 alter the *PWD* environment variable. Whether the actions equivalent to *chdir*() succeed or
8691 fail, no further steps shall be taken.
- 8692 8. The **curpath** value shall then be converted to canonical form as follows, considering each
8693 component from beginning to end, in sequence:
 - 8694 a. Dot components and any slashes that separate them from the next component shall
8695 be deleted.
 - 8696 b. For each dot-dot component, if there is a preceding component and it is neither root
8697 nor dot-dot, the preceding component, all slashes separating the preceding
8698 component from dot-dot, dot-dot, and all slashes separating dot-dot from the

- 8699 following component shall be deleted.
- 8700 c. An implementation may further simplify **curpath** by removing any trailing slash
8701 characters that are not also leading slashes, replacing multiple non-leading
8702 consecutive slashes with a single slash, and replacing three or or more leading
8703 slashes with a single slash. If, as a result of this canonicalization, the **curpath** variable
8704 is null, no further steps shall be taken.
- 8705 9. The *cd* utility shall then perform actions equivalent to the *chdir()* function called with
8706 **curpath** as the *path* argument. If these actions failed for any reason, the *cd* utility shall
8707 display an appropriate error message and no further steps shall be taken. The *PWD*
8708 environment variable shall be set to **curpath**.

8709 If, during the execution of the above steps, the *PWD* environment variable is changed, the
8710 *OLDPWD* environment variable shall also be changed to the value of the old working directory
8711 (that is the current working directory immediately prior to the call to *cd*).

8712 OPTIONS

8713 The *cd* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
8714 Utility Syntax Guidelines.

8715 The following options shall be supported by the implementation:

- 8716 **-L** Handle the operand dot-dot logically; symbolic link components shall not be
8717 resolved before dot-dot components are processed (see steps 5. and 6. in the
8718 DESCRIPTION).
- 8719 **-P** Handle the operand dot-dot physically; symbolic link components shall be
8720 resolved before dot-dot components are processed (see step 4. in the
8721 DESCRIPTION).

8722 If both **-L** and **-P** options are specified, the last of these options shall be used and all others
8723 ignored. If neither **-L** nor **-P** is specified, the operand shall be handled dot-dot logically; see the
8724 DESCRIPTION.

8725 OPERANDS

8726 The following operands shall be supported:

8727 *directory* An absolute or relative path name of the directory that shall become the new
8728 working directory. The interpretation of a relative path name by *cd* depends on the
8729 **-L** option and the *CDPATH* and *PWD* environment variables. If *directory* is an
8730 empty string, the results are unspecified.

8731 **-** When a hyphen is used as the operand, this is equivalent to the command:

```
8732 cd "$OLDPWD" && pwd
```

8733 which changes to the previous working directory and then writes its name.

8734 STDIN

8735 Not used.

8736 INPUT FILES

8737 None.

8738 ENVIRONMENT VARIABLES

8739 The following environment variables shall affect the execution of *cd*:

8740 *CDPATH* A colon-separated list of path names that refer to directories. The *cd* utility shall
8741 use this list in its attempt to change the directory, as described in the
8742 DESCRIPTION. An empty string in place of a directory path name represents the

- 8743 current directory. If *CDPATH* is not set, it shall be treated as if it were an empty
8744 string.
- 8745 *HOME* The name of the directory, used when no *directory* operand is specified.
- 8746 *LANG* Provide a default value for the internationalization variables that are unset or null.
8747 If *LANG* is unset or null, the corresponding value from the implementation-
8748 defined default locale shall be used. If any of the internationalization variables
8749 contains an invalid setting, the utility shall behave as if none of the variables had
8750 been defined.
- 8751 *LC_ALL* If set to a non-empty string value, override the values of all the other
8752 internationalization variables.
- 8753 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
8754 characters (for example, single-byte as opposed to multi-byte characters in
8755 arguments).
- 8756 *LC_MESSAGES*
8757 Determine the locale that should be used to affect the format and contents of
8758 diagnostic messages written to standard error.
- 8759 xSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 8760 *OLDPWD* A path name of the previous working directory, used by *cd -*.
- 8761 *PWD* This variable shall be set as specified in the DESCRIPTION. If an application sets
8762 or unsets the value of *PWD*, the behavior of *cd* is unspecified.
- 8763 **ASYNCHRONOUS EVENTS**
- 8764 Default.
- 8765 **STDOUT**
- 8766 If a non-empty directory name from *CDPATH* is used, or if *cd -* is used, an absolute path name of
8767 the new working directory shall be written to the standard output as follows:
- 8768 "%s\n", <*new directory*>
- 8769 Otherwise, there shall be no output.
- 8770 **STDERR**
- 8771 Used only for diagnostic messages.
- 8772 **OUTPUT FILES**
- 8773 None.
- 8774 **EXTENDED DESCRIPTION**
- 8775 None.
- 8776 **EXIT STATUS**
- 8777 The following exit values shall be returned:
- 8778 0 The directory was successfully changed.
- 8779 >0 An error occurred.
- 8780 **CONSEQUENCES OF ERRORS**
- 8781 The working directory shall remain unchanged.

8782 APPLICATION USAGE

8783 Since *cd* affects the current shell execution environment, it is always provided as a shell regular
8784 built-in. If it is called in a subshell or separate utility execution environment, such as one of the
8785 following:

```
8786 (cd /tmp)
8787 nohup cd
8788 find . -exec cd {} \;
```

8789 it does not affect the working directory of the caller's environment.

8790 The user must have execute (search) permission in *directory* in order to change to it.

8791 EXAMPLES

8792 None.

8793 RATIONALE

8794 The use of the *CDPATH* was introduced in the System V shell. Its use is analogous to the use of
8795 the *PATH* variable in the shell. The BSD C shell used a shell parameter *cdpath* for this purpose.

8796 A common extension when *HOME* is undefined is to get the login directory from the user
8797 database for the invoking user. This does not occur on System V implementations.

8798 Some historical shells, such as the KornShell, took special actions when the directory name
8799 contained a dot-dot component, selecting the logical parent of the directory, rather than the
8800 actual parent directory; that is, it moved up one level toward the '/' in the path name,
8801 remembering what the user typed, rather than performing the equivalent of:

```
8802 chdir("../");
```

8803 In such a shell, the following commands would not necessarily produce equivalent output for all
8804 directories:

```
8805 cd .. && ls      ls ..
```

8806 This behavior is not permitted by default because it is not consistent with the definition of dot-
8807 dot in most historical practice; that is, while this behavior has been optionally available in the
8808 KornShell, other shells have historically not supported this functionality. The logical path name
8809 is stored in the *PWD* environment variable when the *cd* utility completes and this value is used
8810 to construct the next directory name if *cd* is invoked with the *-L* option.

8811 FUTURE DIRECTIONS

8812 None.

8813 SEE ALSO

8814 *pwd*, the System Interfaces volume of IEEE Std. 1003.1-200x, *chdir()*

8815 CHANGE HISTORY

8816 First released in Issue 2.

8817 Issue 4

8818 Aligned with the ISO/IEC 9945-2:1993 standard.

8819 Extensions added for *cd-*, *PWD*, and *OLDPWD*.

8820 Issue 6

8821 The following new requirements on POSIX implementations derive from alignment with the
8822 Single UNIX Specification:

- 8823 • The *cd-*, *PWD*, and *OLDPWD* are added.

8824
8825

The **-L** and **-P** options are added to align with the IEEE P1003.2b draft standard. This also includes the introduction of a new description to include the effect of these options.

8826 NAME

8827 cflow — generate a C-language flowgraph (DEVELOPMENT)

8828 SYNOPSIS

```
8829 xSI cflow [-r][-d num][-D name[=def]] ... [-i incl][-I dir] ...
8830 [-U dir] ... file ...
```

8831

8832 DESCRIPTION

8833 The *cflow* utility shall analyse a collection of object files or assembler, C-language, *lex* or *yacc*
 8834 source files, and attempt to build a graph, written to standard output, charting the external
 8835 references.

8836 OPTIONS

8837 The *cflow* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 8838 12.2, Utility Syntax Guidelines, except that the order of the **-D**, **-I**, and **-U** options (which are
 8839 identical to their interpretation by *c99*) is significant.

8840 The following options shall be supported:

8841 **-d num** Indicate the depth at which the flowgraph is cut off. The application shall ensure
 8842 that the argument *num* is a decimal integer. By default this is a very large number
 8843 (typically greater than 32 000). Attempts to set the cut-off depth to a non-positive
 8844 integer are ignored.

8845 **-i incl** Increase the number of included symbols. The *incl* option-argument is one of the
 8846 following characters:

8847 *x* Include external and static data symbols. The default shall be to include only
 8848 functions in the flowgraph.

8849 *_* (Underscore) Include names that begin with an underscore. The default shall
 8850 be to exclude these functions (and data if **-i x** is used).

8851 **-r** Reverse the caller:callee relationship, producing an inverted listing showing the
 8852 callers of each function. The listing is also sorted in lexicographical order by callee.

8853 OPERANDS

8854 The following operand is supported:

8855 *file* The path name of a file for which a graph is to be generated. Files suffixed in **.l**, **.y**,
 8856 **.c**, and **.i** shall be processed by *lex* and *yacc* and preprocessed by the *c99*
 8857 preprocessor phase (bypassed for **.i** files) as appropriate, and then run through the
 8858 first pass of *lint*. Files suffixed with **.s** shall be assembled and information shall be
 8859 extracted (as in **.o** files) from the symbol table.

8860 STDIN

8861 Not used.

8862 INPUT FILES

8863 The input files shall be object files or assembler, C-language, *lex* or *yacc* source files.

8864 ENVIRONMENT VARIABLES

8865 The following environment variables shall affect the execution of *cflow*:

8866 *LANG* Provide a default value for the internationalization variables that are unset or null.
 8867 If *LANG* is unset or null, the corresponding value from the implementation-
 8868 defined default locale shall be used. If any of the internationalization variables
 8869 contains an invalid setting, the utility shall behave as if none of the variables had
 8870 been defined.

- 8871 *LC_ALL* If set to a non-empty string value, override the values of all the other
8872 internationalization variables.
- 8873 *LC_COLLATE*
8874 Determine the locale for the ordering of the output when the *-r* option is used.
- 8875 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
8876 characters (for example, single-byte as opposed to multi-byte characters in
8877 arguments and input files).
- 8878 *LC_MESSAGES*
8879 Determine the locale that should be used to affect the format and contents of
8880 diagnostic messages written to standard error.
- 8881 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 8882 **ASYNCHRONOUS EVENTS**
8883 Default.
- 8884 **STDOUT**
8885 The flowgraph written to standard output shall be formatted as follows:
8886 "%d %s:%s\n", <reference number>, <global>, <definition>
- 8887 Each line of output begins with a reference (that is, line) number, followed by indentation of at
8888 least one column position per level. This is followed by the name of the global, a colon, and its
8889 definition. Normally globals are only functions not defined as an external or beginning with an
8890 underscore; see the *OPTIONS* section for the *-i* inclusion option. For information extracted from
8891 C-language source, the definition consists of an abstract type declaration (for example, **char***)
8892 and, delimited by angle brackets, the name of the source file and the line number where the
8893 definition was found. Definitions extracted from object files indicate the file name and location
8894 counter under which the symbol appeared (for example, *text*).
- 8895 Once a definition of a name has been written, subsequent references to that name contain only
8896 the reference number of the line where the definition can be found. For undefined references,
8897 only "< >" shall be written.
- 8898 **STDERR**
8899 Used only for diagnostic messages.
- 8900 **OUTPUT FILES**
8901 None.
- 8902 **EXTENDED DESCRIPTION**
8903 None.
- 8904 **EXIT STATUS**
8905 The following exit values shall be returned:
8906 0 Successful completion.
8907 >0 An error occurred.
- 8908 **CONSEQUENCES OF ERRORS**
8909 Default.

8910 APPLICATION USAGE

8911 Files produced by *lex* and *yacc* cause the reordering of line number declarations, and this can
8912 confuse *cflow*. To obtain proper results, the input of *yacc* or *lex* must be directed to *cflow*.

8913 EXAMPLES

8914 Given the following in **file.c**:

```
8915     int i;  
8916     main()  
8917     {  
8918         f();  
8919         g();  
8920         f();  
8921     }  
8922     f()  
8923     {  
8924         i = h();  
8925     }
```

8926 The command:

```
8927 cflow -i x file.c
```

8928 produces the output:

```
8929 1 main: int(), <file.c 2>  
8930 2   f: int(), <file.c 8>  
8931 3     h: <>  
8932 4     i: int, <file.c 1>  
8933 5   g: <>
```

8934 RATIONALE

8935 None.

8936 FUTURE DIRECTIONS

8937 None.

8938 SEE ALSO

8939 *c99*, *lex*, *yacc*

8940 CHANGE HISTORY

8941 First released in Issue 2.

8942 Issue 4

8943 Format reorganized.

8944 Internationalized environment variable support mandated.

8945 Issue 6

8946 The normative text is reworded to avoid use of the term “must” for application requirements.

8947 **NAME**

8948 chgrp — change the file group ownership

8949 **SYNOPSIS**8950 chgrp -hR *group file ...*8951 chgrp -R [-H | -L | -P] *group file ...*8952 **DESCRIPTION**8953 The *chgrp* utility shall set the group ID of the file named by each *file* operand to the group ID
8954 specified by the *group* operand.8955 For each *file* operand, it shall perform actions equivalent to the *chown()* function defined in the
8956 System Interfaces volume of IEEE Std. 1003.1-200x, called with the following arguments:

- 8957 • The *file* operand shall be used as the *path* argument.
- 8958 • The user ID of the file shall be used as the *owner* argument.
- 8959 • The specified group ID shall be used as the *group* argument.

8960 Unless *chgrp* is invoked by a process with appropriate privileges, the set-user-ID and set-group-
8961 ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-
8962 group-ID bits of other file types may be cleared.8963 **OPTIONS**8964 The *chgrp* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
8965 12.2, Utility Syntax Guidelines.

8966 The following options shall be supported by the implementation:

8967 **-h** If the system supports group IDs for symbolic links, for each *file* operand that
8968 names a file of type symbolic link, *chgrp* shall attempt to set the group ID of the
8969 symbolic link instead of the file referenced by the symbolic link. If the system does
8970 not support group IDs for symbolic links, for each *file* operand that names a file of
8971 type symbolic link, *chgrp* shall do nothing more with the current file and shall go
8972 on to any remaining files.

8973 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory
8974 is specified on the command line, *chgrp* shall change the group of the directory
8975 referenced by the symbolic link and all files in the file hierarchy below it.

8976 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory
8977 is specified on the command line or encountered during the traversal of a file
8978 hierarchy, *chgrp* shall change the group of the directory referenced by the symbolic
8979 link and all files in the file hierarchy below it.

8980 **-P** If the **-R** option is specified and a symbolic link is specified on the command line
8981 or encountered during the traversal of a file hierarchy, *chgrp* shall change the
8982 group ID of the symbolic link if the system supports this operation. The *chgrp*
8983 utility shall not follow the symbolic link to any other part of the file hierarchy.

8984 **-R** Recursively change file group IDs. For each *file* operand that names a directory,
8985 *chgrp* shall change the group of the directory and all files in the file hierarchy below
8986 it. Unless a **-H**, **-L**, or **-P** option is specified, it is unspecified which of these
8987 options will be used as the default.

8988 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be
8989 considered an error. The last option specified shall determine the behavior of the utility.

8990 **OPERANDS**

8991 The following operands shall be supported:

8992 *group* A group name from the group database or a numeric group ID. Either specifies a
8993 group ID to be given to each file named by one of the *file* operands. If a numeric
8994 *group* operand exists in the group database as a group name, the group ID number
8995 associated with that group name is used as the group ID.

8996 *file* A path name of a file whose group ID is to be modified.

8997 **STDIN**

8998 Not used.

8999 **INPUT FILES**

9000 None.

9001 **ENVIRONMENT VARIABLES**9002 The following environment variables shall affect the execution of *chgrp*:

9003 *LANG* Provide a default value for the internationalization variables that are unset or null.
9004 If *LANG* is unset or null, the corresponding value from the implementation-
9005 defined default locale shall be used. If any of the internationalization variables
9006 contains an invalid setting, the utility shall behave as if none of the variables had
9007 been defined.

9008 *LC_ALL* If set to a non-empty string value, override the values of all the other
9009 internationalization variables.

9010 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
9011 characters (for example, single-byte as opposed to multi-byte characters in
9012 arguments).

9013 *LC_MESSAGES*

9014 Determine the locale that should be used to affect the format and contents of
9015 diagnostic messages written to standard error.

9016 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

9017 **ASYNCHRONOUS EVENTS**

9018 Default.

9019 **STDOUT**

9020 Not used.

9021 **STDERR**

9022 Used only for diagnostic messages.

9023 **OUTPUT FILES**

9024 None.

9025 **EXTENDED DESCRIPTION**

9026 None.

9027 **EXIT STATUS**

9028 The following exit values shall be returned:

9029 0 The utility executed successfully and all requested changes were made.

9030 >0 An error occurred.

9031 **CONSEQUENCES OF ERRORS**

9032 Default.

9033 **APPLICATION USAGE**9034 Only the owner of a file or the user with appropriate privileges may change the owner or group
9035 of a file.9036 Some systems restrict the use of *chgrp* to a user with appropriate privileges when the *group*
9037 specified is not the effective group ID or one of the supplementary group IDs of the calling
9038 process.9039 **EXAMPLES**

9040 None.

9041 **RATIONALE**9042 The System V and BSD versions use different exit status codes. Some implementations used the
9043 exit status as a count of the number of errors that occurred; this practice is unworkable since it
9044 can overflow the range of valid exit status values. The standard developers chose to mask these
9045 by specifying only 0 and >0 as exit values.9046 The functionality of *chgrp* is described substantially through references to *chown()*. In this way,
9047 there is no duplication of effort required for describing the interactions of permissions, multiple
9048 groups, and so on.9049 **FUTURE DIRECTIONS**

9050 None.

9051 **SEE ALSO**9052 *chmod*, *chown*, the System Interfaces volume of IEEE Std. 1003.1-200x, *chown()*9053 **CHANGE HISTORY**

9054 First released in Issue 2.

9055 **Issue 4**

9056 Aligned with the ISO/IEC 9945-2:1993 standard.

9057 **Issue 6**9058 New options **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These
9059 options affect the processing of symbolic links.9060 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS
9061 section to "Default."

9062 **NAME**

9063 chmod — change the file modes

9064 **SYNOPSIS**9065 chmod [-R] *mode file ...*9066 **DESCRIPTION**9067 The *chmod* utility shall change any or all of the file mode bits of the file named by each *file*
9068 operand in the way specified by the *mode* operand.9069 It is implementation-defined whether and how the *chmod* utility affects any alternate or
9070 additional file access control mechanism (see the Base Definitions volume of
9071 IEEE Std. 1003.1-200x, Section 4.1, File Access Permissions) being used for the specified file.9072 Only a process whose effective user ID matches the user ID of the file, or a process with the
9073 appropriate privileges, shall be permitted to change the file mode bits of a file.9074 **OPTIONS**9075 The *chmod* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
9076 12.2, Utility Syntax Guidelines.

9077 The following option shall be supported:

9078 **-R** Recursively change file mode bits. For each *file* operand that names a directory,
9079 *chmod* shall change the file mode bits of the directory and all files in the file
9080 hierarchy below it.9081 **OPERANDS**

9082 The following operands shall be supported:

9083 *mode* Represents the change to be made to the file mode bits of each file named by one of
9084 the *file* operands; see the EXTENDED DESCRIPTION section.9085 *file* A path name of a file whose file mode bits shall be modified.9086 **STDIN**

9087 Not used.

9088 **INPUT FILES**

9089 None.

9090 **ENVIRONMENT VARIABLES**9091 The following environment variables shall affect the execution of *chmod*:9092 *LANG* Provide a default value for the internationalization variables that are unset or null.
9093 If *LANG* is unset or null, the corresponding value from the implementation-
9094 defined default locale shall be used. If any of the internationalization variables
9095 contains an invalid setting, the utility shall behave as if none of the variables had
9096 been defined.9097 *LC_ALL* If set to a non-empty string value, override the values of all the other
9098 internationalization variables.9099 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
9100 characters (for example, single-byte as opposed to multi-byte characters in
9101 arguments).9102 *LC_MESSAGES*9103 Determine the locale that should be used to affect the format and contents of
9104 diagnostic messages written to standard error.

9105 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

9106 **ASYNCHRONOUS EVENTS**

9107 Default.

9108 **STDOUT**

9109 Not used.

9110 **STDERR**

9111 Used only for diagnostic messages.

9112 **OUTPUT FILES**

9113 None.

9114 **EXTENDED DESCRIPTION**

9115 The *mode* operand shall be either a *symbolic_mode* expression or a non-negative octal integer. The
 9116 *symbolic_mode* form is described by the grammar later in this section.

9117 Each **clause** shall specify an operation to be performed on the current file mode bits of each *file*.
 9118 The operations shall be performed on each *file* in the order in which the **clauses** are specified.

9119 The **who** symbols **u**, **g**, and **o** shall specify the *user*, *group*, and *other* parts of the file mode bits,
 9120 respectively. A **who** consisting of the symbol **a** shall be equivalent to **ugo**.

9121 The **perm** symbols **r**, **w**, and **x** represent the *read*, *write*, and *execute/search* portions of file mode
 9122 bits, respectively. The **perm** symbol **s** shall represent the *set-user-ID-on-execution* (when **who**
 9123 contains or implies **u**) and *set-group-ID-on-execution* (when **who** contains or implies **g**) bits.

9124 The **perm** symbol **X** shall represent the execute/search portion of the file mode bits if the file is a
 9125 directory or if the current (unmodified) file mode bits have at least one of the execute bits
 9126 (S_IXUSR, S_IXGRP, or S_IXOTH) set. It shall be ignored if the file is not a directory and none of
 9127 the execute bits are set in the current file mode bits.

9128 The **permcop** symbols **u**, **g**, and **o** shall represent the current permissions associated with the
 9129 user, group, and other parts of the file mode bits, respectively. For the remainder of this section,
 9130 **perm** refers to the non-terminals **perm** and **permcop** in the grammar.

9131 If multiple **actionlists** are grouped with a single **wholist** in the grammar, each **actionlist** shall be
 9132 applied in the order specified with that **wholist**. The *op* symbols shall represent the operation
 9133 performed, as follows:

9134 + If **perm** is not specified, the '+' operation shall not change the file mode bits.

9135 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
 9136 other permissions, except for those with corresponding bits in the file mode creation mask
 9137 of the invoking process, shall be set.

9138 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

9139 - If **perm** is not specified, the '-' operation shall not change the file mode bits.

9140 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
 9141 other permissions, except for those with corresponding bits in the file mode creation mask
 9142 of the invoking process, shall be cleared.

9143 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be
 9144 cleared.

9145 = Clear the file mode bits specified by the **who** value, or, if no **who** value is specified, all of the
 9146 file mode bits specified in this volume of IEEE Std. 1003.1-200x.

9147 If **perm** is not specified, the '=' operation shall make no further modifications to the file
9148 mode bits.

9149 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
9150 other permissions, except for those with corresponding bits in the file mode creation mask
9151 of the invoking process, shall be set.

9152 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

9153 When using the symbolic mode form on a regular file, it is implementation-defined whether or
9154 not:

- 9155 • Requests to set the set-user-ID-on-execution or set-group-ID-on-execution bit when all
9156 execute bits are currently clear and none are being set are ignored.
- 9157 • Requests to clear all execute bits also clear the set-user-ID-on-execution and set-group-ID-
9158 on-execution bits.
- 9159 • Requests to clear the set-user-ID-on-execution or set-group-ID-on-execution bits when all
9160 execute bits are currently clear are ignored. However, if the command *ls -l file* writes an *s* in
9161 the position indicating that the set-user-ID-on-execution or set-group-ID-on-execution is set,
9162 the commands *chmod u-s file* or *chmod g-s file*, respectively, shall not be ignored.

9163 When using the symbolic mode form on other file types, it is implementation-defined whether
9164 or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are
9165 honored.

9166 If the **who** symbol **o** is used in conjunction with the **perm** symbol **s** with no other **who** symbols
9167 being specified, the set-user-ID-on-execution and set-group-ID-on-execution bits shall not be
9168 modified. It shall not be an error to specify the **who** symbol **o** in conjunction with the **perm**
9169 symbol **s**.

9170 For an octal integer *mode* operand, the file mode bits shall be set absolutely.

9171 For each bit set in the octal number, the corresponding file permission bit shown in the following
9172 table shall be set; all other file permission bits shall be cleared. For regular files, for each bit set in
9173 the octal number corresponding to the set-user-ID-on-execution or the set-group-ID-on-
9174 execution, bits shown in the following table shall be set; if these bits are not set in the octal
9175 number, they are cleared. For other file types, it is implementation-defined whether or not
9176 requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are
9177 honored.

9178	Octal	Mode Bit	Octal	Mode Bit	Octal	Mode Bit	Octal	Mode Bit
9179	4000	S_ISUID	0400	S_IRUSR	0040	S_IRGRP	0004	S_IROTH
9180	2000	S_ISGID	0200	S_IWUSR	0020	S_IWGRP	0002	S_IWOTH
9181			0100	S_IXUSR	0010	S_IXGRP	0001	S_IXOTH

9182 When bits are set in the octal number other than those listed in the table above, the behavior is
9183 unspecified.

9184 **Grammar for chmod**

9185 The grammar and lexical conventions in this section describe the syntax for the *symbolic_mode*
 9186 operand. The general conventions for this style of grammar are described in Section 1.10 (on
 9187 page 2223). A valid *symbolic_mode* can be represented as the non-terminal symbol *symbolic_mode*
 9188 in the grammar. This formal syntax shall take precedence over the preceding text syntax
 9189 description.

9190 The lexical processing is based entirely on single characters. Implementations need not allow
 9191 blank characters within the single argument being processed.

```

9192 %start    symbolic_mode
9193 %%
9194 symbolic_mode    : section
9195                  | symbolic_mode ',' clause
9196                  ;
9197 clause           : actionlist
9198                  | wholist actionlist
9199                  ;
9200 wholist          : who
9201                  | wholist who
9202                  ;
9203 who              : 'u' | 'g' | 'o' | 'a'
9204                  ;
9205 actionlist       : action
9206                  | actionlist action
9207                  ;
9208 action           : op
9209                  | op permlist
9210                  | op permcopy
9211                  ;
9212 permcopy         : 'u' | 'g' | 'o'
9213                  ;
9214 op               : '+' | '-' | '='
9215                  ;
9216 permlist         : perm
9217                  | perm permlist
9218                  ;
9219 perm             : 'r' | 'w' | 'x' | 'X' | 's'
9220                  ;

```

9221 **EXIT STATUS**

9222 The following exit values shall be returned:

9223 0 The utility executed successfully and all requested changes were made.

9224 >0 An error occurred.

9225 CONSEQUENCES OF ERRORS

9226 Default.

9227 APPLICATION USAGE

9228 Some implementations of the *chmod* utility change the mode of a directory before the files in the
 9229 directory when performing a recursive (**-R** option) change; others change the directory mode
 9230 after the files in the directory. If an application tries to remove read or search permission for a
 9231 file hierarchy, the removal attempt fails if the directory is changed first; on the other hand, trying
 9232 to re-enable permissions to a restricted hierarchy fails if directories are changed last. Users
 9233 should not try to make a hierarchy inaccessible to themselves.

9234 Some implementations of *chmod* never used the process' *umask* when changing modes; systems
 9235 conformant with this volume of IEEE Std. 1003.1-200x do so when **who** is not specified. Note the
 9236 difference between:

9237 `chmod a-w file`

9238 which removes all write permissions, and:

9239 `chmod -- -w file`

9240 which removes write permissions that would be allowed if **file** was created with the same
 9241 *umask*.

9242 Portable applications should never assume that they know how the set-user-ID and set-group-
 9243 ID bits on directories are interpreted.

9244 EXAMPLES

9245

9246

9247

9248

9249

9250

9251

9252

9253

9254

Mode	Results
<i>a+=</i>	Equivalent to <i>a+,a=</i> ; clears all file mode bits.
<i>go+-w</i>	Equivalent to <i>go+,go-w</i> ; clears group and other write bits.
<i>g=0-w</i>	Equivalent to <i>g=0,g-w</i> ; sets group bit to match other bits and then clears group write bit.
<i>g-r+w</i>	Equivalent to <i>g-r,g+w</i> ; clears group read bit and sets group write bit.
<i>=g</i>	Sets owner bits to match group bits and sets other bits to match group bits.

9255 RATIONALE

9256 The functionality of *chmod* is described substantially through references to concepts defined in
 9257 the System Interfaces volume of IEEE Std. 1003.1-200x. In this way, there is less duplication of
 9258 effort required for describing the interactions of permissions. However, the behavior of this
 9259 utility is not described in terms of the *chmod()* function from the System Interfaces volume of
 9260 IEEE Std. 1003.1-200x because that specification requires certain side effects upon alternate file
 9261 access control mechanisms that might not be appropriate, depending on the implementation.

9262 Implementations that support mandatory file and record locking as specified by the 1984
 9263 /usr/group standard historically used the combination of set-group-ID bit set and group
 9264 execute bit clear to indicate mandatory locking. This condition is usually set or cleared with the
 9265 symbolic mode **perm** symbol **l** instead of the **perm** symbols **s** and **x** so that the mandatory
 9266 locking mode is not changed without explicit indication that that was what the user intended.
 9267 Therefore, the details on how the implementation treats these conditions must be defined in the
 9268 documentation. This volume of IEEE Std. 1003.1-200x does not require mandatory locking (nor
 9269 does the System Interfaces volume of IEEE Std. 1003.1-200x), but does allow it as an extension.
 9270 However, this volume of IEEE Std. 1003.1-200x does require that the *ls* and *chmod* utilities work

- 9271 consistently in this area. If *ls -l file* indicates that the set-group-ID bit is set, *chmod g-s file* must
9272 clear it (assuming appropriate privileges exist to change modes).
- 9273 The System V and BSD versions use different exit status codes. Some implementations used the
9274 exit status as a count of the number of errors that occurred; this practice is unworkable since it
9275 can overflow the range of valid exit status values. This problem is avoided here by specifying
9276 only 0 and >0 as exit values.
- 9277 The System Interfaces volume of IEEE Std. 1003.1-200x indicates that implementation-defined
9278 restrictions may cause the S_ISUID and S_ISGID bits to be ignored. This volume of
9279 IEEE Std. 1003.1-200x allows the *chmod* utility to choose to modify these bits before calling
9280 *chmod()* (or some function providing equivalent capabilities) for non-regular files. Among other
9281 things, this allows implementations that use the set-user-ID and set-group-ID bits on directories
9282 to enable extended features to handle these extensions in an intelligent manner.
- 9283 The **X perm** symbol was adopted from BSD-based systems because it provides commonly
9284 desired functionality when doing recursive (**-R** option) modifications. Similar functionality is
9285 not provided by the *find* utility. Historical BSD versions of *chmod*, however, only supported **X**
9286 with *op+*; it has been extended in this volume of IEEE Std. 1003.1-200x because it is also useful
9287 with *op=*. (It has also been added for *op-* even though it duplicates **x**, in this case, because it is
9288 intuitive and easier to explain.)
- 9289 The grammar was extended with the *permcop*y non-terminal to allow historical-practice forms of
9290 symbolic modes like **o=u -g** (that is, set the “other” permissions to the permissions of “owner”
9291 minus the permissions of “group”).
- 9292 **FUTURE DIRECTIONS**
- 9293 None.
- 9294 **SEE ALSO**
- 9295 *ls*, *umask*, the System Interfaces volume of IEEE Std. 1003.1-200x, *chmod()*
- 9296 **CHANGE HISTORY**
- 9297 First released in Issue 2.
- 9298 **Issue 4**
- 9299 Aligned with the ISO/IEC 9945-2:1993 standard.
- 9300 **Issue 6**
- 9301 The following new requirements on POSIX implementations derive from alignment with the
9302 Single UNIX Specification:
- 9303 • Octal modes have been kept and made mandatory despite being marked obsolescent in the
9304 previous version of this volume of IEEE Std. 1003.1-200x.
- 9305 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS
9306 section to “Default.”.

9307 **NAME**

9308 chown — change the file ownership

9309 **SYNOPSIS**

9310 chown -hR owner[:group] file ...

9311 chown -R [-H | -L | -P] owner[:group] file ...

9312 **DESCRIPTION**9313 The *chown* utility shall set the user ID of the file named by each *file* operand to the user ID
9314 specified by the *owner* operand.9315 For each *file* operand, it shall perform actions equivalent to the *chown()* function defined in the
9316 System Interfaces volume of IEEE Std. 1003.1-200x, called with the following arguments:

- 9317 1. The *file* operand shall be used as the *path* argument.
- 9318 2. The user ID indicated by the *owner* portion of the first operand shall be used as the *owner*
9319 argument.
- 9320 3. If the *group* portion of the first operand is given, the group ID indicated by it shall be used
9321 as the *group* argument; otherwise, the group ID of the file shall be used as the *group*
9322 argument.

9323 Unless *chown* is invoked by a process with appropriate privileges, the set-user-ID and set-
9324 group-ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and
9325 set-group-ID bits of other file types may be cleared.9326 **OPTIONS**9327 The *chown* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
9328 12.2, Utility Syntax Guidelines.

9329 The following options shall be supported by the implementation:

- 9330 **-h** If the system supports user IDs for symbolic links, for each *file* operand that names
9331 a file of type symbolic link, *chown* shall attempt to set the user ID of the symbolic
9332 link. If the system supports group IDs for symbolic links, and a group ID was
9333 specified, for each *file* operand that names a file of type symbolic link, *chown* shall
9334 attempt to set the group ID of the symbolic link. If the system does not support
9335 user or group IDs for symbolic links, for each *file* operand that names a file of type
9336 symbolic link, *chown* shall do nothing more with the current file and shall go on to
9337 any remaining files.
- 9338 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory
9339 is specified on the command line, *chown* shall change the user ID (and group ID, if
9340 specified) of the directory referenced by the symbolic link and all files in the file
9341 hierarchy below it.
- 9342 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory
9343 is specified on the command line or encountered during the traversal of a file
9344 hierarchy, *chown* shall change the user ID (and group ID, if specified) of the
9345 directory referenced by the symbolic link and all files in the file hierarchy below it.
- 9346 **-P** If the **-R** option is specified and a symbolic link is specified on the command line
9347 or encountered during the traversal of a file hierarchy, *chown* shall change the
9348 owner ID (and group ID, if specified) of the symbolic link if the system supports
9349 this operation. The *chown* utility shall not follow the symbolic link to any other
9350 part of the file hierarchy.

9351 **-R** Recursively change file user and group IDs. For each *file* operand that names a
 9352 directory, *chown* shall change the user ID (and group ID, if specified) of the
 9353 directory and all files in the file hierarchy below it. Unless a **-H**, **-L**, or **-P** option is
 9354 specified, it is unspecified which of these options will be used as the default.

9355 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be
 9356 considered an error. The last option specified shall determine the behavior of the utility.

9357 **OPERANDS**

9358 The following operands shall be supported:

9359 *owner[:group]* A user ID and optional group ID to be assigned to *file*. The application shall
 9360 ensure that the *owner* portion of this operand is a user name from the user database
 9361 or a numeric user ID. Either specifies a user ID to be given to each file named by
 9362 one of the *file* operands. If a numeric *owner* operand exists in the user database as a
 9363 user name, the user ID number associated with that user name is used as the user
 9364 ID. Similarly, if the *group* portion of this operand is present, it shall be a group
 9365 name from the group database or a numeric group ID. Either specifies a group ID
 9366 to be given to each file. If a numeric group operand exists in the group database as a
 9367 group name, the group ID number associated with that group name shall be used
 9368 as the group ID.

9369 *file* A path name of a file whose user ID is to be modified.

9370 **STDIN**

9371 Not used.

9372 **INPUT FILES**

9373 None.

9374 **ENVIRONMENT VARIABLES**

9375 The following environment variables shall affect the execution of *chown*:

9376 **LANG** Provide a default value for the internationalization variables that are unset or null.
 9377 If **LANG** is unset or null, the corresponding value from the implementation-
 9378 defined default locale shall be used. If any of the internationalization variables
 9379 contains an invalid setting, the utility shall behave as if none of the variables had
 9380 been defined.

9381 **LC_ALL** If set to a non-empty string value, override the values of all the other
 9382 internationalization variables.

9383 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 9384 characters (for example, single-byte as opposed to multi-byte characters in
 9385 arguments).

9386 **LC_MESSAGES**

9387 Determine the locale that should be used to affect the format and contents of
 9388 diagnostic messages written to standard error.

9389 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

9390 **ASYNCHRONOUS EVENTS**

9391 Default.

9392 **STDOUT**

9393 Not used.

9394 **STDERR**

9395 Used only for diagnostic messages.

9396 **OUTPUT FILES**

9397 None.

9398 **EXTENDED DESCRIPTION**

9399 None.

9400 **EXIT STATUS**

9401 The following exit values shall be returned:

9402 0 The utility executed successfully and all requested changes were made.

9403 >0 An error occurred.

9404 **CONSEQUENCES OF ERRORS**

9405 Default.

9406 **APPLICATION USAGE**

9407 Only the owner of a file or the user with appropriate privileges may change the owner or group of a file.

9409 Some systems restrict the use of *chown* to a user with appropriate privileges.

9410 **EXAMPLES**

9411 None.

9412 **RATIONALE**

9413 The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. These are masked by specifying only 0 and >0 as exit values.

9417 The functionality of *chown* is described substantially through references to functions in the System Interfaces volume of IEEE Std. 1003.1-200x. In this way, there is no duplication of effort required for describing the interactions of permissions, multiple groups, and so on.

9420 The 4.3 BSD method of specifying both owner and group was included in this volume of IEEE Std. 1003.1-200x because:

9422 • There are cases where the desired end condition could not be achieved using the *chgrp* and *chown* (that only changed the user ID) utilities. (If the current owner is not a member of the desired group and the desired owner is not a member of the current group, the *chown()* function could fail unless both owner and group are changed at the same time.)

9426 • Even if they could be changed independently, in cases where both are being changed, there is a 100% performance penalty caused by being forced to invoke both utilities.

9428 The BSD syntax *user[.group]* was changed to *user[:group]* in this volume of IEEE Std. 1003.1-200x because the period is a valid character in login names (as specified by the Base Definitions volume of IEEE Std. 1003.1-200x, login names consist of characters in the portable file name character set). The colon character was chosen as the replacement for the period character because it would never be allowed as a character in a user name or group name on historical implementations.

9434 The **-R** option is considered by some observers as an undesirable departure from the historical UNIX system tools approach; since a tool, *find*, already exists to recurse over directories, there seemed to be no good reason to require other tools to have to duplicate that functionality. However, the **-R** option was deemed an important user convenience, is far more efficient than

9438 forking a separate process for each element of the directory hierarchy, and is in widespread
9439 historical use.

9440 **FUTURE DIRECTIONS**

9441 None.

9442 **SEE ALSO**

9443 *chmod*, *chgrp*, the System Interfaces volume of IEEE Std. 1003.1-200x, *chown()*

9444 **CHANGE HISTORY**

9445 First released in Issue 2.

9446 **Issue 4**

9447 Aligned with the ISO/IEC 9945-2:1993 standard.

9448 **Issue 6**

9449 New options **-h**, **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These
9450 options affect the processing of symbolic links.

9451 The normative text is reworded to avoid use of the term “must” for application requirements.

9452 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS
9453 section is changed to “Default.”.

9454 **NAME**

9455 cksum — write file checksums and sizes

9456 **SYNOPSIS**9457 cksum [*file* ...]9458 **DESCRIPTION**

9459 The *cksum* utility shall calculate and write to standard output a cyclic redundancy check (CRC)
 9460 for each input file, and also write to standard output the number of octets in each file. The CRC
 9461 used is based on the polynomial used for CRC error checking in the ISO/IEC 8802-3:1996
 9462 standard (Ethernet).

9463 The encoding for the CRC checksum is defined by the generating polynomial:

$$9464 G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

9465 Mathematically, the CRC value corresponding to a given file shall be defined by the following
 9466 procedure:

- 9467 1. The *n* bits to be evaluated are considered to be the coefficients of a mod 2 polynomial *M(x)*
 9468 of degree *n*−1. These *n* bits are the bits from the file, with the most significant bit being the
 9469 most significant bit of the first octet of the file and the last bit being the least significant bit
 9470 of the last octet, padded with zero bits (if necessary) to achieve an integral number of
 9471 octets, followed by one or more octets representing the length of the file as a binary value,
 9472 least significant octet first. The smallest number of octets capable of representing this
 9473 integer shall be used.
- 9474 2. *M(x)* is multiplied by x^{32} (that is, shifted left 32 bits) and divided by *G(x)* using mod 2
 9475 division, producing a remainder *R(x)* of degree ≤ 31.
- 9476 3. The coefficients of *R(x)* are considered to be a 32-bit sequence.
- 9477 4. The bit sequence is complemented and the result is the CRC.

9478 **OPTIONS**

9479 None.

9480 **OPERANDS**

9481 The following operand shall be supported:

9482 *file* A path name of a file to be checked. If no *file* operands are specified, the standard
 9483 input is used.

9484 **STDIN**9485 The standard input is used only if no *file* operands are specified. See the INPUT FILES section.9486 **INPUT FILES**

9487 The input files can be any file type.

9488 **ENVIRONMENT VARIABLES**9489 The following environment variables shall affect the execution of *cksum*:

9490 *LANG* Provide a default value for the internationalization variables that are unset or null.
 9491 If *LANG* is unset or null, the corresponding value from the implementation-
 9492 defined default locale shall be used. If any of the internationalization variables
 9493 contains an invalid setting, the utility shall behave as if none of the variables had
 9494 been defined.

9495 *LC_ALL* If set to a non-empty string value, override the values of all the other
 9496 internationalization variables.

9497 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 9498 characters (for example, single-byte as opposed to multi-byte characters in
 9499 arguments).

9500 **LC_MESSAGES**
 9501 Determine the locale that should be used to affect the format and contents of
 9502 diagnostic messages written to standard error.

9503 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

9504 **ASYNCHRONOUS EVENTS**
 9505 Default.

9506 **STDOUT**
 9507 For each file processed successfully, the *cksum* utility shall write in the following format:
 9508 "%u %d %s\n", <checksum>, <# of octets>, <pathname>
 9509 If no *file* operand was specified, the path name and its leading <space> shall be omitted.

9510 **STDERR**
 9511 Used only for diagnostic messages.

9512 **OUTPUT FILES**
 9513 None.

9514 **EXTENDED DESCRIPTION**
 9515 None.

9516 **EXIT STATUS**
 9517 The following exit values shall be returned:
 9518 0 All files were processed successfully.
 9519 >0 An error occurred.

9520 **CONSEQUENCES OF ERRORS**
 9521 Default.

9522 **APPLICATION USAGE**
 9523 The *cksum* utility is typically used to quickly compare a suspect file against a trusted version of
 9524 the same, such as to ensure that files transmitted over noisy media arrive intact. However, this
 9525 comparison cannot be considered cryptographically secure. The chances of a damaged file
 9526 producing the same CRC as the original are small; deliberate deception is difficult, but probably
 9527 not impossible.

9528 Although input files to *cksum* can be any type, the results need not be what would be expected
 9529 on character special device files or on file types not described by the System Interfaces volume of
 9530 IEEE Std. 1003.1-200x. Since this volume of IEEE Std. 1003.1-200x does not specify the block size
 9531 used when doing input, checksums of character special files need not process all of the data in
 9532 those files.

9533 The algorithm is expressed in terms of a bitstream divided into octets. If a file is transmitted
 9534 between two systems and undergoes any data transformation (such as moving 8-bit characters
 9535 into 9-bit bytes or changing little-endian byte ordering to big-endian), identical CRC values
 9536 cannot be expected. Implementations performing such transformations may extend *cksum* to
 9537 handle such situations.

9538 **EXAMPLES**
 9539 None.

9540 **RATIONALE**

9541 The following C-language program can be used as a model to describe the algorithm. It assumes
 9542 that a **char** is one octet. It also assumes that the entire file is available for one pass through the
 9543 function. This was done for simplicity in demonstrating the algorithm, rather than as an
 9544 implementation model.

```

9545       static unsigned long crctab[] = {
9546       0x00000000,
9547       0x04c11db7, 0x09823b6e, 0x0d4326d9, 0x130476dc, 0x17c56b6b,
9548       0x1a864db2, 0x1e475005, 0x2608edb8, 0x22c9f00f, 0x2f8ad6d6,
9549       0x2b4bcb61, 0x350c9b64, 0x31cd86d3, 0x3c8ea00a, 0x384fbd8d,
9550       0x4c11db70, 0x48d0c6c7, 0x4593e01e, 0x4152fda9, 0x5f15adac,
9551       0x5bd4b01b, 0x569796c2, 0x52568b75, 0x6a1936c8, 0x6ed82b7f,
9552       0x639b0da6, 0x675a1011, 0x791d4014, 0x7ddc5da3, 0x709f7b7a,
9553       0x745e66cd, 0x9823b6e0, 0x9ce2ab57, 0x91a18d8e, 0x95609039,
9554       0x8b27c03c, 0x8fe6dd8b, 0x82a5fb52, 0x8664e6e5, 0xbe2b5b58,
9555       0xbaea46ef, 0xb7a96036, 0xb3687d81, 0xad2f2d84, 0xa9ee3033,
9556       0xa4ad16ea, 0xa06c0b5d, 0xd4326d90, 0xd0f37027, 0xddb056fe,
9557       0xd9714b49, 0xc7361b4c, 0xc3f706fb, 0xceb42022, 0xca753d95,
9558       0xf23a8028, 0xf6fb9d9f, 0xfbb8bb46, 0xff79a6f1, 0xe13ef6f4,
9559       0xe5ffeb43, 0xe8bccd9a, 0xec7dd02d, 0x34867077, 0x30476dc0,
9560       0x3d044b19, 0x39c556ae, 0x278206ab, 0x23431b1c, 0x2e003dc5,
9561       0x2ac12072, 0x128e9dcf, 0x164f8078, 0x1b0ca6a1, 0x1fcd8bb16,
9562       0x018aeb13, 0x054bf6a4, 0x0808d07d, 0x0cc9cdca, 0x7897ab07,
9563       0x7c56b6b0, 0x71159069, 0x75d48dde, 0x6b93ddd8, 0x6f52c06c,
9564       0x6211e6b5, 0x66d0fb02, 0x5e9f46bf, 0x5a5e5b08, 0x571d7dd1,
9565       0x53dc6066, 0x4d9b3063, 0x495a2dd4, 0x44190b0d, 0x40d816ba,
9566       0xaca5c697, 0xa864db20, 0xa527fd9, 0xa1e6e04e, 0xbf1b04b,
9567       0xbb60adfc, 0xb6238b25, 0xb2e29692, 0x8aad2b2f, 0x8e6c3698,
9568       0x832f1041, 0x87ee0df6, 0x99a95df3, 0x9d684044, 0x902b669d,
9569       0x94ea7b2a, 0xe0b41de7, 0xe4750050, 0xe9362689, 0xedf73b3e,
9570       0xf3b06b3b, 0xf771768c, 0xfa325055, 0xfef34de2, 0xc6bcf05f,
9571       0xc27dede8, 0xcf3ecb31, 0xcbbfd686, 0xd5b88683, 0xd1799b34,
9572       0xdc3abded, 0xd8fba05a, 0x690ce0ee, 0x6dcd9d59, 0x608edb80,
9573       0x644fc637, 0x7a089632, 0x7ec98b85, 0x738aad5c, 0x774bb0eb,
9574       0x4f040d56, 0x4bc510e1, 0x46863638, 0x42472b8f, 0x5c007b8a,
9575       0x58c1663d, 0x558240e4, 0x51435d53, 0x251d3b9e, 0x21dc2629,
9576       0x2c9f00f0, 0x285e1d47, 0x36194d42, 0x32d850f5, 0x3f9b762c,
9577       0x3b5a6b9b, 0x0315d626, 0x07d4cb91, 0x0a97ed48, 0x0e56f0ff,
9578       0x1011a0fa, 0x14d0bd4d, 0x19939b94, 0x1d528623, 0xf12f560e,
9579       0xf5ee4bb9, 0xf8ad6d60, 0xfc6c70d7, 0xe22b20d2, 0xe6ea3d65,
9580       0xeba91bbc, 0xef68060b, 0xd727bbb6, 0xd3e6a601, 0xdea580d8,
9581       0xda649d6f, 0xc423cd6a, 0xc0e2d0dd, 0xcda1f604, 0xc960ebb3,
9582       0xbd3e8d7e, 0xb9ff90c9, 0xb4bcb610, 0xb07daba7, 0xae3afba2,
9583       0xaafb615, 0xa7b8c0cc, 0xa379dd7b, 0x9b3660c6, 0x9ff77d71,
9584       0x92b45ba8, 0x9675461f, 0x8832161a, 0x8cf30bad, 0x81b02d74,
9585       0x857130c3, 0x5d8a9099, 0x594b8d2e, 0x5408abf7, 0x50c9b640,
9586       0x4e8ee645, 0x4a4ffb2, 0x470cdd2b, 0x43cdc09c, 0x7b827d21,
9587       0x7f436096, 0x7200464f, 0x76c15bf8, 0x68860bfd, 0x6c47164a,
9588       0x61043093, 0x65c52d24, 0x119b4be9, 0x155a565e, 0x18197087,

```

```

9589     0x1cd86d30, 0x029f3d35, 0x065e2082, 0x0b1d065b, 0x0fdc1bec,
9590     0x3793a651, 0x3352bbe6, 0x3e119d3f, 0x3ad08088, 0x2497d08d,
9591     0x2056cd3a, 0x2d15ebe3, 0x29d4f654, 0xc5a92679, 0xc1683bce,
9592     0xcc2b1d17, 0xc8ea00a0, 0xd6ad50a5, 0xd26c4d12, 0xdf2f6bcb,
9593     0xdbee767c, 0xe3a1cbc1, 0xe760d676, 0xea23f0af, 0xee2ed18,
9594     0xf0a5bd1d, 0xf464a0aa, 0xf9278673, 0xfde69bc4, 0x89b8fd09,
9595     0x8d79e0be, 0x803ac667, 0x84fbdbd0, 0x9abc8bd5, 0x9e7d9662,
9596     0x933eb0bb, 0x97ffad0c, 0xafb010b1, 0xab710d06, 0xa6322bdf,
9597     0xa2f33668, 0xbcb4666d, 0xb8757bda, 0xb5365d03, 0xb1f740b4
9598     };

9599     unsigned long memcrc(const unsigned char *b, size_t n)
9600     {
9601     /* Input arguments:
9602     *  const char*   b == byte sequence to checksum
9603     *  size_t       n == length of sequence
9604     */

9605         register unsigned   i, c, s = 0;

9606         for (i = n; i > 0; --i) {
9607             c = (unsigned)(*b++);
9608             s = (s << 8) ^ crctab[(s >> 24) ^ c];
9609         }

9610         /* Extend with the length of the string. */
9611         while (n != 0) {
9612             c = n & 0377;
9613             n >>= 8;
9614             s = (s << 8) ^ crctab[(s >> 24) ^ c];
9615         }

9616         return ~s;
9617     }

```

9618 The historical practice of writing the number of “blocks” has been changed to writing the
9619 number of octets, since the latter is not only more useful, but also since historical
9620 implementations have not been consistent in defining what a “block” meant. Octets are used
9621 instead of bytes because bytes can differ in size between systems.

9622 The algorithm used was selected to increase the operational robustness of *cksum*. Neither the
9623 System V nor BSD *sum* algorithm was selected. Since each of these was different and each was
9624 the default behavior on those systems, no realistic compromise was available if either were
9625 selected—some set of historical applications would break. Therefore, the name was changed to
9626 *cksum*. Although the historical *sum* commands will probably continue to be provided for many
9627 years, programs designed for portability across systems should use the new name.

9628 The algorithm selected is based on that used by the ISO/IEC 8802-3: 1996 standard (Ethernet) for
9629 the frame check sequence field. The algorithm used does not match the technical definition of a
9630 *checksum*; the term is used for historical reasons. The length of the file is included in the CRC
9631 calculation because this parallels inclusion of a length field by Ethernet in its CRC, but also
9632 because it guards against inadvertent collisions between files that begin with different series of
9633 zero octets. The chance that two different files produce identical CRCs is much greater when
9634 their lengths are not considered. Keeping the length and the checksum of the file itself separate
9635 would yield a slightly more robust algorithm, but historical usage has always been that a single
9636 number (the checksum as printed) represents the signature of the file. It was decided that

9637 historical usage was the more important consideration.

9638 Early proposals contained modifications to the Ethernet algorithm that involved extracting table
9639 values whenever an intermediate result became zero. This was demonstrated to be less robust
9640 than the current method and mathematically difficult to describe or justify.

9641 The calculation used is identical to that given in pseudo-code in the referenced Sarwate article.
9642 The pseudo-code rendition is:

```
9643 X ← 0; Y ← 0;  
9644 for i ← m - 1 step -1 until 0 do  
9645     begin  
9646         T ← X(1) ^ A[i];  
9647         X(1) ← X(0); X(0) ← Y(1); Y(1) ← Y(0); Y(0) ← 0;  
9648         comment: f[T] and f'[T] denote the T-th words in the  
9649                 table f and f' ;  
9650         X ← X ^ f[T]; Y ← Y ^ f'[T];  
9651     end
```

9652 The pseudo-code is reproduced exactly as given; however, note that in the case of *cksum*, **A[i]**
9653 represents a byte of the file, the words **X** and **Y** are treated as a single 32-bit value, and the tables
9654 **f** and **f'** are a single table containing 32-bit values.

9655 The referenced Sarwate article also discusses generating the table.

9656 **FUTURE DIRECTIONS**

9657 None.

9658 **SEE ALSO**

9659 None.

9660 **CHANGE HISTORY**

9661 First released in Issue 4.

9662 **NAME**

9663 cmp — compare two files

9664 **SYNOPSIS**9665 cmp [-l | -s] *file1 file2*9666 **DESCRIPTION**

9667 The *cmp* utility shall compare two files. The *cmp* utility writes no output if the files are the same.
 9668 Under default options, if they differ, it shall write to standard output the byte and line number at
 9669 which the first difference occurred. Bytes and lines shall be numbered beginning with 1.

9670 **OPTIONS**

9671 The *cmp* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 9672 12.2, Utility Syntax Guidelines.

9673 The following options shall be supported:

9674 -l (Lowercase ell.) Write the byte number (decimal) and the differing bytes (octal) for
 9675 each difference.

9676 -s Write nothing for differing files; return exit status only.

9677 **OPERANDS**

9678 The following operands shall be supported:

9679 *file1* A path name of the first file to be compared. If *file1* is '-', the standard input shall
 9680 be used.

9681 *file2* A path name of the second file to be compared. If *file2* is '-', the standard input
 9682 shall be used.

9683 If both *file1* and *file2* refer to standard input or refer to the same FIFO special, block special, or
 9684 character special file, the results are undefined.

9685 **STDIN**

9686 The standard input shall be used only if the *file1* or *file2* operand refers to standard input. See the
 9687 INPUT FILES section.

9688 **INPUT FILES**

9689 The input files can be any file type.

9690 **ENVIRONMENT VARIABLES**9691 The following environment variables shall affect the execution of *cmp*:

9692 LANG Provide a default value for the internationalization variables that are unset or null.
 9693 If LANG is unset or null, the corresponding value from the implementation-
 9694 defined default locale shall be used. If any of the internationalization variables
 9695 contains an invalid setting, the utility shall behave as if none of the variables had
 9696 been defined.

9697 LC_ALL If set to a non-empty string value, override the values of all the other
 9698 internationalization variables.

9699 LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as
 9700 characters (for example, single-byte as opposed to multi-byte characters in
 9701 arguments).

9702 LC_MESSAGES

9703 Determine the locale that should be used to affect the format and contents of
 9704 diagnostic messages written to standard error and informative messages written to
 9705 standard output.

9706 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

9707 **ASYNCHRONOUS EVENTS**

9708 Default.

9709 **STDOUT**

9710 In the POSIX locale, results of the comparison shall be written to standard output. When no
9711 options are used, the format shall be:

9712 "%s %s differ: char %d, line %d\n", *file1*, *file2*,
9713 <byte number>, <line number>

9714 When the **-I** option is used, the format shall be:

9715 "%d %o %o\n", <byte number>, <differing byte>,
9716 <differing byte>

9717 for each byte that differs. The first <differing byte> number is from *file1* while the second is from
9718 *file2*. In both cases, <byte number> shall be relative to the beginning of the file, beginning with 1.

9719 No output shall be written to standard output when the **-s** option is used.

9720 **STDERR**

9721 Used only for diagnostic messages. If *file1* and *file2* are identical for the entire length of the
9722 shorter file, in the POSIX locale the following diagnostic message shall be written, unless the **-s**
9723 option is specified:

9724 "cmp: EOF on %s%s\n", <name of shorter file>, <additional info>

9725 The <additional info> field shall either be null or a string that starts with a <blank> character and
9726 contains no <newline> characters. Some systems report on the number of lines in this case.

9727 **OUTPUT FILES**

9728 None.

9729 **EXTENDED DESCRIPTION**

9730 None.

9731 **EXIT STATUS**

9732 The following exit values shall be returned:

9733 0 The files are identical.

9734 1 The files are different; this includes the case where one file is identical to the first part of the
9735 other.

9736 >1 An error occurred.

9737 **CONSEQUENCES OF ERRORS**

9738 Default.

9739 **APPLICATION USAGE**

9740 Although input files to *cmp* can be any type, the results might not be what would be expected on
9741 character special device files or on file types not described by the System Interfaces volume of
9742 IEEE Std. 1003.1-200x. Since this volume of IEEE Std. 1003.1-200x does not specify the block size
9743 used when doing input, comparisons of character special files need not compare all of the data
9744 in those files.

9745 For files which are not text files, line numbers simply reflect the presence of a <newline>
9746 character, without any implication that the file is organized into lines.

9747 **EXAMPLES**

9748 None.

9749 **RATIONALE**

9750 The global language in Section 1.11 (on page 2224) indicates that using two mutually-exclusive
9751 options together produces unspecified results. Some System V implementations consider the
9752 option usage:

9753 `cmp -l -s ...`

9754 to be an error. They also treat:

9755 `cmp -s -l ...`

9756 as if no options were specified. Both of these behaviors are considered bugs, but are allowed.

9757 The word **char** in the standard output format comes from historical usage, even though it is
9758 actually a byte number. When *cmp* is supported in other locales, implementations are
9759 encouraged to use the word *byte* or its equivalent in another language. Users should not
9760 interpret this difference to indicate that the functionality of the utility changed between locales.

9761 Some systems report on the number of lines in the identical-but-shorter file case. This is allowed
9762 by the inclusion of the *<additional info>* fields in the output format. The restriction on having a
9763 leading *<blank>* and no *<newline>*s is to make parsing for the file name easier. It is recognized
9764 that some file names containing white-space characters make parsing difficult anyway, but the
9765 restriction does aid programs used on systems where the names are predominantly well
9766 behaved.

9767 **FUTURE DIRECTIONS**

9768 None.

9769 **SEE ALSO**9770 *comm, diff*9771 **CHANGE HISTORY**

9772 First released in Issue 2.

9773 **Issue 4**

9774 Aligned with the ISO/IEC 9945-2:1993 standard.

9775 **NAME**

9776 comm — select or reject lines common to two files

9777 **SYNOPSIS**

9778 comm [-123] *file1 file2*

9779 **DESCRIPTION**

9780 The *comm* utility shall read *file1* and *file2*, which should be ordered in the current collating
9781 sequence, and produce three text columns as output: lines only in *file1*, lines only in *file2*, and
9782 lines in both files.

9783 If the lines in both files are not ordered according to the collating sequence of the current locale,
9784 the results are unspecified.

9785 **OPTIONS**

9786 The *comm* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
9787 12.2, Utility Syntax Guidelines.

9788 The following options shall be supported:

9789 -1 Suppress the output column of lines unique to *file1*.

9790 -2 Suppress the output column of lines unique to *file2*.

9791 -3 Suppress the output column of lines duplicated in *file1* and *file2*.

9792 **OPERANDS**

9793 The following operands shall be supported:

9794 *file1* A path name of the first file to be compared. If *file1* is '-', the standard input is
9795 used.

9796 *file2* A path name of the second file to be compared. If *file2* is '-', the standard input is
9797 used.

9798 If both *file1* and *file2* refer to standard input or to the same FIFO special, block special, or
9799 character special file, the results are undefined.

9800 **STDIN**

9801 The standard input shall be used only if one of the *file1* or *file2* operands refers to standard input.
9802 See the INPUT FILES section.

9803 **INPUT FILES**

9804 The input files shall be text files.

9805 **ENVIRONMENT VARIABLES**

9806 The following environment variables shall affect the execution of *comm*:

9807 *LANG* Provide a default value for the internationalization variables that are unset or null.
9808 If *LANG* is unset or null, the corresponding value from the implementation-
9809 defined default locale shall be used. If any of the internationalization variables
9810 contains an invalid setting, the utility shall behave as if none of the variables had
9811 been defined.

9812 *LC_ALL* If set to a non-empty string value, override the values of all the other
9813 internationalization variables.

9814 *LC_COLLATE*

9815 Determine the locale for the collating sequence *comm* expects to have been used
9816 when the input files were sorted.

- 9817 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 9818 characters (for example, single-byte as opposed to multi-byte characters in
 9819 arguments and input files).
- 9820 **LC_MESSAGES**
 9821 Determine the locale that should be used to affect the format and contents of
 9822 diagnostic messages written to standard error.
- 9823 **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 9824 **ASYNCHRONOUS EVENTS**
 9825 Default.
- 9826 **STDOUT**
 9827 The *comm* utility shall produce output depending on the options selected. If the **-1**, **-2**, and **-3**
 9828 options are all selected, *comm* shall write nothing to standard output.
- 9829 If the **-1** option is not selected, lines contained only in *file1* shall be written using the format:
 9830 "%s\n", <line in file1>
- 9831 If the **-2** option is not selected, lines contained only in *file2* are written using the format:
 9832 "%s%s\n", <lead>, <line in file2>
- 9833 where the string <lead> is as follows:
 9834 <tab> The **-1** option is not selected.
 9835 null string The **-1** option is selected.
- 9836 If the **-3** option is not selected, lines contained in both files shall be written using the format:
 9837 "%s%s\n", <lead>, <line in both>
- 9838 where the string <lead> is as follows:
 9839 <tab><tab> Neither the **-1** nor the **-2** option is selected.
 9840 <tab> Exactly one of the **-1** and **-2** options is selected.
 9841 null string Both the **-1** and **-2** options are selected.
- 9842 If the input files were ordered according to the collating sequence of the current locale, the lines
 9843 written shall be in the collating sequence of the original lines.
- 9844 **STDERR**
 9845 Used only for diagnostic messages.
- 9846 **OUTPUT FILES**
 9847 None.
- 9848 **EXTENDED DESCRIPTION**
 9849 None.
- 9850 **EXIT STATUS**
 9851 The following exit values shall be returned:
 9852 0 All input files were successfully output as specified.
 9853 >0 An error occurred.

9854 **CONSEQUENCES OF ERRORS**

9855 Default.

9856 **APPLICATION USAGE**9857 If the input files are not properly presorted, the output of *comm* might not be useful.9858 **EXAMPLES**

9859 If a file named **xcu** contains a sorted list of the utilities in this volume of IEEE Std. 1003.1-200x, a
9860 file named **xpg3** contains a sorted list of the utilities specified in the X/Open Portability Guide,
9861 Issue 3, and a file named **svid89** contains a sorted list of the utilities in the System V Interface
9862 Definition Third Edition:

9863 `comm -23 xcu xpg3 | comm -23 - svid89`9864 would print a list of utilities in this volume of IEEE Std. 1003.1-200x not specified by either of the
9865 other documents:9866 `comm -12 xcu xpg3 | comm -12 - svid89`

9867 would print a list of utilities specified by all three documents, and:

9868 `comm -12 xpg3 svid89 | comm -23 - xcu`9869 would print a list of utilities specified by both XPG3 and the SVID, but not specified in this
9870 volume of IEEE Std. 1003.1-200x.9871 **RATIONALE**

9872 None.

9873 **FUTURE DIRECTIONS**

9874 None.

9875 **SEE ALSO**9876 *cmp, diff, sort, uniq*9877 **CHANGE HISTORY**

9878 First released in Issue 2.

9879 **Issue 4**

9880 Aligned with the ISO/IEC 9945-2:1993 standard.

9881 **Issue 6**

9882 The normative text is reworded to avoid use of the term “must” for application requirements.

9883 **NAME**9884 `command` — execute a simple command9885 **SYNOPSIS**9886 `command [-p] command_name [argument ...]`9887 UP `command [-v | -V] command_name`

9888

9889 **DESCRIPTION**9890 The *command* utility shall cause the shell to treat the arguments as a simple command, suppressing the shell function lookup that is described in Section 2.9.1.1 (on page 2257), item 1b.9892 If the *command_name* is the same as the name of one of the special built-in utilities, the special properties in the enumerated list at the beginning of Section 2.15 (on page 2276) shall not occur. In every other respect, if *command_name* is not the name of a function, the effect of *command* shall be the same as omitting *command*.9896 On systems supporting the User Portability Utilities option, the *command* utility also shall provide information concerning how a command name is interpreted by the shell; see `-v` and `-V`.9899 **OPTIONS**9900 The *command* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

9902 The following options shall be supported:

9903 `-p` Perform the command search using a default value for *PATH* that is guaranteed to find all of the standard utilities.9905 `-v` (On systems supporting the User Portability Utilities option.) Write a string to standard output that indicates the path name or command that will be used by the shell, in the current shell execution environment (see Section 2.13 (on page 2273)), to invoke *command_name*.9909 • Utilities, regular built-in utilities, *command_names* including a slash character, and any implementation-defined functions that are found using the *PATH* variable (as described in Section 2.9.1.1 (on page 2257)), shall be written as absolute path names.9913 • Shell functions, special built-in utilities, regular built-in utilities not associated with a *PATH* search, and shell reserved words shall be written as just their names.

9916 • An alias shall be written as a command line that represents its alias definition.

9917 • Otherwise, no output shall be written and the exit status shall reflect that the name was not found.

9919 `-V` (On systems supporting the User Portability Utilities option.) Write a string to standard output that indicates how the name given in the *command_name* operand will be interpreted by the shell, in the current shell execution environment (see Section 2.13 (on page 2273)). Although the format of this string is unspecified, it shall indicate in which of the following categories *command_name* falls and shall include the information stated:9925 • Utilities, regular built-in utilities, and any implementation-defined functions that are found using the *PATH* variable (as described in Section 2.9.1.1 (on page 2257)), shall be identified as such and include the absolute path name in the

- 9928 string.
- 9929 • Other shell functions shall be identified as functions.
- 9930 • Aliases shall be identified as aliases and their definitions included in the string.
- 9931 • Special built-in utilities shall be identified as special built-in utilities.
- 9932 • Regular built-in utilities not associated with a *PATH* search shall be identified
- 9933 as regular built-in utilities. (The term “regular” need not be used.)
- 9934 • Shell reserved words shall be identified as reserved words.
- 9935 **OPERANDS**
- 9936 The following operands shall be supported:
- 9937 *argument* One of the strings treated as an argument to *command_name*.
- 9938 *command_name*
- 9939 The name of a utility or a special built-in utility.
- 9940 **STDIN**
- 9941 Not used.
- 9942 **INPUT FILES**
- 9943 None.
- 9944 **ENVIRONMENT VARIABLES**
- 9945 The following environment variables shall affect the execution of *command*:
- 9946 *LANG* Provide a default value for the internationalization variables that are unset or null.
- 9947 If *LANG* is unset or null, the corresponding value from the implementation-
- 9948 defined default locale shall be used. If any of the internationalization variables
- 9949 contains an invalid setting, the utility shall behave as if none of the variables had
- 9950 been defined.
- 9951 *LC_ALL* If set to a non-empty string value, override the values of all the other
- 9952 internationalization variables.
- 9953 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
- 9954 characters (for example, single-byte as opposed to multi-byte characters in
- 9955 arguments).
- 9956 *LC_MESSAGES*
- 9957 Determine the locale that should be used to affect the format and contents of
- 9958 diagnostic messages written to standard error and informative messages written to
- 9959 standard output.
- 9960 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 9961 *PATH* Determine the search path used during the command search described in Section
- 9962 2.9.1.1 (on page 2257), except as described under the *-p* option.
- 9963 **ASYNCHRONOUS EVENTS**
- 9964 Default.
- 9965 **STDOUT**
- 9966 When the *-v* option is specified, standard output shall be formatted as:
- 9967 "%s\n", *<pathname or command>*
- 9968 When the *-V* option is specified, standard output shall be formatted as:

9969 "%s\n", <unspecified>

9970 **STDERR**

9971 Used only for diagnostic messages.

9972 **OUTPUT FILES**

9973 None.

9974 **EXTENDED DESCRIPTION**

9975 None.

9976 **EXIT STATUS**

9977 When the `-v` or `-V` options are specified, the following exit values shall be returned:

9978 0 Successful completion.

9979 >0 The *command_name* could not be found or an error occurred.

9980 Otherwise, the following exit values shall be returned:

9981 126 The utility specified by *command_name* was found but could not be invoked.

9982 127 An error occurred in the *command* utility or the utility specified by *command_name* could not
9983 be found.

9984 Otherwise, the exit status of *command* shall be that of the simple command specified by the
9985 arguments to *command*.

9986 **CONSEQUENCES OF ERRORS**

9987 Default.

9988 **APPLICATION USAGE**

9989 The order for command search allows functions to override regular built-ins and path searches.
9990 This utility is necessary to allow functions that have the same name as a utility to call the utility
9991 (instead of a recursive call to the function).

9992 The system default path is available using *getconf*; however, since *getconf* may need to have the
9993 *PATH* set up before it can be called itself, the following can be used:

9994 `command -p getconf _CS_PATH`

9995 There are some advantages to suppressing the special characteristics of special built-ins on
9996 occasion. For example:

9997 `command exec > unwritable-file`

9998 does not cause a non-interactive script to abort, so that the output status can be checked by the
9999 script.

10000 The *command*, *env*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an
10001 error occurs so that applications can distinguish “failure to find a utility” from “invoked utility
10002 exited with an error indication”. The value 127 was chosen because it is not commonly used for
10003 other meanings; most utilities use small values for “normal error conditions” and the values
10004 above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen
10005 in a similar manner to indicate that the utility could be found, but not invoked. Some scripts
10006 produce meaningful error messages differentiating the 126 and 127 cases. The distinction
10007 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
10008 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
10009 any other reason.

10010 Since the `-v` and `-V` options of *command* produce output in relation to the current shell execution
10011 environment, *command* is generally provided as a shell regular built-in. If it is called in a subshell

10012 or separate utility execution environment, such as one of the following:

```
10013 (PATH=foo command -v)
10014 nohup command -v
```

10015 it does not necessarily produce correct results. For example, when called with *nohup* or an *exec*
10016 function, in a separate utility execution environment, most implementations are not able to
10017 identify aliases, functions, or special built-ins.

10018 Two types of regular built-ins could be encountered on a system and these are described
10019 separately by *command*. The description of command search in Section 2.9.1.1 (on page 2257)
10020 allows for a standard utility to be implemented as a regular built-in as long as it is found in the
10021 appropriate place in a *PATH* search. So, for example, *command -v true* might yield */bin/true* or
10022 some similar path name. Other implementation-defined utilities that are not defined by this
10023 volume of IEEE Std. 1003.1-200x might exist only as built-ins and have no path name associated
10024 with them. These produce output identified as (regular) built-ins. Applications encountering
10025 these are not able to count on *execing* them, using them with *nohup*, overriding them with a
10026 different *PATH*, and so on.

10027 EXAMPLES

10028 1. Make a version of *cd* that always prints out the new working directory exactly once:

```
10029 cd() {
10030     command cd "$@" >/dev/null
10031     pwd
10032 }
```

10033 2. Start off a “secure shell script” in which the script avoids being spoofed by its parent:

```
10034 IFS='
10035 '
10036 # The preceding value should be <space><tab><newline>.
10037 # Set IFS to its default value.
```

```
10038 \unalias -a
10039 # Unset all possible aliases.
10040 # Note that unalias is escaped to prevent an alias
10041 # being used for unalias.
```

```
10042 unset -f command
10043 # Ensure command is not a user function.
```

```
10044 PATH="$(command -p getconf _CS_PATH):$PATH"
10045 # Put on a reliable PATH prefix.
```

```
10046 # ...
```

10047 At this point, given correct permissions on the directories called by *PATH*, the script has
10048 the ability to ensure that any utility it calls is the intended one. It is being very cautious
10049 because it assumes that implementation extensions may be present that would allow user
10050 functions to exist when it is invoked; this capability is not specified by this volume of
10051 IEEE Std. 1003.1-200x, but it is not prohibited as an extension. For example, the *ENV*
10052 variable precedes the invocation of the script with a user start-up script. Such a script
10053 could define functions to spoof the application.

10054 **RATIONALE**

10055 Since *command* is a regular built-in utility it is always found prior to the *PATH* search.

10056 There is nothing in the description of *command* that implies the command line is parsed any
10057 differently from that of any other simple command. For example:

10058 `command a | b ; c`

10059 is not parsed in any special way that causes ' | ' or ' ; ' to be treated other than a pipe operator
10060 or semicolon or that prevents function lookup on **b** or **c**.

10061 The *command* utility is somewhat similar to the Eighth Edition shell *builtin* command, but since
10062 *command* also goes to the file system to search for utilities, the name *builtin* would not be
10063 intuitive.

10064 The *command* utility is most likely to be provided as a regular built-in. It is not listed as a special
10065 built-in for the following reasons:

- 10066 • The removal of exportable functions made the special precedence of a special built-in
10067 unnecessary.
- 10068 • A special built-in has special properties (see Section 2.15 (on page 2276)) that were
10069 inappropriate for invoking other utilities. For example, two commands such as:

10070 `date > unwritable-file`

10071 `command date > unwritable-file`

10072 would have entirely different results; in a non-interactive script, the former would continue
10073 to execute the next command, the latter would abort. Introducing this semantic difference
10074 along with suppressing functions was seen to be non-intuitive.

10075 The **-p** option is present because it is useful to be able to ensure a safe path search that finds all
10076 the POSIX Shell and Utilities standard utilities. This search might not be identical to the one that
10077 occurs through one of the POSIX System Interfaces *exec* functions when *PATH* is unset. At the
10078 very least, this feature is required to allow the script to access the correct version of *getconf* so
10079 that the value of the default path can be accurately retrieved.

10080 The *command* **-v** and **-V** options were added to satisfy requirements from users that are
10081 currently accomplished by three different historical utilities: *type* in the System V shell, *whence* in
10082 the KornShell, and *which* in the C shell. Since there is no historical agreement on how and what
10083 to accomplish here, the POSIX *command* utility was enhanced and the historical utilities were left
10084 unmodified. The C shell *which* merely conducts a path search. The KornShell *whence* is more
10085 elaborate—in addition to the categories required by POSIX, it also reports on tracked aliases,
10086 exported aliases, and undefined functions.

10087 The output format of **-V** was left mostly unspecified because human users are its only audience.
10088 Applications should not be written to care about this information; they can use the output of **-v**
10089 to differentiate between various types of commands, but the additional information that may be
10090 emitted by the more verbose **-V** is not needed and should not be arbitrarily constrained in its
10091 verbosity or localization for application parsing reasons.

10092 **FUTURE DIRECTIONS**

10093 None.

10094 **SEE ALSO**

10095 *sh*, *type*

10096 **CHANGE HISTORY**

10097 First released in Issue 4.

10098 NAME

10099 compress — compress data

10100 SYNOPSIS

10101 xSI compress [-fv][-b *bits*][*file* ...]10102 compress [-cfv][-b *bits*][*file*]

10103

10104 DESCRIPTION

10105 **Notes to Reviewers**10106 *This section with side shading will not appear in the final copy. - Ed.*10107 We need to cite the patent number for Lempel-Ziv coding; if anyone knows what it is, please let
10108 us know.10109 The *compress* utility shall attempt to reduce the size of the named files by using adaptive
10110 Lempel-Ziv coding algorithm. On systems not supporting adaptive Lempel-Ziv coding
10111 algorithm, the input files shall not be changed and an error value greater than two shall be
10112 returned. Except when the output is to the standard output, each file shall be replaced by one
10113 with the extension *.Z*. If the invoking process has appropriate privileges, the ownership, modes,
10114 access time, and modification time of the original file are preserved. If appending the *.Z* to the
10115 file name would make the name exceed {NAME_MAX} bytes, the command shall fail. If no files
10116 are specified, the standard input shall be compressed to the standard output.

10117 OPTIONS

10118 The *compress* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x,
10119 Section 12.2, Utility Syntax Guidelines.

10120 The following options shall be supported:

10121 **-b *bits*** Specify the maximum number of bits to use in a code. For a portable application,
10122 the *bits* argument shall be:10123 $9 \leq bits \leq 14$ 10124 The implementation may allow *bits* values of greater than 14. The default is 14, 15,
10125 or 16.10126 **-c** Cause *compress* to write to the standard output; the input file is not changed, and
10127 no *.Z* files are created.10128 **-f** Force compression of *file*, even if it does not actually reduce the size of the file, or if
10129 the corresponding *file.Z* file already exists. If the **-f** option is not given, and the
10130 process is not running in the background, the user is prompted as to whether an
10131 existing *file.Z* file should be overwritten.10132 **-v** Write the percentage reduction of each file to standard error.

10133 OPERANDS

10134 The following operand shall be supported:

10135 *file* A path name of a file to be compressed.

10136 STDIN

10137 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '- '.

10138 **INPUT FILES**

10139 If *file* operands are specified, the input files contain the data to be compressed.

10140 **ENVIRONMENT VARIABLES**

10141 The following environment variables shall affect the execution of *compress*:

10142 *LANG* Provide a default value for the internationalization variables that are unset or null.
10143 If *LANG* is unset or null, the corresponding value from the implementation-
10144 defined default locale shall be used. If any of the internationalization variables
10145 contains an invalid setting, the utility shall behave as if none of the variables had
10146 been defined.

10147 *LC_ALL* If set to a non-empty string value, override the values of all the other
10148 internationalization variables.

10149 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
10150 characters (for example, single-byte as opposed to multi-byte characters in
10151 arguments).

10152 *LC_MESSAGES*

10153 Determine the locale that should be used to affect the format and contents of
10154 diagnostic messages written to standard error.

10155 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

10156 **ASYNCHRONOUS EVENTS**

10157 Default.

10158 **STDOUT**

10159 If no *file* operands are specified, or if a *file* operand is '-', or if the *-c* option is specified, the
10160 standard output contains the compressed output.

10161 **STDERR**

10162 Used for all diagnostic and prompt messages and the output from *-v*.

10163 **OUTPUT FILES**

10164 The output files shall contain the compressed output. The format of compressed files is
10165 unspecified and interchange of such files between implementations (including access via
10166 unspecified file sharing mechanisms) is not required by IEEE Std. 1003.1-200x.

10167 **EXTENDED DESCRIPTION**

10168 None.

10169 **EXIT STATUS**

10170 The following exit values shall be returned:

10171 0 Successful completion.

10172 1 An error occurred.

10173 2 One or more files were not compressed because they would have increased in size (and the
10174 *-f* option was not specified).

10175 >2 An error occurred.

10176 **CONSEQUENCES OF ERRORS**

10177 The input file shall remain unmodified.

10178 **APPLICATION USAGE**

10179 The amount of compression obtained depends on the size of the input, the number of *bits* per
10180 code, and the distribution of common substrings. Typically, text such as source code or English
10181 is reduced by 50-60%. Compression is generally much better than that achieved by Huffman
10182 coding or adaptive Huffman coding (*compact*), and takes less time to compute.

10183 Although *compress* strictly follows the default actions upon receipt of a signal or when an error
10184 occurs, some unexpected results may occur. In some implementations it is likely that a partially
10185 compressed file is left in place, alongside its uncompressed input file. Since the general
10186 operation of *compress* is to delete the uncompressed file only after the *.Z* file has been
10187 successfully filled, an application should always carefully check the exit status of *compress* before
10188 arbitrarily deleting files that have like-named neighbors with *.Z* suffixes.

10189 The limit of 14 on the *bits* option-argument is to achieve portability to all systems (within the
10190 restrictions imposed by the lack of an explicit published file format). Some systems based on
10191 16-bit architectures cannot support 15 or 16-bit uncompression.

10192 **EXAMPLES**

10193 None.

10194 **RATIONALE**

10195 None.

10196 **FUTURE DIRECTIONS**

10197 None.

10198 **SEE ALSO**

10199 *uncompress*, *zcat*

10200 **CHANGE HISTORY**

10201 First released in Issue 4.

10202 **Issue 4, Version 2**

10203 The DESCRIPTION section is clarified to state that the ownership, modes, access time, and
10204 modification time of the original file are preserved if the invoking process has appropriate
10205 privileges.

10206 The STDOUT section includes the case where a *file* operand is *'-'*.

10207 **Issue 6**

10208 The normative text is reworded to avoid use of the term “must” for application requirements.

10209 An error case is added for systems not supporting adaptive Lempel-Ziv coding.

10210 NAME

10211 cp — copy files

10212 SYNOPSIS

10213 cp [-fip] *source_file target_file*10214 cp [-fip] *source_file ... target*10215 cp -R [-H | -L | -P][-fip] *source_file ... target*10216 OB cp -r [-H | -L | -P][-fip] *source_file ... target*

10217 DESCRIPTION

10218 The first synopsis form is denoted by two operands, neither of which are existing files of type
 10219 directory. The *cp* utility shall copy the contents of *source_file* (or, if *source_file* is a file of type
 10220 symbolic link, the contents of the file referenced by *source_file*) to the destination path named by
 10221 *target_file*.

10222 The second synopsis form is denoted by two or more operands where the **-R** or **-r** options are
 10223 not specified and the first synopsis form is not applicable. It shall be an error if any *source_file* is a
 10224 file of type directory, if *target* does not exist, or if *target* is a file of a type defined by the System
 10225 Interfaces volume of IEEE Std. 1003.1-200x, but is not a file of type directory. The *cp* utility shall
 10226 copy the contents of each *source_file* (or, if *source_file* is a file of type symbolic link, the contents
 10227 of the file referenced by *source_file*) to the destination path named by the concatenation of *target*,
 10228 a slash character, and the last component of *source_file*.

10229 The third and fourth synopsis forms are denoted by two or more operands where the **-R** or **-r**
 10230 options are specified. The *cp* utility shall copy each file in the file hierarchy rooted in each
 10231 *source_file* to a destination path named as follows.

10232 If *target* exists and is a file of type directory, the name of the corresponding destination path for
 10233 each file in the file hierarchy shall be the concatenation of *target*, a slash character, and the path
 10234 name of the file relative to the directory containing *source_file*.

10235 If *target* does not exist and two operands are specified, the name of the corresponding
 10236 destination path for *source_file* shall be *target*; the name of the corresponding destination path for
 10237 all other files in the file hierarchy shall be the concatenation of *target*, a slash character, and the
 10238 path name of the file relative to *source_file*.

10239 It shall be an error if *target* does not exist and more than two operands are specified, or if *target*
 10240 exists and is a file of a type defined by the System Interfaces volume of IEEE Std. 1003.1-200x,
 10241 but is not a file of type directory.

10242 In the following description, the term *dest_file* refers to the file named by the destination path.
 10243 The term *source_file* refers to the file that is being copied, whether specified as an operand or a
 10244 file in a file hierarchy rooted in a *source_file* operand. If *source_file* is a file of type symbolic link:

- 10245 • If neither the **-R** nor **-r** options were specified, *cp* shall take actions based on the type and
 10246 contents of the file referenced by the symbolic link, and not by the symbolic link itself.
- 10247 • If the **-R** option was specified:
 - 10248 — If none of the options **-H**, **-L**, nor **-P** were specified, it is unspecified which of **-H**, **-L**, or
 10249 **-P** will be used as a default.
 - 10250 — If the **-H** option was specified, *cp* shall take actions based on the type and contents of the
 10251 file referenced by any symbolic link specified as a *source_file* operand.
 - 10252 — If the **-L** option was specified, *cp* shall take actions based on the type and contents of the
 10253 file referenced by any symbolic link specified as a *source_file* operand or any symbolic

- 10254 links encountered during traversal of a file hierarchy.
- 10255 — If the **-P** option was specified, *cp* shall copy any symbolic link specified as a *source_file*
 10256 operand and any symbolic links encountered during traversal of a file hierarchy, and shall
 10257 not follow any symbolic links.
- 10258 • If the **-r** option was specified, the behavior is implementation-defined.
- 10259 For each *source_file*, the following steps shall be taken:
- 10260 1. If *source_file* references the same file as *dest_file*, *cp* may write a diagnostic message to
 10261 standard error; it shall do nothing more with *source_file* and shall go on to any remaining
 10262 files.
- 10263 2. If *source_file* is of type directory, the following steps shall be taken:
- 10264 a. If neither the **-R** or **-r** options were specified, *cp* shall write a diagnostic message to
 10265 standard error, do nothing more with *source_file*, and go on to any remaining files.
- 10266 b. If *source_file* was not specified as an operand and *source_file* is dot or dot-dot, *cp* shall
 10267 do nothing more with *source_file* and go on to any remaining files.
- 10268 c. If *dest_file* exists and it is a file type not specified by the System Interfaces volume of
 10269 IEEE Std. 1003.1-200x, the behavior is implementation-defined.
- 10270 d. If *dest_file* exists and it is not of type directory, *cp* shall write a diagnostic message to
 10271 standard error, do nothing more with *source_file* or any files below *source_file* in the
 10272 file hierarchy, and go on to any remaining files.
- 10273 e. If the directory *dest_file* does not exist, it shall be created with file permission bits set
 10274 to the same value as those of *source_file*, modified by the file creation mask of the
 10275 user if the **-p** option was not specified, and then bitwise-inclusively OR'ed with
 10276 S_IRWXU. If *dest_file* cannot be created, *cp* shall write a diagnostic message to
 10277 standard error, do nothing more with *source_file*, and go on to any remaining files. It
 10278 is unspecified if *cp* attempts to copy files in the file hierarchy rooted in *source_file*.
- 10279 f. The files in the directory *source_file* shall be copied to the directory *dest_file*, taking
 10280 the four steps [1-4] listed here with the files as *source_files*.
- 10281 g. If *dest_file* was created, its file permission bits shall be changed (if necessary) to be the
 10282 same as those of *source_file*, modified by the file creation mask of the user if the **-p**
 10283 option was not specified.
- 10284 h. The *cp* utility shall do nothing more with *source_file* and go on to any remaining files.
- 10285 3. If *source_file* is of type regular file, the following steps shall be taken:
- 10286 a. If *dest_file* exists, the following steps shall be taken:
- 10287 i. If the **-i** option is in effect, the *cp* utility shall write a prompt to the standard
 10288 error and read a line from the standard input. If the response is not affirmative,
 10289 *cp* shall do nothing more with *source_file* and go on to any remaining files.
- 10290 ii. A file descriptor for *dest_file* shall be obtained by performing actions equivalent
 10291 to the *open()* function defined in the System Interfaces volume of
 10292 IEEE Std. 1003.1-200x called using *dest_file* as the *path* argument, and the
 10293 bitwise-inclusive OR of O_WRONLY and O_TRUNC as the *oflag* argument.
- 10294 iii. If the attempt to obtain a file descriptor fails and the **-f** option is in effect, *cp*
 10295 shall attempt to remove the file by performing actions equivalent to the
 10296 *unlink()* function defined in the System Interfaces volume of

- 10297 IEEE Std. 1003.1-200x called using *dest_file* as the *path* argument. If this attempt
10298 succeeds, *cp* shall continue with step 3b.
- 10299 b. If *dest_file* does not exist, a file descriptor shall be obtained by performing actions
10300 equivalent to the *open()* function defined in the System Interfaces volume of
10301 IEEE Std. 1003.1-200x called using *dest_file* as the *path* argument, and the bitwise-
10302 inclusive OR of *O_WRONLY* and *O_CREAT* as the *oflag* argument. The file
10303 permission bits of *source_file* shall be the *mode* argument.
- 10304 c. If the attempt to obtain a file descriptor fails, *cp* shall write a diagnostic message to
10305 standard error, do nothing more with *source_file*, and go on to any remaining files.
- 10306 d. The contents of *source_file* shall be written to the file descriptor. Any write errors
10307 shall cause *cp* to write a diagnostic message to standard error and continue to step 3e.
- 10308 e. The file descriptor shall be closed.
- 10309 f. The *cp* utility shall do nothing more with *source_file*. If a write error occurred in step
10310 3d, it is unspecified if *cp* continues with any remaining files. If no write error
10311 occurred in step 3d, *cp* shall go on to any remaining files.
- 10312 4. Otherwise, the following steps shall be taken:
- 10313 a. If the **-r** option was specified, the behavior is implementation-defined.
- 10314 b. If the **-R** option was specified, the following steps shall be taken:
- 10315 i. The *dest_file* shall be created with the same file type as *source_file*.
- 10316 ii. If *source_file* is a file of type FIFO, the file permission bits shall be the same as
10317 those of *source_file*, modified by the file creation mask of the user if the **-p**
10318 option was not specified. Otherwise, the permissions, owner ID, and group ID
10319 of *dest_file* are implementation-defined.
- 10320 If this creation fails for any reason, *cp* shall write a diagnostic message to
10321 standard error, do nothing more with *source_file*, and go on to any remaining
10322 files.
- 10323 iii. If *source_file* is a file of type symbolic link, the path name contained in *dest_file*
10324 shall be the same as the path name contained in *source_file*.
- 10325 If this fails for any reason, *cp* shall write a diagnostic message to standard error,
10326 do nothing more with *source_file*, and go on to any remaining files.
- 10327 If the implementation provides additional or alternate access control mechanisms (see the Base
10328 Definitions volume of IEEE Std. 1003.1-200x, Section 4.1, File Access Permissions), their effect on
10329 copies of files is implementation-defined.

10330 OPTIONS

- 10331 The *cp* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
10332 Utility Syntax Guidelines.
- 10333 The following options shall be supported:
- 10334 **-f** If a file descriptor for a destination file cannot be obtained, as described in step
10335 3.a.ii., attempt to unlink the destination file and proceed.
- 10336 **-H** Take actions based on the type and contents of the file referenced by any symbolic
10337 link specified as a *source_file* operand.
- 10338 **-i** Write a prompt to standard error before copying to any existing destination file. If
10339 the response from the standard input is affirmative, the copy shall be attempted;

10340 otherwise, it shall not.

10341 **-L** Take actions based on the type and contents of the file referenced by any symbolic
10342 link specified as a *source_file* operand or any symbolic links encountered during
10343 traversal of a file hierarchy.

10344 **Notes to Reviewers**

10345 *This section with side shading will not appear in the final copy. - Ed.*

10346 A description of the **-P** option is needed. This is not in the IEEE P1003.2b draft
10347 standard.

10348 **-p** Duplicate the following characteristics of each source file in the corresponding
10349 destination file:

10350 1. The time of last data modification and time of last access. If this duplication
10351 fails for any reason, *cp* shall write a diagnostic message to standard error.

10352 2. The user ID and group ID. If this duplication fails for any reason, it is
10353 unspecified whether *cp* writes a diagnostic message to standard error.

10354 3. The file permission bits and the S_ISUID and S_ISGID bits. Other,
10355 implementation-defined, bits may be duplicated as well. If this duplication
10356 fails for any reason, *cp* shall write a diagnostic message to standard error.

10357 If the user ID or the group ID cannot be duplicated, the file permission bits
10358 S_ISUID and S_ISGID shall be cleared. If these bits are present in the source file but
10359 are not duplicated in the destination file, it is unspecified whether *cp* writes a
10360 diagnostic message to standard error.

10361 The order in which the preceding characteristics are duplicated is unspecified. The
10362 *dest_file* shall not be deleted if these characteristics cannot be preserved.

10363 **-R** Copy file hierarchies.

10364 OB **-r** Copy file hierarchies. The treatment of special files is implementation-defined.

10365 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be
10366 considered an error. The last option specified shall determine the behavior of the utility.

10367 **OPERANDS**

10368 The following operands shall be supported:

10369 *source_file* A path name of a file to be copied.

10370 *target_file* A path name of an existing or nonexistent file, used for the output when a single
10371 file is copied.

10372 *target* A path name of a directory to contain the copied files.

10373 **STDIN**

10374 Used to read an input line in response to each prompt specified in the STDERR section.
10375 Otherwise, the standard input shall not be used.

10376 **INPUT FILES**

10377 The input files specified as operands may be of any file type.

10378 **ENVIRONMENT VARIABLES**

10379 The following environment variables shall affect the execution of *cp*:

10380 *LANG* Provide a default value for the internationalization variables that are unset or null.
 10381 If *LANG* is unset or null, the corresponding value from the implementation-
 10382 defined default locale shall be used. If any of the internationalization variables
 10383 contains an invalid setting, the utility shall behave as if none of the variables had
 10384 been defined.

10385 *LC_ALL* If set to a non-empty string value, override the values of all the other
 10386 internationalization variables.

10387 *LC_COLLATE*
 10388 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 10389 character collating elements used in the extended regular expression defined for
 10390 the **yesexpr** locale keyword in the *LC_MESSAGES* category.

10391 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 10392 characters (for example, single-byte as opposed to multi-byte characters in
 10393 arguments and input files) and the behavior of character classes used in the
 10394 extended regular expression defined for the **yesexpr** locale keyword in the
 10395 *LC_MESSAGES* category.

10396 *LC_MESSAGES*
 10397 Determine the locale for the processing of affirmative responses that should be
 10398 used to affect the format and contents of diagnostic messages written to standard
 10399 error.

10400 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

10401 **ASYNCHRONOUS EVENTS**

10402 Default.

10403 **STDOUT**

10404 Not used.

10405 **STDERR**

10406 A prompt shall be written to standard error under the conditions specified in the *DESCRIPTION*
 10407 section. The prompt shall contain the destination path name, but its format is otherwise
 10408 unspecified. Otherwise, the standard error shall be used only for diagnostic messages.

10409 **OUTPUT FILES**

10410 The output files may be of any type.

10411 **EXTENDED DESCRIPTION**

10412 None.

10413 **EXIT STATUS**

10414 The following exit values shall be returned:

10415 0 All files were copied successfully.

10416 >0 An error occurred.

10417 **CONSEQUENCES OF ERRORS**

10418 If *cp* is prematurely terminated by a signal or error, files or file hierarchies may be only partially
 10419 copied and files and directories may have incorrect permissions or access and modification
 10420 times.

10421 **APPLICATION USAGE**

10422 The difference between **-R** and **-r** is in the treatment by *cp* of file types other than regular and
10423 directory. The original **-r** flag, for historic reasons, does not handle special files any differently
10424 from regular files, but always reads the file and copies its contents. This has obvious problems in
10425 the presence of special file types; for example, character devices, FIFOs, and sockets. The **-R**
10426 option is intended to recreate the file hierarchy and the **-r** option supports historical practice. It
10427 was anticipated that a future version of this volume of IEEE Std. 1003.1-200x would deprecate
10428 the **-r** option, and for that reason, there has been no attempt to fix its behavior with respect to
10429 FIFOs or other file types where copying the file is clearly wrong. However, some systems
10430 support **-r** with the same abilities as the **-R** defined in this volume of IEEE Std. 1003.1-200x. To
10431 accommodate them as well as systems that do not, the differences between **-r** and **-R** are
10432 implementation-defined. Implementations may make them identical. The **-r** option is now
10433 marked obsolescent.

10434 The set-user-ID and set-group-ID bits are explicitly cleared when files are created. This is to
10435 prevent users from creating programs that are set-user-ID or set-group-ID to them when
10436 copying files or to make set-user-ID or set-group-ID files accessible to new groups of users. For
10437 example, if a file is set-user-ID and the copy has a different group ID than the source, a new
10438 group of users has execute permission to a set-user-ID program than did previously. In
10439 particular, this is a problem for superusers copying users' trees.

10440 **EXAMPLES**

10441 None.

10442 **RATIONALE**

10443 The **-i** option exists on BSD systems, giving applications and users a way to avoid accidentally
10444 removing files when copying. Although the 4.3 BSD version does not prompt if the standard
10445 input is not a terminal, the standard developers decided that use of **-i** is a request for interaction,
10446 so when the destination path exists, the utility takes instructions from whatever responds on
10447 standard input.

10448 The exact format of the interactive prompts is unspecified. Only the general nature of the
10449 contents of prompts are specified because implementations may desire more descriptive
10450 prompts than those used on historical implementations. Therefore, an application using the **-i**
10451 option relies on the system to provide the most suitable dialog directly with the user, based on
10452 the behavior specified.

10453 The **-p** option is historical practice on BSD systems, duplicating the time of last data
10454 modification and time of last access. This volume of IEEE Std. 1003.1-200x extends it to preserve
10455 the user and group IDs, as well as the file permissions. This requirement has obvious problems
10456 in that the directories are almost certainly modified after being copied. This volume of
10457 IEEE Std. 1003.1-200x requires that the modification times be preserved. The statement that the
10458 order in which the characteristics are duplicated is unspecified is to permit implementations to
10459 provide the maximum amount of security for the user. Implementations should take into
10460 account the obvious security issues involved in setting the owner, group, and mode in the
10461 wrong order or creating files with an owner, group, or mode different from the final value.

10462 It is unspecified whether *cp* writes diagnostic messages when the user and group IDs cannot be
10463 set due to the widespread practice of users using **-p** to duplicate some portion of the file
10464 characteristics, indifferent to the duplication of others. Historic implementations only write
10465 diagnostic messages on errors other than [EPERM].

10466 The **-r** option is historical practice on BSD and BSD-derived systems, copying file hierarchies as
10467 opposed to single files. This functionality is used heavily in historical applications, and its loss
10468 would significantly decrease consensus. The **-R** option was added as a close synonym to the **-r**
10469 option, selected for consistency with all other options in this volume of IEEE Std. 1003.1-200x

10470 that do recursive directory descent.

10471 When a failure occurs during the copying of a file hierarchy, *cp* is required to attempt to copy
10472 files that are on the same level in the hierarchy or above the file where the failure occurred. It is
10473 unspecified if *cp* shall attempt to copy files below the file where the failure occurred (which
10474 cannot succeed in any case).

10475 Permissions, owners, and groups of created special file types have been deliberately left as
10476 implementation-defined. This is to allow systems to satisfy special requirements (for example,
10477 allowing users to create character special devices, but requiring them to be owned by a certain
10478 group). In general, it is strongly suggested that the permissions, owner, and group be the same
10479 as if the user had run the historical *mknod*, *ln*, or other utility to create the file. It is also probable
10480 that additional privileges are required to create block, character, or other implementation-
10481 defined special file types.

10482 Additionally, the *-p* option explicitly requires that all set-user-ID and set-group-ID permissions
10483 be discarded if any of the owner or group IDs cannot be set. This is to keep users from
10484 unintentionally giving away special privilege when copying programs.

10485 When creating regular files, historical versions of *cp* use the mode of the source file as modified
10486 by the file mode creation mask. Other choices would have been to use the mode of the source file
10487 unmodified by the creation mask or to use the same mode as would be given to a new file
10488 created by the user (plus the execution bits of the source file) and then modify it by the file mode
10489 creation mask. In the absence of any strong reason to change historic practice, it was in large part
10490 retained.

10491 When creating directories, historical versions of *cp* use the mode of the source directory, plus
10492 read, write, and search bits for the owner, as modified by the file mode creation mask. This is
10493 done so that *cp* can copy trees where the user has read permission, but the owner does not. A
10494 side effect is that if the file creation mask denies the owner permissions, *cp* fails. Also, once the
10495 copy is done, historical versions of *cp* set the permissions on the created directory to be the same
10496 as the source directory, unmodified by the file creation mask.

10497 This behavior has been modified so that *cp* is always able to create the contents of the directory,
10498 regardless of the file creation mask. After the copy is done, the permissions are set to be the same
10499 as the source directory, as modified by the file creation mask. This latter change from historical
10500 behavior is to prevent users from accidentally creating directories with permissions beyond
10501 those they would normally set and for consistency with the behavior of *cp* in creating files.

10502 It is not a requirement that *cp* detect attempts to copy a file to itself; however, implementations
10503 are strongly encouraged to do so. Historical implementations have detected the attempt in most
10504 cases.

10505 There are two methods of copying subtrees in this volume of IEEE Std. 1003.1-200x. The other
10506 method is described as part of the *pax* utility (see *pax* (on page 2910)). Both methods are
10507 historical practice. The *cp* utility provides a simpler, more intuitive interface, while *pax* offers a
10508 finer granularity of control. Each provides additional functionality to the other; in particular, *pax*
10509 maintains the hard-link structure of the hierarchy, while *cp* does not. It is the intention of the
10510 standard developers that the results be similar (using appropriate option combinations in both
10511 utilities). The results are not required to be identical; there seemed insufficient gain to
10512 applications to balance the difficulty of implementations having to guarantee that the results
10513 would be exactly identical.

10514 The wording allowing *cp* to copy a directory to implementation-defined file types not specified
10515 by the System Interfaces volume of IEEE Std. 1003.1-200x is provided so that implementations
10516 supporting symbolic links are not required to prohibit copying directories to symbolic links.
10517 Other extensions to the System Interfaces volume of IEEE Std. 1003.1-200x file types may need to

10518 use this loophole as well.

10519 **FUTURE DIRECTIONS**

10520 The `-r` option may be removed; use `-R` instead.

10521 **SEE ALSO**

10522 *mv, find, ln, pax*

10523 **CHANGE HISTORY**

10524 First released in Issue 2.

10525 **Issue 4**

10526 Aligned with the ISO/IEC 9945-2:1993 standard.

10527 **Issue 6**

10528 The `-r` option is marked obsolescent.

10529 The new options `-H`, `-L`, and `-P` are added to align with the IEEE P1003.2b draft standard. These

10530 options affect the processing of symbolic links.

10531 NAME

10532 crontab — schedule periodic background work

10533 SYNOPSIS

10534 UP crontab [*file*]

10535 crontab [-e | -l | -r]

10536

10537 DESCRIPTION

10538 The *crontab* utility shall create, replace, or edit a user's *crontab* entry; a *crontab* entry is a list of
 10539 commands and the times at which they shall be executed. The new *crontab* entry can be input by
 10540 specifying *file* or input from standard input if no *file* operand is specified, or by using an editor, if
 10541 **-e** is specified.

10542 Upon execution of a command from a *crontab* entry, the implementation shall supply a default
 10543 environment, defining at least the following environment variables:

10544 **HOME** A path name of the user's home directory.

10545 **LOGNAME** The user's login name.

10546 **PATH** A string representing a search path guaranteed to find all of the standard utilities.

10547 **SHELL** A path name of the command interpreter. When *crontab* is invoked as specified by
 10548 this volume of IEEE Std. 1003.1-200x, the value shall be a path name for *sh*.

10549 The values of these variables when *crontab* is invoked as specified by this volume of
 10550 IEEE Std. 1003.1-200x shall not affect the default values provided when the scheduled command
 10551 is run.

10552 If standard output and standard error are not redirected by commands executed from the
 10553 *crontab* entry, any generated output or errors shall be mailed, via an implementation-defined
 10554 method, to the user.

10555 XSI Users are permitted to use *crontab* if their names appear in the file **/usr/lib/cron/cron.allow**. If
 10556 that file does not exist, the file **/usr/lib/cron/cron.deny** is checked to determine whether the user
 10557 should be denied access to *crontab*. If neither file exists, only a process with appropriate
 10558 privileges is allowed to submit a job. If only **cron.deny** exists and is empty, global usage is
 10559 permitted. The **cron.allow** and **cron.deny** files consist of one user name per line.

10560 OPTIONS

10561 The *crontab* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 10562 12.2, Utility Syntax Guidelines.

10563 The following options shall be supported:

10564 **-e** Edit a copy of the invoking user's *crontab* entry, or create an empty entry to edit if
 10565 the *crontab* entry does not exist. When editing is complete, the entry shall be
 10566 installed as the user's *crontab* entry.

10567 **-l** (The letter ell.) List the invoking user's *crontab* entry.

10568 **-r** Remove the invoking user's *crontab* entry.

10569 OPERANDS

10570 The following operand shall be supported:

10571 **file** The path name of a file that contains specifications, in the format defined in the
 10572 INPUT FILES section, for *crontab* entries.

10573 **STDIN**

10574 See the INPUT FILES section.

10575 **INPUT FILES**

10576 In the POSIX locale, the user or application shall ensure that a crontab entry is a text file
 10577 consisting of lines of six fields each. The fields shall be separated by <blank> characters. The first
 10578 five fields shall be integer patterns that specify the following:

- 10579 1. Minute (0-59)
- 10580 2. Hour (0-23)
- 10581 3. Day of the month (1-31)
- 10582 4. Month of the year (1-12)
- 10583 5. Day of the week (0-6 with 0=Sunday)

10584 Each of these patterns can be either an asterisk (meaning all valid values), an element, or a list of
 10585 elements separated by commas. An element shall be either a number or two numbers separated
 10586 by a hyphen (meaning an inclusive range). The specification of days can be made by two fields
 10587 (day of the month and day of the week). If month, day of month, and day of week are all
 10588 asterisks, every day shall be matched. If either the month or day of month is specified as an
 10589 element or list, but the day of week is an asterisk, the month and day of month fields shall
 10590 specify the days that match. If both month and day of month are specified as asterisk, but day of
 10591 week is an element or list, then only the specified days of the week match. Finally, if either the
 10592 month or day of month is specified as an element or list, and the day of week is also specified as
 10593 an element or list, then any day matching either the month and day of month, or the day of
 10594 week, shall be matched.

10595 The sixth field of a line in a crontab entry is a string that shall be executed by *sh* at the specified
 10596 times. A percent sign character in this field shall be translated to a <newline> character. Any
 10597 character preceded by a backslash (including the '%') shall cause that character to be treated
 10598 literally. Only the first line (up to a '%' or end-of-line) of the command field shall be executed
 10599 by the command interpreter. The other lines shall be made available to the command as
 10600 standard input.

10601 Blank lines and those whose first non-<blank> character is '#' shall be ignored.

10602 XSI The text files `/usr/lib/cron/cron.allow` and `/usr/lib/cron/cron.deny` contain user names, one per
 10603 line, of users who are, respectively, authorized or denied access to the service underlying the
 10604 *crontab* utility.

10605 **ENVIRONMENT VARIABLES**

10606 The following environment variables shall affect the execution of *crontab*:

- | | | |
|---|-----------------|--|
| 10607
10608 | <i>EDITOR</i> | Determine the editor to be invoked when the <code>-e</code> option is specified. The default editor shall be <i>vi</i> . |
| 10609
10610
10611
10612
10613 | <i>LANG</i> | Provide a default value for the internationalization variables that are unset or null. If <i>LANG</i> is unset or null, the corresponding value from the implementation-defined default locale shall be used. If any of the internationalization variables contains an invalid setting, the utility shall behave as if none of the variables had been defined. |
| 10614
10615 | <i>LC_ALL</i> | If set to a non-empty string value, override the values of all the other internationalization variables. |
| 10616
10617 | <i>LC_CTYPE</i> | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in |

- 10618 arguments and input files).
- 10619 **LC_MESSAGES**
- 10620 Determine the locale that should be used to affect the format and contents of
- 10621 diagnostic messages written to standard error.
- 10622 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 10623 **ASYNCHRONOUS EVENTS**
- 10624 Default.
- 10625 **STDOUT**
- 10626 If the `-l` option is specified, the crontab entry shall be written to the standard output.
- 10627 **STDERR**
- 10628 Used only for diagnostic messages.
- 10629 **OUTPUT FILES**
- 10630 None.
- 10631 **EXTENDED DESCRIPTION**
- 10632 None.
- 10633 **EXIT STATUS**
- 10634 The following exit values shall be returned:
- 10635 0 Successful completion.
- 10636 >0 An error occurred.
- 10637 **CONSEQUENCES OF ERRORS**
- 10638 The user's crontab entry is not submitted, removed, edited, or listed.
- 10639 **APPLICATION USAGE**
- 10640 The format of the *crontab* entry shown here is guaranteed only for the POSIX locale. Other
- 10641 cultures may be supported with substantially different interfaces, although implementations are
- 10642 encouraged to provide comparable levels of functionality.
- 10643 The default settings of the *HOME*, *LOGNAME*, *PATH*, and *SHELL* variables that are given to the
- 10644 scheduled job are not affected by the settings of those variables when *crontab* is run; as stated,
- 10645 they are defaults. The text about "invoked as specified by this volume of IEEE Std. 1003.1-200x"
- 10646 means that the implementation may provide extensions that allow these variables to be affected
- 10647 at runtime, but that the user has to take explicit action in order to access the extension, such as
- 10648 give a new option flag or modify the format of the crontab entry.
- 10649 A typical user error is to type only *crontab*; this causes the system to wait for the new crontab
- 10650 entry on standard input. If end-of-file is typed (generally `<control>-D`), the crontab entry is
- 10651 replaced by an empty file. In this case, the user should type the interrupt character, which
- 10652 prevents the crontab entry from being replaced.
- 10653 **EXAMPLES**
- 10654 1. Clean up **core** files every weekday morning at 3:15 am:
- 10655 15 3 * * 1-5 find \$HOME -name core 2>/dev/null | xargs rm -f
- 10656 2. Mail a birthday greeting:
- 10657 0 12 14 2 * mailx john%Happy Birthday!%Time for lunch.
- 10658 3. As an example of specifying the two types of days:

10659 0 0 1,15 * 1

10660 would run a command on the first and fifteenth of each month, as well as on every
10661 Monday. To specify days by only one field, the other field should be set to '*'; for
10662 example:

10663 0 0 * * 1

10664 would run a command only on Mondays.

10665 **RATIONALE**

10666 All references to a *cron* daemon and to *cron files* have been omitted. Although historical
10667 implementations have used this arrangement, there is no reason to limit future implementations.

10668 This description of *crontab* is designed to support only users with normal privileges. The format
10669 of the input is based on the System V *crontab*; however, there is no requirement here that the
10670 actual system database used by the *cron* daemon (or a similar mechanism) use this format
10671 internally. For example, systems derived from BSD are likely to have an additional field
10672 appended that indicates the user identity to be used when the job is submitted.

10673 The `-e` option was adopted from the SVID as a user convenience, although it does not exist in all
10674 historical implementations.

10675 **FUTURE DIRECTIONS**

10676 None.

10677 **SEE ALSO**

10678 *at*

10679 **CHANGE HISTORY**

10680 First released in Issue 2.

10681 **Issue 4**

10682 Aligned with the ISO/IEC 9945-2:1993 standard.

10683 **Issue 6**

10684 This utility is now marked as part of the User Portability Utilities option.

10685 The normative text is reworded to avoid use of the term “must” for application requirements.

10686 NAME

10687 csplit — split files based on context

10688 SYNOPSIS

10689 UP csplit [-ks][-f *prefix*][-n *number*] *file arg1 ... argn*

10690

10691 DESCRIPTION

10692 The *csplit* utility shall read the file named by the *file* operand, write all or part of that file into
10693 other files as directed by the *arg* operands, and write the sizes of the files.

10694 OPTIONS

10695 The *csplit* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
10696 12.2, Utility Syntax Guidelines.

10697 The following options shall be supported:

10698 **-f *prefix*** Name the created files *prefix00*, *prefix01*, ..., *prefixn*. The default is **xx00** ... **xxn**. If
10699 the *prefix* argument would create a file name exceeding {NAME_MAX} bytes, an
10700 error shall result, *csplit* shall exit with a diagnostic message and no files shall be
10701 created.10702 **-k** Leave previously created files intact. By default, *csplit* shall remove created files if
10703 an error occurs.10704 **-n *number*** Use *number* decimal digits to form file names for the file pieces. The default shall be
10705 2.10706 **-s** Suppress the output of file size messages.

10707 OPERANDS

10708 The following operands shall be supported:

10709 ***file*** The path name of a text file to be split. If *file* is '-', the standard input shall be
10710 used.10711 The operands *arg1* ... *argn* can be a combination of the following:10712 ***/rexp/[offset]***10713 A file shall be created using the content of the lines from the current line up to, but
10714 not including, the line that results from the evaluation of the regular expression
10715 with *offset*, if any, applied. The regular expression *rexp* shall follow the rules for
10716 basic regular expressions described in the Base Definitions volume of
10717 IEEE Std. 1003.1-200x, Section 9.3, Basic Regular Expressions. The application shall
10718 use the sequence "\/" to specify a slash character within the *rexp*. The optional
10719 offset shall be a positive or negative integer value representing a number of lines.
10720 A positive integer value can be preceded by '+'. If the selection of lines from an
10721 *offset* expression of this type would create a file with zero lines, or one with greater
10722 than the number of lines left in the input file, the results are unspecified. After the
10723 section is created, the current line shall be set to the line that results from the
10724 evaluation of the regular expression with any offset applied. If the current line is
10725 the first line in the file and a regular expression operation has not yet been
10726 performed, the pattern match of *rexp* shall be applied from the current line to the
10727 end of the file. Otherwise, the pattern match of *rexp* shall be applied from the line
10728 following the current line to the end of the file.10729 ***%rexp%[offset]***10730 Equivalent to */rexp/[offset]*, except that no file shall be created for the selected
10731 section of the input file. The application shall use the sequence "\%" to specify a

- 10732 percent-sign character within the *rexp*.
- 10733 *line_no* Create a file from the current line up to (but not including) the line number *line_no*.
 10734 Lines in the file shall be numbered starting at one. The current line becomes
 10735 *line_no*.
- 10736 *{num}* Repeat operand. This operand can follow any of the operands described
 10737 previously. If it follows a *rexp* type operand, that operand shall be applied *num*
 10738 more times. If it follows a *line_no* operand, the file shall be split every *line_no* lines,
 10739 *num* times, from that point.
- 10740 An error shall be reported if an operand does not reference a line between the current position
 10741 and the end of the file.
- 10742 **STDIN**
- 10743 See the INPUT FILES section.
- 10744 **INPUT FILES**
- 10745 The input file shall be a text file.
- 10746 **ENVIRONMENT VARIABLES**
- 10747 The following environment variables shall affect the execution of *csplit*:
- 10748 *LANG* Provide a default value for the internationalization variables that are unset or null.
 10749 If *LANG* is unset or null, the corresponding value from the implementation-
 10750 defined default locale shall be used. If any of the internationalization variables
 10751 contains an invalid setting, the utility shall behave as if none of the variables had
 10752 been defined.
- 10753 *LC_ALL* If set to a non-empty string value, override the values of all the other
 10754 internationalization variables.
- 10755 *LC_COLLATE*
 10756 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 10757 character collating elements within regular expressions.
- 10758 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 10759 characters (for example, single-byte as opposed to multi-byte characters in
 10760 arguments and input files) and the behavior of character classes within regular
 10761 expressions.
- 10762 *LC_MESSAGES*
 10763 Determine the locale that should be used to affect the format and contents of
 10764 diagnostic messages written to standard error.
- 10765 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 10766 **ASYNCHRONOUS EVENTS**
- 10767 If the *-k* option is specified, created files shall be retained. Otherwise, the default action occurs.
- 10768 **STDOUT**
- 10769 Unless the *-s* option is used, the standard output shall consist of one line per file created, with a
 10770 format as follows:
 10771 "%d\n", <file size in bytes>

10772 **STDERR**

10773 Used only for diagnostic messages.

10774 **OUTPUT FILES**

10775 The output files shall contain portions of the original input file; otherwise, unchanged.

10776 **EXTENDED DESCRIPTION**

10777 None.

10778 **EXIT STATUS**

10779 The following exit values shall be returned:

10780 0 Successful completion.

10781 >0 An error occurred.

10782 **CONSEQUENCES OF ERRORS**

10783 By default, created files shall be removed if an error occurs. When the **-k** option is specified,
10784 created files shall not be removed if an error occurs.

10785 **APPLICATION USAGE**

10786 None.

10787 **EXAMPLES**

10788 1. This example creates four files, **cobol00 ... cobol03**:

10789 `csplit -f cobol file '/procedure division/' /par5./ /par16./`

10790 After editing the split files, they can be recombined as follows:

10791 `cat cobol0[0-3] > file`

10792 Note that this example overwrites the original file.

10793 2. This example would split the file after the first 99 lines, and every 100 lines thereafter, up
10794 to 9999 lines; this is because lines in the file are numbered from 1 rather than zero, for
10795 historical reasons:

10796 `csplit -k file 100 {99}`

10797 3. Assuming that **prog.c** follows the C-language coding convention of ending routines with a
10798 ' } ' at the beginning of the line, this example creates a file containing each separate C
10799 routine (up to 21) in **prog.c**:

10800 `csplit -k prog.c '%main(%' '/^}/+1' {20}`

10801 **RATIONALE**

10802 The **-n** option was added to extend the range of file names that could be handled.

10803 Consideration was given to adding a **-a** flag to use the alphabetic file name generation used by
10804 the historical *split* utility, but the functionality added by the **-n** option was deemed to make
10805 alphabetic naming unnecessary.

10806 **FUTURE DIRECTIONS**

10807 None.

10808 **SEE ALSO**

10809 *sed*, *split*

10810 **CHANGE HISTORY**

10811 First released in Issue 2.

10812 **Issue 4**

10813 Aligned with the ISO/IEC 9945-2: 1993 standard.

10814 **Issue 5**

10815 FUTURE DIRECTIONS section added.

10816 **Issue 6**

10817 This utility is now marked as part of the User Portability Utilities option.

10818 The APPLICATION USAGE section is added.

10819 The description of regular expression operands is changed to align with the IEEE P1003.2b draft standard.

10820

10821 The normative text is reworded to avoid use of the term “must” for application requirements.

10822 NAME

10823 ctags — create a tags file (DEVELOPMENT, FORTRAN)

10824 SYNOPSIS

10825 UP `ctags [-a][-f tagsfile] pathname ...`

10826 `ctags -x pathname ...`

10827

10828 DESCRIPTION

10829 The *ctags* utility shall be provided on systems that support the User Portability Utilities option,
10830 the Software Development Utilities option, and either or both of the C-Language Development
10831 Utilities option and FORTRAN Development Utilities option. On other systems, it is optional.

10832 The *ctags* utility shall write a *tags* file or an index of objects from C-language or FORTRAN
10833 source files specified by the *pathname* operands. The tags file shall list the locators of language-
10834 specific objects within the source files. A locator consists of a name, path name, and either a
10835 search pattern or a line number that can be used in searching for the object definition. The
10836 objects that shall be recognized are specified in the EXTENDED DESCRIPTION section.

10837 OPTIONS

10838 The *ctags* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
10839 12.2, Utility Syntax Guidelines.

10840 The following options shall be supported:

10841 **-a** Append to tags file.

10842 **-f *tagsfile*** Write the object locator lists into *tagsfile* instead of the default file named **tags** in
10843 the current directory.

10844 **-x** Produce a list of object names, the line number, and file name in which each is
10845 defined, as well as the text of that line, and write this to the standard output. A
10846 **tags** file shall not be created when **-x** is specified.

10847 OPERANDS

10848 The following *pathname* operands are supported:

10849 ***file.c*** Files with basenames ending with the **.c** suffix shall be treated as C-language
10850 source code. Such files that are not valid input to *c99* produce unspecified results.

10851 ***file.h*** Files with basenames ending with the **.h** suffix shall be treated as C-language
10852 source code. Such files that are not valid input to *c99* produce unspecified results.

10853 ***file.f*** Files with basenames ending with the **.f** suffix shall be treated as FORTRAN-
10854 language source code. Such files that are not valid input to *fort77* produce
10855 unspecified results.

10856 The handling of other files is implementation-defined.

10857 STDIN

10858 See the INPUT FILES section.

10859 INPUT FILES

10860 The input files shall be text files containing source code in the language indicated by the operand
10861 file name suffixes.

10862 **ENVIRONMENT VARIABLES**

10863 The following environment variables shall affect the execution of *ctags*:

10864 *LANG* Provide a default value for the internationalization variables that are unset or null.
 10865 If *LANG* is unset or null, the corresponding value from the implementation-
 10866 defined default locale shall be used. If any of the internationalization variables
 10867 contains an invalid setting, the utility shall behave as if none of the variables had
 10868 been defined.

10869 *LC_ALL* If set to a non-empty string value, override the values of all the other
 10870 internationalization variables.

10871 *LC_COLLATE*
 10872 Determine the order in which output is sorted for the *-x* option. The POSIX locale
 10873 determines the order in which the tags file is written.

10874 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 10875 characters (for example, single-byte as opposed to multi-byte characters in
 10876 arguments and input files). When processing C-language source code, if the locale
 10877 is not compatible with the C locale described by the ISO C standard, the results are
 10878 unspecified.

10879 *LC_MESSAGES*
 10880 Determine the locale that should be used to affect the format and contents of
 10881 diagnostic messages written to standard error.

10882 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

10883 **ASYNCHRONOUS EVENTS**

10884 Default.

10885 **STDOUT**

10886 The list of object name information produced by the *-x* option shall be written to standard
 10887 output in the following format:

```
10888 "%s %d %s %s", <object-name>, <line-number>, <filename>,  
10889 <text>
```

10890 where *<text>* is the text of line *<line-number>* of file *<filename>*.

10891 **STDERR**

10892 Used only for diagnostic messages.

10893 **OUTPUT FILES**

10894 When the *-x* option is not specified, the format of the output file shall be:

```
10895 "%s\t%s\t\t/%s/\n", <identifier>, <filename>, <pattern>
```

10896 where *<pattern>* is a search pattern that could be used by an editor to find the defining instance
 10897 of *<identifier>* in *<filename>* (where *defining instance* is indicated by the declarations listed in the
 10898 EXTENDED DESCRIPTION).

10899 An optional circumflex (*'^'*) can be added as a prefix to *<pattern>*, and an optional dollar sign
 10900 can be appended to *<pattern>* to indicate that the pattern is anchored to the beginning (end) of a
 10901 line of text. Any slash or backslash characters in *<pattern>* shall be preceded by a backslash
 10902 character. The anchoring circumflex, dollar sign, and escaping backslash characters shall not be
 10903 considered part of the search pattern. All other characters in the search pattern shall be
 10904 considered literal characters.

10905 An alternative format is:

10906 "%s\t%s\t?%s?\n", <identifier>, <filename>, <pattern>

10907 which is identical to the first format except that slashes in <pattern> shall not be preceded by
10908 escaping backslash characters, and question mark characters in <pattern> shall be preceded by
10909 backslash characters.

10910 A second alternative format is:

10911 "%s\t%s\t%d\n", <identifier>, <filename>, <lineno>

10912 where <lineno> is a decimal line number that could be used by an editor to find <identifier> in
10913 <filename>.

10914 Neither alternative format shall be produced by *ctags* when it is used as described by
10915 IEEE Std. 1003.1-200x, but the standard utilities that process tags files shall be able to process
10916 those formats as well as the first format.

10917 In any of these formats, the file shall be sorted by identifier, based on the collation sequence in
10918 the POSIX locale.

10919 EXTENDED DESCRIPTION

10920 If the operand identifies C-language source, the *ctags* utility shall attempt to produce an output
10921 line for each of the following objects:

- 10922 • Function definitions
- 10923 • Type definitions
- 10924 • Macros with arguments

10925 It may also produce output for any of the following objects:

- 10926 • Function prototypes
- 10927 • Structures
- 10928 • Unions
- 10929 • Global variable definitions
- 10930 • Enumeration types
- 10931 • Macros without arguments
- 10932 • **#define** statements
- 10933 • **#line** statements

10934 Any **#if** and **#ifdef** statements shall produce no output. The tag **main** is treated specially in C
10935 programs. The tag formed shall be created by prefixing **M** to the name of the file, with the
10936 trailing **.c**, and leading path name components (if any) removed.

10937 On systems that do not support the C-Language Development Utilities option, *ctags* produces
10938 undefined results for C-language source code files.

10939 If the operand identifies FORTRAN source, the *ctags* utility shall produce an output line for each
10940 function definition. It may also produce output for any of the following objects:

- 10941 • Subroutine definitions
- 10942 • COMMON statements

- 10943 • PARAMETER statements
 - 10944 • DATA and BLOCK DATA statements
 - 10945 • Statement numbers
- 10946 On systems that do not support the FORTRAN Development Utilities option, *ctags* produces
 10947 unspecified results for FORTRAN source code files. It should write to standard error a message
 10948 identifying this condition and cause a non-zero exit status to be produced.
- 10949 It is implementation-defined what other objects (including duplicate identifiers) produce output.
- 10950 **EXIT STATUS**
- 10951 The following exit values shall be returned:
- 10952 0 Successful completion.
 - 10953 >0 An error occurred.
- 10954 **CONSEQUENCES OF ERRORS**
- 10955 Default.
- 10956 **APPLICATION USAGE**
- 10957 The output with *-x* is meant to be a simple index that can be written out as an off-line readable
 10958 function index. If the input files to *ctags* (such as *.c* files) were not created using the same locales
 10959 as those in effect when *ctags -x* is run, results might not be as expected.
- 10960 The description of C-language processing says “attempts to” because the C language can be
 10961 greatly confused, especially through the use of *#defines*, and this utility would be of no use if
 10962 the real C preprocessor were run to identify them. The output from *ctags* may be fooled and
 10963 incorrect for various constructs.
- 10964 **EXAMPLES**
- 10965 None.
- 10966 **RATIONALE**
- 10967 The option list was significantly reduced from that provided by historical implementations. The
 10968 *-F* option was omitted as redundant, since it is the default. The *-B* option was omitted as being
 10969 of very limited usefulness. The *-t* option was omitted since the recognition of typedefs is now
 10970 required for C source files. The *-u* option was omitted because the update function was judged
 10971 to be not only inefficient, but also rarely needed.
- 10972 An early proposal included a *-w* option to suppress warning diagnostics. Since the types of such
 10973 diagnostics could not be described, the option was omitted as being not useful.
- 10974 The text for *LC_CTYPE* about compatibility with the C locale acknowledges that the ISO C
 10975 standard imposes requirements on the locale used to process C source. This could easily be a
 10976 superset of that known as “the C locale” by way of implementation extensions, or one of a few
 10977 alternative locales for systems supporting different codesets. No statement is made for
 10978 FORTRAN because the ANSI X3.9-1978 standard (FORTRAN 77) does not (yet) define a similar
 10979 locale concept. However, a general rule in this volume of IEEE Std. 1003.1-200x is that any time
 10980 that locales do not match (preparing a file for one locale and processing it in another), the results
 10981 are suspect.
- 10982 The collation sequence of the tags file is not affected by *LC_COLLATE* because it is typically not
 10983 used by human readers, but only by programs such as *vi* to locate the tag within the source files.
 10984 Using the POSIX locale eliminates some of the problems of coordinating locales between the
 10985 *ctags* file creator and the *vi* file reader.

- 10986 Historically, the tags file has been used only by *ex* and *vi*. However, the format of the tags file
 10987 has been published to encourage other programs to use the tags in new ways. The format allows
 10988 either patterns or line numbers to find the identifiers because the historical *vi* recognizes either.
 10989 The *ctags* utility does not produce the format using line numbers because it is not useful
 10990 following any source file changes that add or delete lines. The documented search patterns
 10991 match historical practice. It should be noted that literal leading circumflex or trailing dollar-sign
 10992 characters in the search pattern will only behave correctly if anchored to the beginning of the
 10993 line or end of the line by an additional circumflex or dollar-sign character.
- 10994 Historical implementations also understand the objects used by the languages Pascal and
 10995 sometimes LISP, and they understand the C source output by *lex* and *yacc*. The *ctags* utility is
 10996 not required to accommodate these languages, although implementors are encouraged to do so.
- 10997 The following historical option was not specified, as *vgrind* is not included in this volume of
 10998 IEEE Std. 1003.1-200x:
- 10999 -v If the -v flag is given, an index of the form expected by *vgrind* is produced on the
 11000 standard output. This listing contains the function name, file name, and page
 11001 number (assuming 64-line pages). Since the output is sorted into lexicographic
 11002 order, it may be desired to run the output through *sort -f*. Sample use:
- 11003 ctags -v files | sort -f > index vgrind -x index
- 11004 The special treatment of the tag **main** makes the use of *ctags* practical in directories with more
 11005 than one program.
- 11006 **FUTURE DIRECTIONS**
- 11007 None.
- 11008 **SEE ALSO**
- 11009 *c99*, *fort77*, *vi*
- 11010 **CHANGE HISTORY**
- 11011 First released in Issue 4.
- 11012 **Issue 5**
- 11013 FUTURE DIRECTIONS section added.
- 11014 **Issue 6**
- 11015 This utility is now marked as part of the User Portability Utilities option.
- 11016 The OUTPUT FILES section is changed to align with the IEEE P1003.2b draft standard.
- 11017 The normative text is reworded to avoid use of the term “must” for application requirements.
- 11018 IEEE PASC Interpretation 1003.2 #168 is applied, changing “create” to “write” in the
 11019 DESCRIPTION.

11020 NAME

11021 cut — cut out selected fields of each line of a file

11022 SYNOPSIS

11023 cut -b *list* [-n] [*file* ...]

11024 cut -c *list* [*file* ...]

11025 cut -f *list* [-d *delim*][-s][*file* ...]

11026 DESCRIPTION

11027 The *cut* utility shall cut out bytes (-b option), characters (-c option) or character-delimited fields
 11028 (-f option) from each line in one or more files, concatenate them, and write them to standard
 11029 output.

11030 OPTIONS

11031 The *cut* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 11032 12.2, Utility Syntax Guidelines.

11033 The application shall ensure that the option-argument *list* (see options -b, -c, and -f below) is a
 11034 comma-separated list or <blank> character-separated list of positive numbers and ranges.
 11035 Ranges can be in three forms. The first is two positive numbers separated by a hyphen
 11036 (*low-high*), which represents all fields from the first number to the second number. The second is
 11037 a positive number preceded by a hyphen (-*high*), which represents all fields from field number 1
 11038 to that number. The third is a positive number followed by a hyphen (*low-*), which represents
 11039 that number to the last field, inclusive. The elements in *list* can be repeated, can overlap, and can
 11040 be specified in any order, but the bytes, characters, or fields selected shall be written in the order
 11041 of the input data. If an element appears in the selection list more than once, it shall be written
 11042 exactly once.

11043 The following options shall be supported:

11044 -b *list* Cut based on a *list* of bytes. Each selected byte shall be output unless the -n option
 11045 is also specified. It shall not be an error to select bytes not present in the input line.

11046 -c *list* Cut based on a *list* of characters. Each selected character shall be output. It shall
 11047 not be an error to select characters not present in the input line.

11048 -d *delim* Set the field delimiter to the character *delim*. The default is the <tab> character.

11049 -f *list* Cut based on a *list* of fields, assumed to be separated in the file by a delimiter
 11050 character (see -d). Each selected field shall be output. Output fields shall be
 11051 separated by a single occurrence of the field delimiter character. Lines with no field
 11052 delimiters shall be passed through intact, unless -s is specified. It shall not be an
 11053 error to select fields not present in the input line.

11054 -n Do not split characters. When specified with the -b option, each element in *list* of
 11055 the form *low-high* (hyphen-separated numbers) shall be modified as follows:

- 11056 • If the byte selected by *low* is not the first byte of a character, *low* shall be
 11057 decremented to select the first byte of the character originally selected by *low*.
 11058 If the byte selected by *high* is not the last byte of a character, *high* shall be
 11059 decremented to select the last byte of the character prior to the character
 11060 originally selected by *high*, or zero if there is no prior character. If the resulting
 11061 range element has *high* equal to zero or *low* greater than *high*, the list element
 11062 shall be dropped from *list* for that input line without causing an error.

11063 Each element in *list* of the form *low-* shall be treated as above with *high* set to the
 11064 number of bytes in the current line, not including the terminating <newline>

11065 character. Each element in *list* of the form *-high* shall be treated as above with *low*
 11066 set to 1. Each element in *list* of the form *num* (a single number) shall be treated as
 11067 above with *low* set to *num* and *high* set to *num*.

11068 **-s** Suppress lines with no delimiter characters, when used with the **-f** option. Unless
 11069 specified, lines with no delimiters shall be passed through untouched.

11070 OPERANDS

11071 The following operand shall be supported:

11072 **file** A path name of an input file. If no *file* operands are specified, or if a *file* operand is
 11073 '–', the standard input shall be used.

11074 STDIN

11075 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '–'.
 11076 See the INPUT FILES section.

11077 INPUT FILES

11078 The input files shall be text files, except that line lengths shall be unlimited.

11079 ENVIRONMENT VARIABLES

11080 The following environment variables shall affect the execution of *cut*:

11081 **LANG** Provide a default value for the internationalization variables that are unset or null.
 11082 If *LANG* is unset or null, the corresponding value from the implementation-
 11083 defined default locale shall be used. If any of the internationalization variables
 11084 contains an invalid setting, the utility shall behave as if none of the variables had
 11085 been defined.

11086 **LC_ALL** If set to a non-empty string value, override the values of all the other
 11087 internationalization variables.

11088 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 11089 characters (for example, single-byte as opposed to multi-byte characters in
 11090 arguments and input files).

11091 LC_MESSAGES

11092 Determine the locale that should be used to affect the format and contents of
 11093 diagnostic messages written to standard error.

11094 **XSI NLS_PATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

11095 ASYNCHRONOUS EVENTS

11096 Default.

11097 STDOUT

11098 The *cut* utility output shall be a concatenation of the selected bytes, characters, or fields (one of
 11099 the following):

11100 "%s\n", <concatenation of bytes>

11101 "%s\n", <concatenation of characters>

11102 "%s\n", <concatenation of fields and field delimiters>

11103 STDERR

11104 Used only for diagnostic messages.

11105 **OUTPUT FILES**

11106 None.

11107 **EXTENDED DESCRIPTION**

11108 None.

11109 **EXIT STATUS**

11110 The following exit values shall be returned:

11111 0 All input files were output successfully.

11112 >0 An error occurred.

11113 **CONSEQUENCES OF ERRORS**

11114 Default.

11115 **APPLICATION USAGE**

11116 Earlier versions of the *cut* utility worked in an environment where bytes and characters were
 11117 considered equivalent (modulo <backspace> and <tab> character processing in some
 11118 implementations). In the extended world of multi-byte characters, the new **-b** option has been
 11119 added. The **-n** option (used with **-b**) allows it to be used to act on bytes rounded to character
 11120 boundaries. The algorithm specified for **-n** guarantees that:

11121 `cut -b 1-500 -n file > file1`11122 `cut -b 501- -n file > file2`

11123 ends up with all the characters in **file** appearing exactly once in **file1** or **file2**. (There is,
 11124 however, a <newline> character in both **file1** and **file2** for each <newline> character in **file**.)

11125 **EXAMPLES**

11126 Examples of the option qualifier list:

11127 1,4,7 Select the first, fourth, and seventh bytes, characters, or fields and field delimiters.

11128 1-3,8 Equivalent to 1,2,3,8.

11129 -5,10 Equivalent to 1,2,3,4,5,10.

11130 3- Equivalent to third to last, inclusive.

11131 The *low-high* forms are not always equivalent when used with **-b** and **-n** and multi-byte
 11132 characters; see the description of **-n**.

11133 The following command:

11134 `cut -d : -f 1,6 /etc/passwd`

11135 reads the System V password file (user database) and produces lines of the form:

11136 `<user ID>:<home directory>`

11137 Most utilities in this volume of IEEE Std. 1003.1-200x work on text files. The *cut* utility can be
 11138 used to turn files with arbitrary line lengths into a set of text files containing the same data. The
 11139 *paste* utility can be used to create (or recreate) files with arbitrary line lengths. For example, if **file**
 11140 contains long lines:

11141 `cut -b 1-500 -n file > file1`11142 `cut -b 501- -n file > file2`

11143 creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline> character) and
 11144 **file2** that contains the remainder of the data from **file**. (Note that **file2** is not a text file if there
 11145 are lines in **file** that are longer than 500 + {LINE_MAX} bytes.) The original file can be recreated
 11146 from **file1** and **file2** using the command:

11147 `paste -d "\0" file1 file2 > file`

11148 RATIONALE

11149 Some historical implementations do not count <backspace> characters in determining character
11150 counts with the `-c` option. This may be useful for using *cut* for processing *nroff* output. It was
11151 deliberately decided not to have the `-c` option treat either <backspace> or <tab> characters in
11152 any special fashion. The *fold* utility does treat these characters specially.

11153 Unlike other utilities, some historical implementations of *cut* exit after not finding an input file,
11154 rather than continuing to process the remaining *file* operands. This behavior is prohibited by this
11155 volume of IEEE Std. 1003.1-200x, where only the exit status is affected by this problem.

11156 The behavior of *cut* when provided with either mutually-exclusive options or options that do
11157 not work logically together has been deliberately left unspecified in favor of global wording in
11158 Section 1.11 (on page 2224).

11159 The OPTIONS section was changed in response to P1003.2-N149. The change represents
11160 historical practice on all known systems. The original standard was ambiguous on the nature of
11161 the output.

11162 The *list* option-arguments are historically used to select the portions of the line to be written, but
11163 do not affect the order of the data. For example:

11164 `echo abcdefghi | cut -c6,2,4-7,1`

11165 yields "abdefg".

11166 A proposal to enhance *cut* with the following option:

11167 `-o` Preserve the selected field order. When this option is specified, each byte, character, or field
11168 (or ranges of such) shall be written in the order specified by the *list* option-argument, even if
11169 this requires multiple outputs of the same bytes, characters, or fields.

11170 was rejected because this type of enhancement is outside the scope of the IEEE P1003.2b draft
11171 standard.

11172 FUTURE DIRECTIONS

11173 None.

11174 SEE ALSO

11175 *grep*, *paste*, Section 2.5 (on page 2241)

11176 CHANGE HISTORY

11177 First released in Issue 2.

11178 Issue 4

11179 Aligned with the ISO/IEC 9945-2:1993 standard.

11180 Issue 6

11181 The OPTIONS section is changed to align with the IEEE P1003.2b draft standard.

11182 The normative text is reworded to avoid use of the term “must” for application requirements.

11183 **NAME**11184 `cxref` — generate a C-language program cross-reference table (**DEVELOPMENT**)11185 **SYNOPSIS**

```
11186 xSI cxref [-cs][-o file][-w num] [-D name[=def]]...[-I dir]...
11187 [-U name]... file ...
11188
```

11189 **DESCRIPTION**

11190 The `cxref` utility shall analyze a collection of C-language *files* and attempt to build a cross-
 11191 reference table. Information from **#define** lines is included in the symbol table. A sorted listing
 11192 shall be written to standard output of all symbols (auto, static, and global) in each *file* separately,
 11193 or with the `-c` option, in combination. Each symbol contains an asterisk before the declaring
 11194 reference.

11195 **OPTIONS**

11196 The `cxref` utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 11197 12.2, Utility Syntax Guidelines, except that the order of the `-D`, `-I`, and `-U` options (which are
 11198 identical to their interpretation by *c99*) is significant. The following options shall be supported:

- 11199 `-c` Write a combined cross-reference of all input files.
- 11200 `-w num` Format output no wider than *num* (decimal) columns. This option defaults to 80 if
 11201 *num* is not specified or is less than 51.
- 11202 `-o file` Direct output to named *file*.
- 11203 `-s` Operate silently; do not print input file names.

11204 **OPERANDS**

11205 The following operand shall be supported:

- 11206 *file* A path name of a C-language source file.

11207 **STDIN**

11208 Not used.

11209 **INPUT FILES**

11210 The input files are C-language source files.

11211 **ENVIRONMENT VARIABLES**11212 The following environment variables shall affect the execution of `cxref`:

- 11213 *LANG* Provide a default value for the internationalization variables that are unset or null.
 11214 If *LANG* is unset or null, the corresponding value from the implementation-
 11215 defined default locale shall be used. If any of the internationalization variables
 11216 contains an invalid setting, the utility shall behave as if none of the variables had
 11217 been defined.

- 11218 *LC_ALL* If set to a non-empty string value, override the values of all the other
 11219 internationalization variables.

11220 *LC_COLLATE*

11221 Determine the locale for the ordering of the output.

- 11222 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 11223 characters (for example, single-byte as opposed to multi-byte characters in
 11224 arguments and input files).

11225 *LC_MESSAGES*

11226 Determine the locale that should be used to affect the format and contents of

- 11227 diagnostic messages written to standard error.
- 11228 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 11229 **ASYNCHRONOUS EVENTS**
- 11230 Default.
- 11231 **STDOUT**
- 11232 The standard output shall be used for the cross-reference listing, unless the **-o** option is used to
11233 select a different output file.
- 11234 The format of standard output is unspecified, except that the following information shall be
11235 included:
- 11236 • If the **-c** option is not specified, each portion of the listing starts with the name of the input
11237 file on a separate line.
 - 11238 • The name line is followed by a sorted list of symbols, each with its associated location path
11239 name, the name of the function in which it appears (if it is not a function name itself), and
11240 line number references.
 - 11241 • Each line number may be preceded by an asterisk (**'*'**) flag, meaning that this is the
11242 declaring reference. Other single-character flags, with implementation-defined meanings,
11243 may be included.
- 11244 **STDERR**
- 11245 Used only for diagnostic messages.
- 11246 **OUTPUT FILES**
- 11247 The output file named by the **-o** option shall be used instead of standard output.
- 11248 **EXTENDED DESCRIPTION**
- 11249 None.
- 11250 **EXIT STATUS**
- 11251 The following exit values shall be returned:
- 11252 0 Successful completion.
- 11253 >0 An error occurred.
- 11254 **CONSEQUENCES OF ERRORS**
- 11255 Default.
- 11256 **APPLICATION USAGE**
- 11257 None.
- 11258 **EXAMPLES**
- 11259 None.
- 11260 **RATIONALE**
- 11261 None.
- 11262 **FUTURE DIRECTIONS**
- 11263 None.
- 11264 **SEE ALSO**
- 11265 *c99*

11266 **CHANGE HISTORY**

11267 First released in Issue 2.

11268 **Issue 4**

11269 Format reorganized.

11270 Utility Syntax Guidelines support mandated.

11271 Internationalized environment variable support mandated.

11272 **Issue 5**

11273 In the SYNOPSIS, [-U *dir*]ischangedto[-U *name*].

11274 **Issue 6**

11275 The APPLICATION USAGE section is added.

11276 **NAME**

11277 date — write the date and time

11278 **SYNOPSIS**

11279 date [-u] [+format]

11280 xSI date [-u] *mmddhhmm*[[*cc*]*yy*]

11281

11282 **DESCRIPTION**

11283 xSI The *date* utility shall write the date and time to standard output or attempt to set the system date
 11284 and time. By default, the current date and time shall be written. If an operand beginning with
 11285 '+' is specified, the output format of *date* shall be controlled by the field descriptors and other
 11286 text in the operand.

11287 **OPTIONS**

11288 The *date* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 11289 12.2, Utility Syntax Guidelines.

11290 The following option shall be supported:

11291 -u Perform operations as if the *TZ* environment variable was set to the string "UTC0",
 11292 or its equivalent historical value of "GMT0". Otherwise, *date* shall use the
 11293 timezone indicated by the *TZ* environment variable or the system default if that
 11294 variable is not set.

11295 **OPERANDS**

11296 The following operands shall be supported:

11297 +format When the format is specified, each field descriptor shall be replaced in the
 11298 standard output by its corresponding value. All other characters shall be copied to
 11299 the output without change. The output always shall be terminated with a
 11300 <newline> character.

11301 **Field Descriptors**

11302	%a	Locale's abbreviated weekday name.
11303	%A	Locale's full weekday name.
11304	%b	Locale's abbreviated month name.
11305	%B	Locale's full month name.
11306	%c	Locale's appropriate date and time representation.
11307	%C	Century (a year divided by 100 and truncated to an integer) as a decimal
11308		number [00-99].
11309	%d	Day of the month as a decimal number [01-31].
11310	%D	Date in the format <i>mm/dd/yy</i> .
11311	%e	Day of the month as a decimal number [1-31] in a two-digit field with
11312		leading space character fill.
11313	%h	A synonym for %b.
11314	%H	Hour (24-hour clock) as a decimal number [00-23].
11315	%I	Hour (12-hour clock) as a decimal number [01-12].

11316	%j	Day of the year as a decimal number [001-366].
11317	%m	Month as a decimal number [01-12].
11318	%M	Minute as a decimal number [00-59].
11319	%n	A <newline> character.
11320	%p	Locale's equivalent of either AM or PM.
11321	%r	12-hour clock time [01-12] using the AM/PM notation; in the POSIX locale, this is equivalent to %I:%M:%S% p.
11322		
11323	%S	Seconds as a decimal number [00-61].
11324	%t	A <tab> character.
11325	%T	24-hour clock time [00-23] in the format <i>HH:MM:SS</i> .
11326	%u	Weekday as a decimal number [1 (Monday)-7].
11327	%U	Week of the year (Sunday as the first day of the week) as a decimal number [00-53]. All days in a new year preceding the first Sunday shall be considered to be in week 0.
11328		
11329		
11330	%V	Week of the year (Monday as the first day of the week) as a decimal number [01-53]. If the week containing January 1 has four or more days in the new year, then it shall be considered week 1; otherwise, it shall be the last week of the previous year, and the next week shall be week 1.
11331		
11332		
11333		
11334	%w	Weekday as a decimal number [0 (Sunday)-6].
11335	%W	Week of the year (Monday as the first day of the week) as a decimal number [00-53]. All days in a new year preceding the first Monday shall be considered to be in week 0.
11336		
11337		
11338	%x	Locale's appropriate date representation.
11339	%X	Locale's appropriate time representation.
11340	%y	Year within century [00-99].
11341	%Y	Year with century as a decimal number.
11342	%Z	Timezone name, or no characters if no timezone is determinable.
11343	%%	A percent sign character.
11344		See the Base Definitions volume of IEEE Std. 1003.1-200x, Section 7.3.5, <i>LC_TIME</i>
11345		for the field descriptor values in the POSIX locale.

11346 **Modified Field Descriptors**

11347		Some field descriptors can be modified by the <i>E</i> and <i>O</i> modifier characters to indicate a different format or specification as specified in the <i>LC_TIME</i> locale description (see the Base Definitions volume of IEEE Std. 1003.1-200x, Section 7.3.5, <i>LC_TIME</i>). If the corresponding keyword (see era , era_year , era_d_fmt , and alt_digits in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 7.3.5, <i>LC_TIME</i>) is not specified or not supported for the current locale, the unmodified field descriptor value shall be used.
11348		
11349		
11350		
11351		
11352		
11353		
11354	%Ec	

11355	%EC	The name of the base year (period) in the locale's alternative representation.
11356		
11357	%Ex	Locale's alternative date representation.
11358	%EX	Locale's alternative time representation.
11359	%Ey	Offset from %EC (year only) in the locale's alternative representation.
11360	%EY	Full alternative year representation.
11361	%Od	Day of month using the locale's alternative numeric symbols.
11362	%Oe	Day of month using the locale's alternative numeric symbols.
11363	%OH	Hour (24-hour clock) using the locale's alternative numeric symbols.
11364	%OI	Hour (12-hour clock) using the locale's alternative numeric symbols.
11365	%Om	Month using the locale's alternative numeric symbols.
11366	%OM	Minutes using the locale's alternative numeric symbols.
11367	%OS	Seconds using the locale's alternative numeric symbols.
11368	%Ou	Weekday as a number in the locale's alternative representation (Monday = 1).
11369		
11370	%OU	Week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.
11371		
11372	%OV	Week number of the year (Monday as the first day of the week, rules corresponding to %V), using the locale's alternative numeric symbols.
11373		
11374	%Ow	Weekday as a number in the locale's alternative representation (Sunday = 0).
11375		
11376	%OW	Week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
11377		
11378	%Oy	Year (offset from %C) in alternative representation.

11379	XSI	mmddhhmm[[cc]yy]
11380		Attempt to set the system date and time from the value given in the operand. This is only possible if the user has appropriate privileges and the system permits the setting of the system date and time. The first <i>mm</i> is the month (number); <i>dd</i> is the day (number); <i>hh</i> is the hour (number, 24-hour system); the second <i>mm</i> is the minute (number); <i>cc</i> is the century and is the first two digits of the year (this is optional); <i>yy</i> is the last two digits of the year and is optional. If century is not specified, then values in the range [69-99] shall refer to years 1969 to 1999 inclusive, and values in the range [00-68] shall refer to years 2000 to 2068 inclusive.
11381		
11382		
11383		
11384		
11385		
11386		
11387		

11388	STDIN	
11389		Not used.

11390	INPUT FILES	
11391		None.

11392 ENVIRONMENT VARIABLES

11393		The following environment variables shall affect the execution of <i>date</i> :
11394	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null.
11395		If <i>LANG</i> is unset or null, the corresponding value from the implementation-

11396 defined default locale shall be used. If any of the internationalization variables
 11397 contains an invalid setting, the utility shall behave as if none of the variables had
 11398 been defined.

11399 **LC_ALL** If set to a non-empty string value, override the values of all the other
 11400 internationalization variables.

11401 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 11402 characters (for example, single-byte as opposed to multi-byte characters in
 11403 arguments).

11404 **LC_MESSAGES**
 11405 Determine the locale that should be used to affect the format and contents of
 11406 diagnostic messages written to standard error.

11407 **LC_TIME** Determine the format and contents of date and time strings written by *date*.

11408 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

11409 **TZ** Determine the timezone in which the time and date are written, unless the **-u**
 11410 option is specified. If the **TZ** variable is not set and the **-u** is not specified, an
 11411 unspecified system default timezone is used.

11412 **ASYNCHRONOUS EVENTS**

11413 Default.

11414 **STDOUT**

11415 When no formatting operand is specified, the output in the POSIX locale shall be equivalent to
 11416 specifying:
 11417 `date "+%a %b %e %H:%M:%S %Z %Y"`

11418 **STDERR**

11419 Used only for diagnostic messages.

11420 **OUTPUT FILES**

11421 None.

11422 **EXTENDED DESCRIPTION**

11423 None.

11424 **EXIT STATUS**

11425 The following exit values shall be returned:

11426 0 The date was written successfully.

11427 >0 An error occurred.

11428 **CONSEQUENCES OF ERRORS**

11429 Default.

11430 **APPLICATION USAGE**

11431 Field descriptors are of unspecified format when not in the POSIX locale. Some of them can
 11432 contain <newline> characters in some locales, so it may be difficult to use the format shown in
 11433 standard output for parsing the output of *date* in those locales.

11434 The range of values for %S extends from 0 to 61 seconds to accommodate the occasional leap
 11435 second or double leap second.

11436 Although certain of the field descriptors in the POSIX locale (such as the name of the month) are
 11437 shown with initial capital letters, this need not be the case in other locales. Programs using these
 11438 fields may need to adjust the capitalization if the output is going to be used at the beginning of a

11439 sentence.

11440 The date string formatting capabilities are intended for use in Gregorian-style calendars,
11441 possibly with a different starting year (or years). The `%x` and `%c` field descriptors, however, are
11442 intended for local representation; these may be based on a different, non-Gregorian calendar.

11443 The `%C` field descriptor was introduced to allow a fallback for the `%EC` (alternative year format
11444 base year); it can be viewed as the base of the current subdivision in the Gregorian calendar. A
11445 century is not calculated as an ordinal number; IEEE Std. 1003.1-200x was published in century
11446 20, not the twenty-first. Both the `%Ey` and `%y` can then be viewed as the offset from `%EC` and
11447 `%C`, respectively.

11448 The `E` and `O` modifiers modify the traditional field descriptors, so that they can always be used,
11449 even if the implementation (or the current locale) does not support the modifier.

11450 The `E` modifier supports alternative date formats, such as the Japanese Emperor's Era, as long as
11451 these are based on the Gregorian calendar system. Extending the `E` modifiers to other date
11452 elements may provide an implementation-defined extension capable of supporting other
11453 calendar systems, especially in combination with the `O` modifier.

11454 The `O` modifier supports time and date formats using the locale's alternative numerical symbols,
11455 such as Kanji or Hindi digits or ordinal number representation.

11456 Non-European locales, whether they use Latin digits in computational items or not, often have
11457 local forms of the digits for use in date formats. This is not totally unknown even in Europe; a
11458 variant of dates uses Roman numerals for the months: the third day of September 1991 would be
11459 written as 3.IX.1991. In Japan, Kanji digits are regularly used for dates; in Arabic-speaking
11460 countries, Hindi digits are used. The `%d`, `%e`, `%H`, `%I`, `%m`, `%S`, `%U`, `%w`, `%W`, and `%y` field
11461 descriptors always return the date and time field in Latin digits (that is, 0 to 9). The `%O` modifier
11462 was introduced to support the use for display purposes of non-Latin digits. In the `LC_TIME`
11463 category in `localedef`, the optional `alt_digits` keyword is intended for this purpose. As an
11464 example, assume the following (partial) `localedef` source:

```
11465 alt_digits  "";"I";"II";"III";"IV";"V";"VI";"VII";"VIII" \
11466           "IX";"X";"XI";"XII"
11467 d_fmt      "%e.%Om.%Y"
```

11468 With the above date, the command:

```
11469 date "+%x"
```

11470 would yield 3.IX.1991. With the same `d_fmt`, but without the `alt_digits`, the command would
11471 yield 3.9.1991.

11472 EXAMPLES

11473 1. The following are input/output examples of `date` used at arbitrary times in the POSIX
11474 locale:

```
11475 $ date
11476 Tue Jun 26 09:58:10 PDT 1990
```

```
11477 $ date "+DATE: %m/%d/%y%nTIME: %H:%M:%S"
11478 DATE: 11/02/91
11479 TIME: 13:36:16
```

```
11480 $ date "+TIME: %r"
11481 TIME: 01:36:32 PM
```

```

11482      2. Examples for Denmark, where the default date and time format is %a %d %b %Y %T %Z:
11483      $ LANG=da_DK.iso_8859-1 date
11484      ons 02 okt 1991 15:03:32 CET

11485      $ LANG=da_DK.iso_8859-1 date "+DATO: %A den %e. %B %Y%nKLOKKEN: %H:%M:%S"
11486      DATO: onsdag den 2. oktober 1991
11487      KLOKKEN: 15:03:56

11488      3. Examples for Germany, where the default date and time format is %a %d.%h.%Y, %T %Z:
11489      $ LANG=De_DE.88591 date
11490      Mi 02.Okt.1991, 15:01:21 MEZ

11491      $ LANG=De_DE.88591 date "+DATUM: %A, %d. %B %Y%nZEIT: %H:%M:%S"
11492      DATUM: Mittwoch, 02. Oktober 1991
11493      ZEIT: 15:02:02

11494      4. Examples for France, where the default date and time format is %a %d %h %Y %Z %T:
11495      $ LANG=Fr_FR.88591 date
11496      Mer 02 oct 1991 MET 15:03:32

11497      $ LANG=Fr_FR.88591 date "+JOUR: %A %d %B %Y%nHEURE: %H:%M:%S"
11498      JOUR: Mercredi 02 octobre 1991
11499      HEURE: 15:03:56

```

11500 RATIONALE

```

11501      Some of the new options for formatting are from the ISO C standard. The -u option was
11502      introduced to allow portable access to Coordinated Universal Time (UTC). The string "GMT0" is
11503      allowed as an equivalent TZ value to be compatible with all of the systems using the BSD
11504      implementation, where this option originated.

11505      The %e format field descriptor (adopted from System V) was added because the ISO C standard
11506      descriptors did not provide any way to produce the historical default date output during the first
11507      nine days of any month.

11508      There are two varieties of day and week numbering supported (in addition to any others created
11509      with the locale-dependent %E and %O modifier characters):

11510      • The historical variety in which Sunday is the first day of the week and the weekdays
11511      preceding the first Sunday of the year are considered week 0. These are represented by %w
11512      and %U. A variant of this is %W, using Monday as the first day of the week, but still referring
11513      to week 0. This view of the calendar was retained because so many historical applications
11514      depend on it and the ISO C standard strftime() function, on which many date
11515      implementations are based, was defined in this way.

11516      • The international standard, based on the ISO 8601:1988 standard where Monday is the first
11517      weekday and the algorithm for the first week number is more complex: If the week (Monday
11518      to Sunday) containing January 1 has four or more days in the new year, then it is week 1;
11519      otherwise, it is week 53 of the previous year, and the next week is week 1. These are
11520      represented by the new field descriptors %u and %V, added as a result of international
11521      comments.

11522      The %C field descriptor was introduced to allow a fallback for the %EC (alternate year format
11523      base year); it can be viewed as the base of the current subdivision in the Gregorian calendar. A
11524      century is not calculated as an ordinal number. The original version of this volume of
11525      IEEE Std. 1003.1-200x was approved in century 19, not the twentieth. Both the %Ey and %y can
11526      then be viewed as the offset from %EC and %C, respectively.

```

11527 **FUTURE DIRECTIONS**

11528 None.

11529 **SEE ALSO**11530 The System Interfaces volume of IEEE Std. 1003.1-200x, *ctime()*, *printf()*11531 **CHANGE HISTORY**

11532 First released in Issue 2.

11533 **Issue 4**

11534 Aligned with the ISO/IEC 9945-2: 1993 standard.

11535 **Issue 5**

11536 Changes are made for Year 2000 alignment.

11537 **Issue 6**11538 The following new requirements on POSIX implementations derive from alignment with the
11539 Single UNIX Specification:

- 11540 • The setting of system date and time is described, including how to interpret two-digit year
- 11541 values if a century is not given.
- 11542 • The *%EX* modified field descriptor is added.

11543 The Open Group corrigenda item U048/2 has been applied, correcting the examples. |

11544 NAME

11545 dd — convert and copy a file

11546 SYNOPSIS

11547 dd [*operand* ...]

11548 DESCRIPTION

11549 The *dd* utility shall copy the specified input file to the specified output file with possible
 11550 conversions using specific input and output block sizes. It shall read the input one block at a
 11551 time, using the specified input block size; it shall then process the block of data actually
 11552 returned, which could be smaller than the requested block size. It shall apply any conversions
 11553 that have been specified and write the resulting data to the output in blocks of the specified
 11554 output block size. If the **bs=expr** operand is specified and no conversions other than **sync**,
 11555 **noerror**, or **notrunc** are requested, the data returned from each input block shall be written as a
 11556 separate output block; if the read returns less than a full block and the **sync** conversion is not
 11557 specified, the resulting output block shall be the same size as the input block. If the **bs=expr**
 11558 operand is not specified, or a conversion other than **sync**, **noerror**, or **notrunc** is requested, the
 11559 input shall be processed and collected into full-sized output blocks until the end of the input is
 11560 reached.

11561 The processing order shall be as follows:

- 11562 1. An input block is read.
- 11563 2. If the input block is shorter than the specified input block size and the **sync** conversion is
 11564 specified, null bytes shall be appended to the input data up to the specified size. (If either
 11565 **block** or **unblock** is also specified, <space> characters shall be appended instead of null
 11566 bytes.) The remaining conversions and output shall include the pad characters as if they
 11567 had been read from the input.
- 11568 3. If the **bs=expr** operand is specified and no conversion other than **sync** or **noerror** is
 11569 requested, the resulting data shall be written to the output as a single block, and the
 11570 remaining steps are omitted.
- 11571 4. If the **swab** conversion is specified, each pair of input data bytes shall be swapped. If there
 11572 is an odd number of bytes in the input block, the last byte in the input record shall not be
 11573 swapped.
- 11574 5. Any remaining conversions (**block**, **unblock**, **lcase**, and **ucase**) shall be performed. These
 11575 conversions shall operate on the input data independently of the input blocking; an input
 11576 or output fixed-length record may span block boundaries.
- 11577 6. The data resulting from input or conversion or both shall be aggregated into output blocks
 11578 of the specified size. After the end of input is reached, any remaining output shall be
 11579 written as a block without padding if **conv=sync** is not specified; thus, the final output
 11580 block may be shorter than the output block size.

11581 OPTIONS

11582 None.

11583 OPERANDS

11584 All of the operands shall be processed before any input is read. The following operands shall be
 11585 supported:

- 11586 **if=file** Specify the input path name; the default is standard input.
- 11587 **of=file** Specify the output path name; the default is standard output. If the **seek=expr**
 11588 conversion is not also specified, the output file shall be truncated before the copy
 11589 begins, unless **conv=notrunc** is specified. If **seek=expr** is specified, but

11590		conv=notrunc is not, the effect of the copy shall be to preserve the blocks in the
11591		output file over which <i>dd</i> seeks, but no other portion of the output file shall be
11592		preserved. (If the size of the seek plus the size of the input file is less than the
11593		previous size of the output file, the output file shall be shortened by the copy.)
11594	ibs=expr	Specify the input block size, in bytes, by <i>expr</i> (default is 512).
11595	obs=expr	Specify the output block size, in bytes, by <i>expr</i> (default is 512).
11596	bs=expr	Set both input and output block sizes to <i>expr</i> bytes, superseding ibs= and obs= . If
11597		no conversion other than sync , noerror , and notrunc is specified, each input block
11598		shall be copied to the output as a single block without aggregating short blocks.
11599	cbs=expr	Specify the conversion block size for block and unblock in bytes by <i>expr</i> (default is
11600		zero). If cbs= is omitted or given a value of zero, using block or unblock produces
11601		unspecified results.
11602 XSI		The application shall ensure that this operand is also specified if the conv=
11603		operand is specified with a value of ascii , ebcdic , or ibm . For a conv= operand
11604		with an ascii value, the input is handled as described for the unblock value, except
11605		that characters are converted to ASCII before any trailing <space> characters are
11606		deleted. For conv= operands with ebcdic or ibm values, the input is handled as
11607		described for the block value except that the characters are converted to EBCDIC
11608		or IBM EBCDIC, respectively, after any trailing <space> characters are added.
11609	skip=n	Skip <i>n</i> input blocks (using the specified input block size) before starting to copy.
11610		On seekable files, the implementation shall read the blocks or seek past them; on
11611		non-seekable files, the blocks shall be read and the data shall be discarded.
11612	seek=n	Skip <i>n</i> blocks (using the specified output block size) from beginning of the output
11613		file before copying. On non-seekable files, existing blocks shall be read and space
11614		from the current end-of-file to the specified offset, if any, filled with null bytes; on
11615		seekable files, the implementation shall seek to the specified offset or read the
11616		blocks as described for non-seekable files.
11617	count=n	Copy only <i>n</i> input blocks.
11618	conv=value[,value ...]	
11619		Where <i>values</i> are comma-separated symbols from the following list:
11620 XSI	ascii	Convert EBCDIC to ASCII; see Table 4-6 (on page 2518).
11621 XSI	ebcdic	Convert ASCII to EBCDIC; see Table 4-6 (on page 2518).
11622 XSI	ibm	Convert ASCII to a different EBCDIC set; see Table 4-7 (on page
11623		2518).
11624		The ascii , ebcdic , and ibm values are mutually-exclusive.
11625	block	Treat the input as a sequence of <newline> character-terminated or
11626		end-of-file-terminated variable-length records independent of the
11627		input block boundaries. Each record shall be converted to a record
11628		with a fixed length specified by the conversion block size. Any
11629		<newline> character shall be removed from the input line; <space>
11630		characters shall be appended to lines that are shorter than their
11631		conversion block size to fill the block. Lines that are longer than the
11632		conversion block size shall be truncated to the largest number of
11633		characters that fit into that size; the number of truncated lines shall
11634		be reported (see the STDERR section).

11635		The block and unblock values are mutually-exclusive.
11636	unblock	Convert fixed-length records to variable length. Read a number of
11637		bytes equal to the conversion block size (or the number of bytes
11638		remaining in the input, if less than the conversion block size), delete
11639		all trailing <space> characters, and append a <newline> character.
11640	lcase	Map uppercase characters specified by the <i>LC_CTYPE</i> keyword
11641		tolower to the corresponding lowercase character. Characters for
11642		which no mapping is specified shall not be modified by this
11643		conversion.
11644		The lcase and ucase symbols are mutually-exclusive.
11645	ucase	Map lowercase characters specified by the <i>LC_CTYPE</i> keyword
11646		toupper to the corresponding uppercase character. Characters for
11647		which no mapping is specified shall not be modified by this
11648		conversion.
11649	swab	Swap every pair of input bytes.
11650	noerror	Do not stop processing on an input error. When an input error
11651		occurs, a diagnostic message shall be written on standard error,
11652		followed by the current input and output block counts in the same
11653		format as used at completion (see the <i>STDERR</i> section). If the sync
11654		conversion is specified, the missing input shall be replaced with null
11655		bytes and processed normally; otherwise, the input block shall be
11656		omitted from the output.
11657	notrunc	Do not truncate the output file. Preserve blocks in the output file not
11658		explicitly written by this invocation of the <i>dd</i> utility. (See also the
11659		preceding of=file operand.)
11660	sync	Pad every input block to the size of the ibs= buffer, appending null
11661		bytes. (If either block or unblock is also specified, append <space>
11662		characters, rather than null bytes.)
11663		The behavior is unspecified if operands other than conv= are specified more than once.
11664		For the bs= , cbs= , ibs= , and obs= operands, the application shall supply an expression
11665		specifying a size in bytes. The expression, <i>expr</i> , can be:
11666		1. A positive decimal number
11667		2. A positive decimal number followed by <i>k</i> , specifying multiplication by 1 024
11668		3. A positive decimal number followed by <i>b</i> , specifying multiplication by 512
11669		4. Two or more positive decimal numbers (with or without <i>k</i> or <i>b</i>) separated by <i>x</i> , specifying
11670		the product of the indicated values
11671		All of the operands are processed before any input is read.
11672	XSI	The following two tables display the octal number character values used for the ascii and ebcdic
11673		conversions (first table) and for the ibm conversion (second table). In both tables, the ASCII
11674		values are the row and column headers and the EBCDIC values are found at their intersections.
11675		For example, ASCII 0012 (LF) is the second row, third column, yielding 0045 in EBCDIC. The
11676		inverted tables (for EBCDIC to ASCII conversion) are not shown, but are in one-to-one
11677		correspondence with these tables. The differences between the two tables are highlighted by
11678		small boxes drawn around five entries.

11679 **Notes to Reviewers**11680 *This section with side shading will not appear in the final copy. - Ed.*

11681 The following 2 tables are commented out of this draft to make document handling easier
 11682 (ability to print 2-up). There are no changes to them. These diagrams are available from the
 11683 Austin Group web site as a separate PDF file.

11684 **Table 4-6** ASCII to EBCDIC Conversion11685 **Table 4-7** ASCII to IBM EBCDIC Conversion11686 **STDIN**11687 If no **if=** operand is specified, the standard input shall be used. See the INPUT FILES section.11688 **INPUT FILES**

11689 The input file can be any file type.

11690 **ENVIRONMENT VARIABLES**11691 The following environment variables shall affect the execution of *dd*:

11692 **LANG** Provide a default value for the internationalization variables that are unset or null.
 11693 If *LANG* is unset or null, the corresponding value from the implementation-
 11694 defined default locale shall be used. If any of the internationalization variables
 11695 contains an invalid setting, the utility shall behave as if none of the variables had
 11696 been defined.

11697 **LC_ALL** If set to a non-empty string value, override the values of all the other
 11698 internationalization variables.

11699 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 11700 characters (for example, single-byte as opposed to multi-byte characters in
 11701 arguments and input files), the classification of characters as uppercase or
 11702 lowercase, and the mapping of characters from one case to the other.

11703 **LC_MESSAGES**

11704 Determine the locale that should be used to affect the format and contents of
 11705 diagnostic messages written to standard error and informative messages written to
 11706 standard output.

11707 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.11708 **ASYNCHRONOUS EVENTS**

11709 For SIGINT, the *dd* utility shall interrupt its current processing, write status information to
 11710 standard error, and exit as though terminated by SIGINT. It shall take the standard action for all
 11711 other signals; see the ASYNCHRONOUS EVENTS section in Section 1.11 (on page 2224).

11712 **STDOUT**

11713 If no **of=** operand is specified, the standard output shall be used. The nature of the output
 11714 depends on the operands selected.

11715 **STDERR**

11716 On completion, *dd* shall write the number of input and output blocks to standard error. In the
 11717 POSIX locale the following formats shall be used:

11718 "%u+%u records in\n", <number of whole input blocks>,
 11719 <number of partial input blocks>

11720 "%u+%u records out\n", <number of whole output blocks>,
 11721 <number of partial output blocks>

11722 A partial input block is one for which *read()* returned less than the input block size. A partial
11723 output block is one that was written with fewer bytes than specified by the output block size.

11724 In addition, when there is at least one truncated block, the number of truncated blocks shall be
11725 written to standard error. In the POSIX locale, the format shall be:

11726 "%u truncated %s\n", *<number of truncated blocks>*, "record" (if
11727 *<number of truncated blocks>* is one) "records" (otherwise)

11728 Diagnostic messages may also be written to standard error.

11729 OUTPUT FILES

11730 If the **of=** operand is used, the output shall be the same as described in the STDOUT section.

11731 EXTENDED DESCRIPTION

11732 None.

11733 EXIT STATUS

11734 The following exit values shall be returned:

11735 0 The input file was copied successfully.

11736 >0 An error occurred.

11737 CONSEQUENCES OF ERRORS

11738 If an input error is detected and the **noerror** conversion has not been specified, any partial
11739 output block shall be written to the output file, a diagnostic message shall be written, and the
11740 copy operation shall be discontinued. If some other error is detected, a diagnostic message shall
11741 be written and the copy operation shall be discontinued.

11742 APPLICATION USAGE

11743 The input and output block size can be specified to take advantage of raw physical I/O.

11744 There are many different versions of the EBCDIC codesets. The ASCII and EBCDIC conversions
11745 specified for the *dd* utility perform conversions for the version specified by the tables.

11746 EXAMPLES

11747 The following command:

```
11748 dd if=/dev/rmt0h of=/dev/rmt1h
```

11749 copies from tape drive 0 to tape drive 1, using a common historical device naming convention.

11750 The following command:

```
11751 dd ibs=10 skip=1
```

11752 strips the first 10 bytes from standard input.

11753 This example reads an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the
11754 ASCII file **x**:

```
11755 dd if=/dev/tape of=x ibs=800 cbs=80 conv=ascii,lcase
```

11756 RATIONALE

11757 The OPTIONS section is listed as “None” because there are no options recognized by historical
11758 *dd* utilities. Certainly, many of the operands could have been designed to use the Utility Syntax
11759 Guidelines, which would have resulted in the classic hyphenated option letters. In this version
11760 of this volume of IEEE Std. 1003.1-200x, *dd* retains its curious JCL-like syntax due to the large
11761 number of applications that depend on the historical implementation.

11762 A suggested implementation technique for **conv=noerror, sync** is to zero (or <space>-fill, if
11763 **blocking** or **unblocking**) the input buffer before each read and to write the contents of the input

11764 buffer to the output even after an error. In this manner, any data transferred to the input buffer
 11765 before the error was detected is preserved. Another point is that a failed read on a regular file or
 11766 a disk generally does not increment the file offset, and *dd* must then seek past the block on which
 11767 the error occurred; otherwise, the input error occurs repetitively. When the input is a magnetic
 11768 tape, however, the tape normally has passed the block containing the error when the error is
 11769 reported, and thus no seek is necessary.

11770 The default **ibs=** and **obs=** sizes are specified as 512 bytes because there are historical (largely
 11771 portable) scripts that assume these values. If they were left unspecified, unusual results could
 11772 occur if an implementation chose an odd block size.

11773 Historical implementations of *dd* used *creat()* when processing **of=file**. This makes the **seek=**
 11774 operand unusable except on special files. The **conv=notrunc** feature was added because more
 11775 recent BSD-based implementations use *open()* (without `O_TRUNC`) instead of *creat()*, but they
 11776 fail to delete output file contents after the data copied.

11777 The *w* multiplier (historically meaning *word*), is used in System V to mean 2 and in 4.2 BSD to
 11778 mean 4. Since *word* is inherently non-portable, its use is not supported by this volume of
 11779 IEEE Std. 1003.1-200x.

11780 Standard EBCDIC does not have the characters '[' and ']'. The values used in the table are
 11781 taken from a common print train that does contain them. Other than those characters, the print
 11782 train values are not filled in, but appear to provide some of the motivation for the historical
 11783 choice of translations reflected here.

11784 The Standard EBCDIC table provides a 1:1 translation for all 256 bytes.

11785 The IBM EBCDIC table does not provide such a translation. The marked cells in the tables differ
 11786 in such a way that:

- 11787 1. EBCDIC 0112 ('ϕ') and 0152 (broken pipe) do not appear in the table.
- 11788 2. EBCDIC 0137 ('¬') translates to/from ASCII 0236 ('^'). In the standard table, EBCDIC
 11789 0232 (no graphic) is used.
- 11790 3. EBCDIC 0241 ('~') translates to/from ASCII 0176 ('~'). In the standard table, EBCDIC
 11791 0137 ('¬') is used.
- 11792 4. 0255 ('[') and 0275 (']') appear twice, once in the same place as for the standard table
 11793 and once in place of 0112 ('ϕ') and 0241 ('~').

11794 In net result:

11795 EBCDIC 0275 (']') displaced EBCDIC 0241 ('~') in cell 0345.

11796 That displaced EBCDIC 0137 ('¬') in cell 0176.

11797 That displaced EBCDIC 0232 (no graphic) in cell 0136.

11798 That replaced EBCDIC 0152 (broken pipe) in cell 0313.

11799 EBCDIC 0255 ('[') replaced EBCDIC 0112 ('ϕ').

11800 This translation, however, reflects historical practice that (ASCII) '~' and '¬' were often
 11801 mapped to each other, as were '[' and 'ϕ'; and ']' and (EBCDIC) '~'.

11802 The **chs** operand is required if any of the **ascii**, **ebcdic**, or **ibm** operands are specified. For the
 11803 **ascii** operand, the input is handled as described for the **unblock** operand except that characters
 11804 are converted to ASCII before the trailing <space>s are deleted. For the **ebcdic** and **ibm**
 11805 operands, the input is handled as described for the **block** operand except that the characters are
 11806 converted to EBCDIC or IBM EBCDIC after the trailing <space>s are added.

- 11807 The **block** and **unblock** keywords are from historical BSD practice.
- 11808 The consistent use of the word **record** in standard error messages matches most historical
11809 practice. An earlier version of System V used **block**, but this has been updated in more recent
11810 releases.
- 11811 Early proposals only allowed two numbers separated by **x** to be used in a product when
11812 specifying **bs=**, **cbs=**, **ibs=**, and **obs=** sizes. This was changed to reflect the historical practice of
11813 allowing multiple numbers in the product as provided by Version 7 and all releases of System V
11814 and BSD.
- 11815 A change to the *swab* conversion is required to match historical practice and is the result of IEEE
11816 PASC Interpretation 1003.2 #03 and #04, submitted for the ISO POSIX-2: 1993 standard.
- 11817 A change to the handling of SIGINT is required to match historical practice and is the result of
11818 IEEE PASC Interpretation 1003.2 #06 submitted for the ISO POSIX-2: 1993 standard.
- 11819 **FUTURE DIRECTIONS**
- 11820 None.
- 11821 **SEE ALSO**
- 11822 *sed, tr*
- 11823 **CHANGE HISTORY**
- 11824 First released in Issue 2.
- 11825 **Issue 4**
- 11826 Aligned with the ISO/IEC 9945-2: 1993 standard.
- 11827 **Issue 5**
- 11828 The second paragraph of the **cbs=** description is reworded and marked EX.
- 11829 FUTURE DIRECTIONS section added.
- 11830 **Issue 6**
- 11831 Changes are made to *swab* conversion and SIGINT handling to align with the IEEE P1003.2b
11832 draft standard.
- 11833 The normative text is reworded to avoid use of the term “must” for application requirements.

11834 NAME

11835 delta — make a delta (change) to an SCCS file (**DEVELOPMENT**)

11836 SYNOPSIS

```
11837 xSI delta [-nps][-g list][-m mrlist][-r SID][-y[comment]] file...
```

11838

11839 DESCRIPTION

11840 The *delta* utility shall be used to permanently introduce into the named SCCS files changes that
11841 were made to the files retrieved by *get* (called the *g-files*, or generated files).

11842 OPTIONS

11843 The *delta* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
11844 12.2, Utility Syntax Guidelines, except that the *-y* option has an optional option-argument. This
11845 optional option-argument cannot be presented as a separate argument.

11846 The following options shall be supported:

11847 *-r SID* Uniquely identify which delta is to be made to the SCCS file. The use of this option
11848 is necessary only if two or more outstanding *get* commands for editing (*get -e*) on
11849 the same SCCS file were done by the same person (login name). The SID value
11850 specified with the *-r* option can be either the SID specified on the *get* command
11851 line or the SID to be made as reported by the *get* utility; see *get* (on page 2685).

11852 *-s* Suppress the report to standard output of the activity associated with each *file*.
11853 See the STDOUT section.

11854 *-n* Specify retention of the edited *g-file* (normally removed at completion of delta
11855 processing).

11856 *-g list* Specify a *list*, (see *get* (on page 2685) for the definition of *list*) of deltas that shall be
11857 ignored when the file is accessed at the change level (SID) created by this delta.

11858 *-m mrlist* Specify a modification request (MR) number that the application shall supply as
11859 the reason for creating the new delta. This is used if the SCCS file has the *v* flag set;
11860 see *admin* (on page 2340).

11861 If *-m* is not used and the standard input is a terminal, the prompt described in the
11862 STDOUT section shall be written to standard output before the standard input is
11863 read; if the standard input is not a terminal, no prompt shall be issued.

11864 MRs in a list shall be separated by <blank>*s*. An unescaped <newline> character
11865 shall terminate the MR list.

11866 If the *v* flag has a value, it shall be taken to be the name of a program which
11867 validates the correctness of the MR numbers. If a non-zero exit status is returned
11868 from the MR number validation program, the *delta* utility shall terminate. (It is
11869 assumed that the MR numbers were not all valid.)

11870 *-y[comment]* Describe the reason for making the delta. The *comment* shall be an arbitrary group
11871 of lines that would meet the definition of a text file. Implementations shall support
11872 *comments* from zero to 512 bytes and may support longer values. A null string
11873 (specified as either *-y*, *-y* " ", or in response to a prompt for a comment) is
11874 considered a valid *comment*.

11875 If *-y* is not specified and the standard input is a terminal, the prompt described in
11876 the STDOUT section shall be written to standard output before the standard input
11877 is read; if the standard input is not a terminal, no prompt shall be issued. An
11878 unescaped <newline> character terminates the comment text.

- 11879 The `-y` option shall be required if the *file* operand is specified as `'-'`.
- 11880 **-p** Write (to standard output) the SCCS file differences before and after the delta is
11881 applied in *diff* format; see *diff* (on page 2529).
- 11882 **OPERANDS**
- 11883 The following operand shall be supported:
- 11884 *file* A path name of an existing SCCS file or a directory. If *file* is a directory, the *delta*
11885 utility shall behave as though each file in the directory were specified as a named
11886 file, except that non-SCCS files (last component of the path name does not begin
11887 with *s*.) and unreadable files shall be silently ignored.
- 11888 If a single instance *file* is specified as `'-'`, the standard input shall be read; each
11889 line of the standard input shall be taken to be the name of an SCCS file to be
11890 processed. Non-SCCS files and unreadable files shall be silently ignored.
- 11891 **STDIN**
- 11892 The standard input shall be a text file used only in the following cases:
- 11893
 - To read an *mrlist* or a *command* (see the `-m` and `-y` options).

11894
 - A *file* operand is specified as `'-'`.

11895 **INPUT FILES**

11896 Input files shall be text files whose data is to be included in the SCCS files. If the first character of
11897 any line of an input file is SOH (binary 001), the results are unspecified.

11898 **ENVIRONMENT VARIABLES**

11899 The following environment variables shall affect the execution of *delta*:

11900 *LANG* Provide a default value for the internationalization variables that are unset or null.
11901 If *LANG* is unset or null, the corresponding value from the implementation-
11902 defined default locale shall be used. If any of the internationalization variables
11903 contains an invalid setting, the utility shall behave as if none of the variables had
11904 been defined.

11905 *LC_ALL* If set to a non-empty string value, override the values of all the other
11906 internationalization variables.

11907 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
11908 characters (for example, single-byte as opposed to multi-byte characters in
11909 arguments and input files).

11910 *LC_MESSAGES*

11911 Determine the locale that should be used to affect the format and contents of
11912 diagnostic messages written to standard error, and informative messages written
11913 to standard output.

11914 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

11915 **ASYNCHRONOUS EVENTS**

11916 Default.

11917 **STDOUT**

11918 The standard output shall be used only for the following messages in the POSIX locale:

11919
 - Prompts (see the `-m` and `-y` options) in the following formats:

11920 "MRs? "

- 11921 "comments? "
- 11922 The MR prompt, if written, shall always precede the comments prompt.
- 11923 • A report of each *file*'s activities (unless the `-s` option is specified) in the following format:
- 11924 " %s\n%d inserted\n%d deleted\n%d unchanged\n", <New SID>,
11925 <number of lines inserted>, <number of lines deleted>,
11926 <number of lines unchanged>
- 11927 **STDERR**
- 11928 Used only for diagnostic messages.
- 11929 **OUTPUT FILES**
- 11930 Any SCCS files updated are files of an unspecified format.
- 11931 **EXTENDED DESCRIPTION**
- 11932 None.
- 11933 **EXIT STATUS**
- 11934 The following exit values shall be returned:
- 11935 0 Successful completion.
- 11936 >0 An error occurred.
- 11937 **CONSEQUENCES OF ERRORS**
- 11938 Default.
- 11939 **APPLICATION USAGE**
- 11940 None.
- 11941 **EXAMPLES**
- 11942 None.
- 11943 **RATIONALE**
- 11944 None.
- 11945 **FUTURE DIRECTIONS**
- 11946 None.
- 11947 **SEE ALSO**
- 11948 *admin, diff, get, prs, rmdel*
- 11949 **CHANGE HISTORY**
- 11950 First released in Issue 2.
- 11951 **Issue 4**
- 11952 Format reorganized.
- 11953 Exceptions to Utility Syntax Guidelines conformance noted.
- 11954 Internationalized environment variable support mandated.
- 11955 **Issue 5**
- 11956 The output format description in the `STDOUT` section is corrected.
- 11957 **Issue 6**
- 11958 The `APPLICATION USAGE` section is added.
- 11959 The normative text is reworded to avoid use of the term "must" for application requirements.
- 11960 The normative text is reworded to emphasise the term "shall" for implementation requirements.

11961 **NAME**

11962 df — report free disk space

11963 **SYNOPSIS**11964 UP XSI df [-k][-P|-t][*file...*]

11965

11966 **DESCRIPTION**

11967 XSI The *df* utility shall write the amount of available space and file slots for file systems on which the
 11968 invoking user has appropriate read access. File systems shall be specified by the *file* operands;
 11969 when none are specified, information shall be written for all file systems. The format of the
 11970 default output from *df* is unspecified, but all space figures are reported in 512-byte units, unless
 11971 the **-k** option is specified. This output shall contain at least the file system names, amount of
 11972 XSI available space on each of these file systems, and the number of free file slots, or *inodes*,
 11973 available; when **-t** is specified, the output contains the total allocated space as well.

11974 **OPTIONS**

11975 The *df* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
 11976 Utility Syntax Guidelines.

11977 The following options shall be supported:

11978 **-k** Use 1024-byte units, instead of the default 512-byte units, when writing space
 11979 figures.

11980 **-P** Produce output in the format described in the STDOUT section.

11981 XSI **-t** Include total allocated-space figures in the output.

11982 **OPERANDS**

11983 The following operand shall be supported:

11984 *file* A path name of a file within the hierarchy of the desired file system. If a file other
 11985 XSI than a FIFO, a regular file, a directory or a special file representing the device
 11986 containing the file system (for example, **/dev/dsk/0s1**) is specified, the results are
 11987 unspecified. Otherwise, *df* shall write the amount of free space in the file system
 11988 containing the specified *file* operand.

11989 **STDIN**

11990 Not used.

11991 **INPUT FILES**

11992 None.

11993 **ENVIRONMENT VARIABLES**11994 The following environment variables shall affect the execution of *df*:

11995 **LANG** Provide a default value for the internationalization variables that are unset or null.
 11996 If **LANG** is unset or null, the corresponding value from the implementation-
 11997 defined default locale shall be used. If any of the internationalization variables
 11998 contains an invalid setting, the utility shall behave as if none of the variables had
 11999 been defined.

12000 **LC_ALL** If set to a non-empty string value, override the values of all the other
 12001 internationalization variables.

12002 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 12003 characters (for example, single-byte as opposed to multi-byte characters in
 12004 arguments).

- 12005 *LC_MESSAGES*
- 12006 Determine the locale that should be used to affect the format and contents of
- 12007 diagnostic messages written to standard error and informative messages written to
- 12008 standard output.
- 12009 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 12010 **ASYNCHRONOUS EVENTS**
- 12011 Default.
- 12012 **STDOUT**
- 12013 When both the **-k** and **-P** options are specified, the following header line shall be written (in the
- 12014 POSIX locale):
- 12015 "Filesystem 1024-blocks Used Available Capacity Mounted on\n"
- 12016 When the **-P** option is specified without the **-k** option, the following header line shall be written
- 12017 (in the POSIX locale):
- 12018 "Filesystem 512-blocks Used Available Capacity Mounted on\n"
- 12019 The implementation may adjust the spacing of the header line and the individual data lines so
- 12020 that the information is presented in orderly columns.
- 12021 The remaining output with **-P** shall consist of one line of information for each specified file
- 12022 system. These lines shall be formatted as follows:
- 12023 "%s %d %d %d %d%% %s\n", *<file system name>*, *<total space>*,
- 12024 *<space used>*, *<space free>*, *<percentage used>*,
- 12025 *<file system root>*
- 12026 In the following list, all quantities expressed in 512-byte units (1 024-byte when **-k** is specified)
- 12027 shall be rounded up to the next higher unit. The fields are:
- 12028 *<file system name>*
- 12029 The name of the file system, in an implementation-defined format.
- 12030 *<total space>* The total size of the file system in 512-byte units. The exact meaning of this figure
- 12031 is implementation-defined, but should include *<space used>*, *<space free>*, plus any
- 12032 space reserved by the system not normally available to a user.
- 12033 *<space used>* The total amount of space allocated to existing files in the file system, in 512-byte
- 12034 units.
- 12035 *<space free>* The total amount of space available within the file system for the creation of new
- 12036 files by unprivileged users, in 512-byte units. When this figure is less than or equal
- 12037 to zero, it shall not be possible to create any new files on the file system without
- 12038 first deleting others, unless the process has appropriate privileges. The figure
- 12039 written may be less than zero.
- 12040 *<percentage used>*
- 12041 The percentage of the normally available space that is currently allocated to all
- 12042 files on the file system. This shall be calculated using the fraction:
- 12043
$$\frac{\textit{<space used>}}{\textit{<space used>} + \textit{<space free>}}$$
- 12044 expressed as a percentage. This percentage may be greater than 100 if *<space free>*
- 12045 is less than zero. The percentage value shall be expressed as a positive integer,
- 12046 with any fractional result causing it to be rounded to the next highest integer.

- 12047 <file system root>
 12048 The directory below which the file system hierarchy appears.
- 12049 XSI The output format is unspecified when `-t` is used.
- 12050 **STDERR**
 12051 Used only for diagnostic messages.
- 12052 **OUTPUT FILES**
 12053 None.
- 12054 **EXTENDED DESCRIPTION**
 12055 None.
- 12056 **EXIT STATUS**
 12057 The following exit values shall be returned:
 12058 0 Successful completion.
 12059 >0 An error occurred.
- 12060 **CONSEQUENCES OF ERRORS**
 12061 Default.
- 12062 **APPLICATION USAGE**
 12063 On most systems, the “name of the file system, in an implementation-defined format” is the
 12064 special file on which the file system is mounted.
 12065 On large file systems, the calculation specified for percentage used can create huge rounding
 12066 errors.
- 12067 **EXAMPLES**
 12068 1. The following example writes portable information about the `/usr` file system:
 12069 `df -P /usr`
 12070 2. Assuming that `/usr/src` is part of the `/usr` file system, the following produces the same
 12071 output as the previous example:
 12072 `df -P /usr/src`
- 12073 **RATIONALE**
 12074 The behavior of `df` with the `-P` option is the default action of the 4.2 BSD `df` utility. The uppercase
 12075 `-P` was selected to avoid collision with a known industry extension using `-p`.
 12076 Historical `df` implementations vary considerably in their default output. It was therefore
 12077 necessary to describe the default output in a loose manner to accommodate all known historical
 12078 implementations and to add a portable option (`-P`) to provide information in a portable format.
 12079 The use of 512-byte units is historical practice and maintains compatibility with `ls` and other
 12080 utilities in this volume of IEEE Std. 1003.1-200x. This does not mandate that the file system itself
 12081 be based on 512-byte blocks. The `-k` option was added as a compromise measure. It was agreed
 12082 by the standard developers that 512 bytes was the best default unit because of its complete
 12083 historical consistency on System V (*versus* the mixed 512/1024-byte usage on BSD systems), and
 12084 that a `-k` option to switch to 1024-byte units was a good compromise. Users who prefer the
 12085 more logical 1024-byte quantity can easily alias `df` to `df -k` without breaking many historical
 12086 scripts relying on the 512-byte units.
 12087 It was suggested that `df` and the various related utilities be modified to access a `BLOCKSIZE`
 12088 environment variable to achieve consistency and user acceptance. Since this is not historical
 12089 practice on any system, it is left as a possible area for system extensions and will be re-evaluated

12090 in a future version if it is widely implemented.

12091 **FUTURE DIRECTIONS**

12092 None.

12093 **SEE ALSO**

12094 *find*

12095 **CHANGE HISTORY**

12096 First released in Issue 2.

12097 **Issue 4**

12098 Aligned with the ISO/IEC 9945-2:1993 standard.

12099 **Issue 6**

12100 This utility is now marked as part of the User Portability Utilities option.

12101 **NAME**

12102 diff — compare two files

12103 **SYNOPSIS**12104 diff [-c | -e | -f | -C *n*][-br] *file1 file2*12105 **DESCRIPTION**

12106 The *diff* utility shall compare the contents of *file1* and *file2* and write to standard output a list of
 12107 changes necessary to convert *file1* into *file2*. This list should be minimal. No output shall be
 12108 produced if the files are identical.

12109 **OPTIONS**

12110 The *diff* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 12111 12.2, Utility Syntax Guidelines.

12112 The following options shall be supported:

12113 **-b** Cause any amount of white space at the end of a line to be treated as a single
 12114 <newline> character (that is, the white-space characters preceding the <newline>
 12115 character are ignored) and other strings of white-space characters, not including
 12116 <newline> characters, to compare equal.

12117 **-c** Produce output in a form that provides three lines of context.

12118 **-C *n*** Produce output in a form that provides *n* lines of context (where *n* shall be
 12119 interpreted as a positive decimal integer).

12120 **-e** Produce output in a form suitable as input for the *ed* utility, which can then be
 12121 used to convert *file1* into *file2*.

12122 **-f** Produce output in an alternative form, similar in format to **-e**, but not intended to
 12123 be suitable as input for the *ed* utility, and in the opposite order.

12124 **-r** Apply *diff* recursively to files and directories of the same name when *file1* and *file2*
 12125 are both directories.

12126 **OPERANDS**

12127 The following operands shall be supported:

12128 *file1, file2* A path name of a file to be compared. If either the *file1* or *file2* operand is '-', the
 12129 standard input shall be used in its place.

12130 If both *file1* and *file2* are directories, *diff* shall not compare block special files, character special
 12131 files, or FIFO special files to any files and shall not compare regular files to directories. The
 12132 system documentation shall specify the behavior of *diff* on implementation-defined file types not
 12133 specified by the System Interfaces volume of IEEE Std. 1003.1-200x when found in directories.
 12134 Further details are as specified in **Diff Directory Comparison Format** (on page 2530).

12135 If only one of *file1* and *file2* is a directory, *diff* shall be applied to the non-directory file and the file
 12136 contained in the directory file with a file name that is the same as the last component of the non-
 12137 directory file.

12138 **STDIN**

12139 The standard input shall be used only if one of the *file1* or *file2* operands references standard
 12140 input. See the INPUT FILES section.

12141 **INPUT FILES**

12142 The input files shall be text files.

12143 **Notes to Reviewers**12144 *This section with side shading will not appear in the final copy. - Ed.*

12145 D3, XCU, ERN 75 proposes adding the following text: "If a file which is not a text file is
 12146 encountered, a binary comparison shall be performed, and if they are not identical, an
 12147 unspecified message containing the two file names and the string "differ" shall be produced." The
 12148 reviewers agreed in principle; however, this change needs further cleanup such as the locale and
 12149 output formats specifying before it can be made.

12150 **ENVIRONMENT VARIABLES**12151 The following environment variables shall affect the execution of *diff*:

12152 **LANG** Provide a default value for the internationalization variables that are unset or null.
 12153 If *LANG* is unset or null, the corresponding value from the implementation-
 12154 defined default locale shall be used. If any of the internationalization variables
 12155 contains an invalid setting, the utility shall behave as if none of the variables had
 12156 been defined.

12157 **LC_ALL** If set to a non-empty string value, override the values of all the other
 12158 internationalization variables.

12159 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 12160 characters (for example, single-byte as opposed to multi-byte characters in
 12161 arguments and input files).

12162 **LC_MESSAGES**

12163 Determine the locale that should be used to affect the format and contents of
 12164 diagnostic messages written to standard error and informative messages written to
 12165 standard output.

12166 **LC_TIME** Determine the locale for affecting the format of file timestamps written with the
 12167 **-C** and **-c** options.

12168 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

12169 **TZ** Determine the locale for affecting the timezone used for calculating file
 12170 timestamps written with the **-C** and **-c** options.

12171 **ASYNCHRONOUS EVENTS**

12172 Default.

12173 **STDOUT**12174 **Diff Directory Comparison Format**12175 If both *file1* and *file2* are directories, the following output formats shall be used.

12176 In the POSIX locale, each file that is present in only one directory shall be reported using the
 12177 following format:

12178 "Only in %s: %s\n", *<directory pathname>*, *<filename>*

12179 In the POSIX locale, subdirectories that are common to the two directories may be reported with
 12180 the following format:

12181 "Common subdirectories: %s and %s\n", *<directory1 pathname>*,
 12182 *<directory2 pathname>*

12183 For each file common to the two directories if the two files are not to be compared, the following
 12184 format shall be used in the POSIX locale:

12185 "File %s is a %s while file %s is a %s\n", <directory1 pathname>,
 12186 <file type of directory1 pathname>, <directory2 pathname>,
 12187 <file type of directory2 pathname>

12188 For each file common to the two directories, if the files are compared and are identical, no output
 12189 shall be written. If the two files differ, the following format is written:

12190 "diff %s %s %s\n", <diff_options>, <filename1>, <filename2>

12191 where <diff_options> are the options as specified on the command line. Depending on these
 12192 options, one of the following output formats shall be used to write the differences.

12193 All directory path names listed in this section shall be relative to the original command line
 12194 arguments. All other names of files listed in this section are file names (path name components).

12195 **Diff Default Output Format**

12196 The default (without **-e**, **-f**, **-c**, or **-C** options) *diff* utility output shall contain lines of these
 12197 forms:

12198 "%da%d\n", <num1>, <num2>

12199 "%da%d,%d\n", <num1>, <num2>, <num3>

12200 "%dd%d\n", <num1>, <num2>

12201 "%d,%dd%d\n", <num1>, <num2>, <num3>

12202 "%dc%d\n", <num1>, <num2>

12203 "%d,%dc%d\n", <num1>, <num2>, <num3>

12204 "%dc%d,%d\n", <num1>, <num2>, <num3>

12205 "%d,%dc%d,%d\n", <num1>, <num2>, <num3>, <num4>

12206 These lines resemble *ed* subcommands to convert *file1* into *file2*. The line numbers before the
 12207 action letters shall pertain to *file1*; those after shall pertain to *file2*. Thus, by exchanging *a* for *d*
 12208 and reading the line in reverse order, one can also determine how to convert *file2* into *file1*. As in
 12209 *ed*, identical pairs (where *num1*= *num2*) are abbreviated as a single number.

12210 Following each of these lines, *diff* shall write to standard output all lines affected in the first file
 12211 using the format:

12212 "<Δ%s", <line>

12213 and all lines affected in the second file using the format:

12214 ">Δ%s", <line>

12215 If there are lines affected in both *file1* and *file2* (as with the *c* subcommand), the changes are
 12216 separated with a line consisting of three hyphens:

12217 "——\n"

12218 **Diff -e Output Format**

12219 With the **-e** option, a script shall be produced that shall, when provided as input to *ed*, along
 12220 with an appended **w** (write) command, convert *file1* into *file2*. Only the **a** (append), **c** (change), **d**
 12221 (delete), **i** (insert), and **s** (substitute) commands of *ed* shall be used in this script. Text lines,
 12222 except those consisting of the single character period (`' . '`), shall be output as they appear in the
 12223 file.

12224 **Diff -f Output Format**

12225 With the **-f** option, an alternative format of script shall be produced. It is similar to that
 12226 produced by **-e**, with the following differences:

- 12227 1. It is expressed in reverse sequence; the output of **-e** orders changes from the end of the file
 12228 to the beginning; the **-f** from beginning to end.
- 12229 2. The command form `<lines> <command-letter>` used by **-e** is reversed. For example,
 12230 `10c` with **-e** would be `c10` with **-f**.
- 12231 3. The form used for ranges of line numbers is `<space>` character-separated, rather than
 12232 comma-separated.

12233 **Diff -c or -C Output Format**

12234 With the **-c** or **-C** option, the output format shall consist of affected lines along with
 12235 surrounding lines of context. The affected lines shall show which ones need to be deleted or
 12236 changed in *file1*, and those added from *file2*. With the **-c** option, three lines of context, if
 12237 available, shall be written before and after the affected lines. With the **-C** option, the user can
 12238 specify how many lines of context are written. The exact format follows.

12239 The name and last modification time of each file shall be output in the following format:

```
12240 "**** %s %s\n", file1, <file1 timestamp>  
12241 "---- %s %s\n", file2, <file2 timestamp>
```

12242 Each `<file>` field shall be the path name of the corresponding file being compared. The path
 12243 name written for standard input is unspecified.

12244 In the POSIX locale, each `<timestamp>` field shall be equivalent to the output from the following
 12245 command:

```
12246 date "+%a %b %e %T %Y"
```

12247 without the trailing `<newline>` character, executed at the time of last modification of the
 12248 corresponding file (or the current time, if the file is standard input).

12249 Then, the following output formats shall be applied for every set of changes.

12250 First, a line shall be written in the following format:

```
12251 "*****\n"
```

12252 Next, the range of lines in *file1* shall be written in the following format:

```
12253 "**** %d,%d ****\n", <beginning line number>, <ending line number>
```

12254 Next, the affected lines along with lines of context (unaffected lines) shall be written. Unaffected
 12255 lines shall be written in the following format:

```
12256 "ΔΔ%s", <unaffected_line>
```


12257 Deleted lines shall be written as:

12258 `"-Δ%s", <deleted_line>`

12259 Changed lines shall be written as:

12260 `"!Δ%s", <changed_line>`

12261 Next, the range of lines in *file2* shall be written in the following format:

12262 `"—— %d,%d ——\n", <beginning line number>, <ending line number>`

12263 Then, lines of context and changed lines shall be written as described in the previous formats.

12264 Lines added from *file2* shall be written in the following format:

12265 `" +Δ%s", <added_line>`

12266 **STDERR**

12267 Used only for diagnostic messages.

12268 **OUTPUT FILES**

12269 None.

12270 **EXTENDED DESCRIPTION**

12271 None.

12272 **EXIT STATUS**

12273 The following exit values shall be returned:

12274 0 No differences were found.

12275 1 Differences were found.

12276 >1 An error occurred.

12277 **CONSEQUENCES OF ERRORS**

12278 Default.

12279 **APPLICATION USAGE**

12280 If lines at the end of a file are changed and other lines are added, *diff* output may show this as a delete and add, as a change, or as a change and add; *diff* is not expected to know which happened and users should not care about the difference in output as long as it clearly shows the differences between the files.

12284 **EXAMPLES**

12285 If **dir1** is a directory containing a directory named **x**, **dir2** is a directory containing a directory

12286 named **x**, **dir1/x** and **dir2/x** both contain files named **date.out**, and **dir2/x** contains a file named **y**,

12287 the command:

12288 `diff -r dir1 dir2`

12289 could produce output similar to:

12290 Common subdirectories: dir1/x and dir2/x

12291 Only in dir2/x: y

12292 `diff -r dir1/x/date.out dir2/x/date.out`

12293 `lc1`

12294 `< Mon Jul 2 13:12:16 PDT 1990`

12295 `——`

12296 `> Tue Jun 19 21:41:39 PDT 1990`

12297 RATIONALE

12298 The `-h` option was omitted because it was insufficiently specified and does not add to
12299 applications portability.

12300 Historical implementations employ algorithms that do not always produce a minimum list of
12301 differences; the current language about making every effort is the best this volume of
12302 IEEE Std. 1003.1-200x can do, as there is no metric that could be employed to judge the quality of
12303 implementations against any and all file contents. The statement “This list should be minimal”
12304 clearly implies that implementations are not expected to provide the following output when
12305 comparing two 100-line files that differ in only one character on a single line:

```
12306 1,100c1,100  
12307 all 100 lines from file1 preceded with "< "  
12308 _____  
12309 all 100 lines from file2 preceded with "> "
```

12310 The “Only in” messages required when the `-r` option is specified are not used by most historical
12311 implementations if the `-e` option is also specified. It is required here because it provides useful
12312 information that must be provided to update a target directory hierarchy to match a source
12313 hierarchy. The “Common subdirectories” messages are written by System V and 4.3 BSD when
12314 the `-r` option is specified. They are allowed here but are not required because they are reporting
12315 on something that is the same, not reporting a difference, and are not needed to update a target
12316 hierarchy.

12317 The `-c` option, which writes output in a format using lines of context, has been included. The
12318 format is useful for a variety of reasons, among them being much improved readability and the
12319 ability to understand difference changes when the target file has line numbers that differ from
12320 another similar, but slightly different, copy. The *patch* utility is most valuable when working
12321 with difference listings using the context format. The BSD version of `-c` takes an optional
12322 argument specifying the amount of context. Rather than overloading `-c` and breaking the Utility
12323 Syntax Guidelines for *diff*, the standard developers decided to add a separate option for
12324 specifying a context *diff* with a specified amount of context (`-C`). Also, the format for context
12325 *diffs* was extended slightly in 4.3 BSD to allow multiple changes that are within context lines
12326 from each other to be merged together. The output format contains an additional four asterisks
12327 after the range of affected lines in the first file name. This was to provide a flag for old programs
12328 (like old versions of *patch*) that only understand the old context format. The version of context
12329 described here does not require that multiple changes within context lines be merged, but it does
12330 not prohibit it either. The extension is upward-compatible, so any vendors that wish to retain the
12331 old version of *diff* can do so by adding the extra four asterisks (that is, utilities that currently use
12332 *diff* and understand the new merged format will also understand the old unmerged format, but
12333 not *vice versa*).

12334 The substitute command was added as an additional format for the `-e` option. This was added to
12335 provide implementations a way to fix the classic “dot alone on a line” bug present in many
12336 versions of *diff*. Since many implementations have fixed this bug, the standard developers
12337 decided not to standardize broken behavior, but rather to provide the necessary tool for fixing
12338 the bug. One way to fix this bug is to output two periods whenever a lone period is needed, then
12339 terminate the append command with a period, and then use the substitute command to convert
12340 the two periods into one period.

12341 The BSD-derived `-r` option was added to provide a mechanism for using *diff* to compare two file
12342 system trees. This behavior is useful, is standard practice on all BSD-derived systems, and is not
12343 easily reproducible with the *find* utility.

12344 The requirement that *diff* not compare files in some circumstances, even though they have the
12345 same name, is based on the actual output of historical implementations. The message specified

- 12346 here is already in use when a directory is being compared to a non-directory. It is extended here
12347 to preclude the problems arising from running into FIFOs and other files that would cause *diff* to
12348 hang waiting for input with no indication to the user that *diff* was hung. In most common usage,
12349 *diff -r* should indicate differences in the file hierarchies, not the difference of contents of devices
12350 pointed to by the hierarchies.
- 12351 Many early implementations of *diff* require seekable files. Since the System Interfaces volume of
12352 IEEE Std. 1003.1-200x supports named pipes, the standard developers decided that such a
12353 restriction was unreasonable. Note also that the allowed file name – almost always refers to a
12354 pipe.
- 12355 No directory search order is specified for *diff*. The historical ordering is, in fact, not optimal, in
12356 that it prints out all of the differences at the current level, including the statements about all
12357 common subdirectories before recursing into those subdirectories.
- 12358 The message:
- 12359 `"diff %s %s %s\n", <diff_options>, <filename1>, <filename2>`
- 12360 does not vary by locale because it is the representation of a command, not an English sentence.
- 12361 **FUTURE DIRECTIONS**
- 12362 None.
- 12363 **SEE ALSO**
- 12364 *cmp, comm, ed*
- 12365 **CHANGE HISTORY**
- 12366 First released in Issue 2.
- 12367 **Issue 4**
- 12368 Aligned with the ISO/IEC 9945-2:1993 standard.
- 12369 **Issue 5**
- 12370 FUTURE DIRECTIONS section added.
- 12371 **Issue 6**
- 12372 The following new requirements on POSIX implementations derive from alignment with the
12373 Single UNIX Specification:
- 12374
 - The `-f` option is added.
- 12375 The output format for `-c` or `-C` format is changed to align with changes to the IEEE P1003.2b
12376 draft standard resulting from IEEE PASC Interpretation 1003.2 #71.
- 12377 The normative text is reworded to avoid use of the term “must” for application requirements.

12378 **NAME**

12379 dirname — return the directory portion of path name

12380 **SYNOPSIS**12381 dirname *string*12382 **DESCRIPTION**

12383 The *string* operand shall be treated as a path name, as defined in the Base Definitions volume of
 12384 IEEE Std. 1003.1-200x, Section 3.268, Path Name. The string *string* shall be converted to the name
 12385 of the directory containing the file name corresponding to the last path name component in
 12386 *string*, performing actions equivalent to the following steps in order:

- 12387 1. If *string* is //, skip steps 2 to 5.
- 12388 2. If *string* consists entirely of slash characters, *string* shall be set to a single slash character. In
 12389 this case, skip steps 3 to 8.
- 12390 3. If there are any trailing slash characters in *string*, they shall be removed.
- 12391 4. If there are no slash characters remaining in *string*, *string* shall be set to a single period
 12392 character. In this case, skip steps 5 to 8.
- 12393 5. If there are any trailing non-slash characters in *string*, they shall be removed.
- 12394 6. If the remaining *string* is //, it is implementation-defined whether steps 7 and 8 are skipped
 12395 or processed.
- 12396 7. If there are any trailing slash characters in *string*, they shall be removed.
- 12397 8. If the remaining *string* is empty, *string* shall be set to a single slash character.

12398 The resulting string shall be written to standard output.

12399 **OPTIONS**

12400 None.

12401 **OPERANDS**

12402 The following operand shall be supported:

12403 *string* A string.12404 **STDIN**

12405 Not used.

12406 **INPUT FILES**

12407 None.

12408 **ENVIRONMENT VARIABLES**12409 The following environment variables shall affect the execution of *dirname*:

- | | | |
|---|-----------------|---|
| 12410
12411
12412
12413
12414 | <i>LANG</i> | Provide a default value for the internationalization variables that are unset or null. If <i>LANG</i> is unset or null, the corresponding value from the implementation-defined default locale will be used. If any of the internationalization variables contains an invalid setting, the utility shall behave as if none of the variables had been defined. |
| 12415
12416 | <i>LC_ALL</i> | If set to a non-empty string value, override the values of all the other internationalization variables. |
| 12417
12418
12419 | <i>LC_CTYPE</i> | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments). |

12420 **LC_MESSAGES**

12421 Determine the locale that should be used to affect the format and contents of
 12422 diagnostic messages written to standard error.

12423 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

12424 **ASYNCHRONOUS EVENTS**

12425 Default.

12426 **STDOUT**

12427 The *dirname* utility shall write a line to the standard output in the following format:

12428 "%s\n", <resulting string>

12429 **STDERR**

12430 Used only for diagnostic messages.

12431 **OUTPUT FILES**

12432 None.

12433 **EXTENDED DESCRIPTION**

12434 None.

12435 **EXIT STATUS**

12436 The following exit values shall be returned:

12437 0 Successful completion.

12438 >0 An error occurred.

12439 **CONSEQUENCES OF ERRORS**

12440 Default.

12441 **APPLICATION USAGE**

12442 The definition of *pathname* specifies implementation-defined behavior for path names starting
 12443 with two slash characters. Therefore, applications shall not arbitrarily add slashes to the
 12444 beginning of a path name unless they can ensure that there are more or less than two or are
 12445 prepared to deal with the implementation-defined consequences.

12446 **EXAMPLES**

	Command	Results
12447	<i>dirname</i> /	/
12448	<i>dirname</i> //	/ or //
12449	<i>dirname</i> /a/b/	/a
12450	<i>dirname</i> //a//b//	//a
12451	<i>dirname</i>	Unspecified
12452	<i>dirname</i> a	.(\$? = 0)
12453	<i>dirname</i> ""	.(\$? = 0)
12454	<i>dirname</i> /a	/
12455	<i>dirname</i> /a/b	/a
12456	<i>dirname</i> a/b	a
12457		

12458 **RATIONALE**

12459 The *dirname* utility originated in System III. It has evolved through the System V releases to a
 12460 version that matches the requirements specified in this description in System V Release 3. 4.3
 12461 BSD and earlier versions did not include *dirname*.

12462 The behaviors of *basename* and *dirname* in this volume of IEEE Std. 1003.1-200x have been
 12463 coordinated so that when *string* is a valid path name:

12464 \$(basename "*string*")

12465 would be a valid file name for the file in the directory:

12466 \$(dirname "*string*")

12467 This would not work for the versions of these utilities in early proposals due to the way
12468 processing of trailing slashes was specified. Consideration was given to leaving processing
12469 unspecified if there were trailing slashes, but this cannot be done; the Base Definitions volume of
12470 IEEE Std. 1003.1-200x, Section 3.268, Path Name allows trailing slashes. The *basename* and
12471 *dirname* utilities have to specify consistent handling for all valid path names.

12472 **FUTURE DIRECTIONS**

12473 None.

12474 **SEE ALSO**

12475 *basename*, Section 2.5 (on page 2241)

12476 **CHANGE HISTORY**

12477 First released in Issue 2.

12478 **Issue 4**

12479 Aligned with the ISO/IEC 9945-2: 1993 standard.

12480 **NAME**12481 `du` — estimate file space usage12482 **SYNOPSIS**12483 UP `du [-a | -s][-kx][-H | -L][file ...]`

12484

12485 **DESCRIPTION**

12486 By default, the *du* utility shall write to standard output the size of the file space allocated to, and
 12487 the size of the file space allocated to each subdirectory of, the file hierarchy rooted in each of the
 12488 specified files. By default, when a symbolic link is encountered on the command line or in the
 12489 file hierarchy, *du* shall count the size of the symbolic link (rather than the file referenced by the
 12490 link), and shall not follow the link to another portion of the file hierarchy. The size of the file
 12491 space allocated to a file of type directory shall be defined as the sum total of space allocated to
 12492 all files in the file hierarchy rooted in the directory plus the space allocated to the directory itself.

12493 When *du* cannot *stat()* files or *stat()* or read directories, it shall report an error condition and the
 12494 final exit status is affected. Files with multiple links shall be counted and written for only one
 12495 entry. The directory entry that is selected in the report is unspecified. By default, file sizes shall
 12496 be written in 512-byte units, rounded up to the next 512-byte unit.

12497 **OPTIONS**

12498 The *du* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 12499 12.2, Utility Syntax Guidelines.

12500 The following options shall be supported:

12501 **-a** In addition to the default output, report the size of each file not of type directory in
 12502 the file hierarchy rooted in the specified file. Regardless of the presence of the **-a**
 12503 option, non-directories given as *file* operands shall always be listed.

12504 **-H** If a symbolic link is specified on the command line, *du* shall count the size of the
 12505 file or file hierarchy referenced by the link.

12506 **-k** Write the files sizes in units of 1024 bytes, rather than the default 512-byte units.

12507 **-L** If a symbolic link is specified on the command line or encountered during the
 12508 traversal of a file hierarchy, *du* shall count the size of the file or file hierarchy
 12509 referenced by the link.

12510 **-s** Instead of the default output, report only the total sum for each of the specified
 12511 files.

12512 **-x** When evaluating file sizes, evaluate only those files that have the same device as
 12513 the file specified by the *file* operand.

12514 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered
 12515 an error. The last option specified shall determine the behavior of the utility.

12516 **OPERANDS**

12517 The following operand shall be supported:

12518 *file* The path name of a file whose size is to be written. If no *file* is specified, the current
 12519 directory shall be used.

12520 **STDIN**

12521 Not used.

12522 **INPUT FILES**

12523 None.

12524 **ENVIRONMENT VARIABLES**12525 The following environment variables shall affect the execution of *du*:

12526 *LANG* Provide a default value for the internationalization variables that are unset or null.
12527 If *LANG* is unset or null, the corresponding value from the implementation-
12528 defined default locale shall be used. If any of the internationalization variables
12529 contains an invalid setting, the utility shall behave as if none of the variables had
12530 been defined.

12531 *LC_ALL* If set to a non-empty string value, override the values of all the other
12532 internationalization variables.

12533 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
12534 characters (for example, single-byte as opposed to multi-byte characters in
12535 arguments).

12536 *LC_MESSAGES*
12537 Determine the locale that should be used to affect the format and contents of
12538 diagnostic messages written to standard error.

12539 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

12540 **ASYNCHRONOUS EVENTS**

12541 Default.

12542 **STDOUT**

12543 The output from *du* shall consist of the amount of the space allocated to a file and the name of
12544 the file, in the following format:

12545 "%d %s\n", <size>, <pathname>

12546 **STDERR**

12547 Used only for diagnostic messages.

12548 **OUTPUT FILES**

12549 None.

12550 **EXTENDED DESCRIPTION**

12551 None.

12552 **EXIT STATUS**

12553 The following exit values shall be returned:

12554 0 Successful completion.

12555 >0 An error occurred.

12556 **CONSEQUENCES OF ERRORS**

12557 Default.

12558 **APPLICATION USAGE**

12559 None.

12560 **EXAMPLES**

12561 None.

12562 **RATIONALE**

12563 The use of 512-byte units is historical practice and maintains compatibility with *ls* and other
12564 utilities in this volume of IEEE Std. 1003.1-200x. This does not mandate that the file system itself
12565 be based on 512-byte blocks. The **-k** option was added as a compromise measure. It was agreed
12566 by the standard developers that 512 bytes was the best default unit because of its complete
12567 historical consistency on System V (*versus* the mixed 512/1 024-byte usage on BSD systems), and
12568 that a **-k** option to switch to 1 024-byte units was a good compromise. Users who prefer the
12569 1 024-byte quantity can easily alias *du* to *du -k* without breaking the many historical scripts
12570 relying on the 512-byte units.

12571 The **-b** option was added to an early proposal to provide a resolution to the situation where
12572 System V and BSD systems give figures for file sizes in *blocks*, which is an implementation-
12573 defined concept. (In common usage, the block size is 512 bytes for System V and 1 024 bytes for
12574 BSD systems.) However, **-b** was later deleted, since the default was eventually decided as 512-
12575 byte units.

12576 Historical file systems provided no way to obtain exact figures for the space allocation given to
12577 files. There are two known areas of inaccuracies in historical file systems: cases of *indirect blocks*
12578 being used by the file system or *sparse* files yielding incorrectly high values. An indirect block is
12579 space used by the file system in the storage of the file, but that need not be counted in the space
12580 allocated to the file. A *sparse* file is one in which an *lseek()* call has been made to a position
12581 beyond the end of the file and data has subsequently been written at that point. A file system
12582 need not allocate all the intervening zero-filled blocks to such a file. It is up to the
12583 implementation to define exactly how accurate its methods are.

12584 The **-a** and **-s** options were mutually-exclusive in the original version of *du*. The POSIX Shell
12585 and Utilities description is implied by the language in the SVID where **-s** is described as causing
12586 “only the grand total” to be reported. Some systems may produce output for **-sa**, but a Strictly
12587 Conforming POSIX Shell and Utilities Application cannot use that combination.

12588 The **-a** and **-s** options were adopted from the SVID except that the System V behavior of not
12589 listing non-directories explicitly given as operands, unless the **-a** option is specified, was
12590 considered a bug; the BSD-based behavior (report for all operands) is mandated. The default
12591 behavior of *du* in the SVID with regard to reporting the failure to read files (it produces no
12592 messages) was considered counter-intuitive, and thus it was specified that the POSIX Shell and
12593 Utilities default behavior shall be to produce such messages. These messages can be turned off
12594 with shell redirection to achieve the System V behavior.

12595 The **-x** option is historical practice on recent BSD systems. It has been adopted by this volume of
12596 IEEE Std. 1003.1-200x because there was no other historical method of limiting the *du* search to a
12597 single file hierarchy. This limitation of the search is necessary to make it possible to obtain file
12598 space usage information about a file system on which other file systems are mounted, without
12599 having to resort to a lengthy *find* and *awk* script.

12600 **FUTURE DIRECTIONS**

12601 None.

12602 **SEE ALSO**12603 *ls*12604 **CHANGE HISTORY**

12605 First released in Issue 2.

12606 **Issue 4**

12607 Aligned with the ISO/IEC 9945-2: 1993 standard.

12608 **Issue 6**

12609 This utility is now marked as part of the User Portability Utilities option.

12610 The APPLICATION USAGE section is added.

12611 This utility is reinstated, as the LEGACY marking was incorrect in Issue 5.

12612 The obsolescent `-r` option has been removed.12613 The Open Group corrigenda item U025/3 has been applied. The *du* utility had incorrectly been
12614 marked LEGACY.12615 The `-H` and `-L` options for symbolic links are added as described in the IEEE P1003.2b draft
12616 standard.

12617 **NAME**

12618 echo — write arguments to standard output

12619 **SYNOPSIS**12620 echo [*string* ...]12621 **DESCRIPTION**12622 The *echo* utility writes its arguments to standard output, followed by a <newline> character. If
12623 there are no arguments, only the <newline> character is written.12624 **OPTIONS**12625 The *echo* utility shall not recognize the "—" argument in the manner specified by Guideline 10
12626 of the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2, Utility Syntax Guidelines;
12627 "—" shall be recognized as a string operand.

12628 Implementations shall not support any options.

12629 **OPERANDS**

12630 The following operands shall be supported:

12631 *string* A string to be written to standard output. If any operand is **-n**, it shall be treated as
12632 a string, not an option. The following character sequences shall be recognized
12633 within any of the arguments:

12634 \a Write an <alert> character.

12635 \b Write a <backspace> character.

12636 \c Suppress the <newline> character that otherwise follows the final
12637 argument in the output. All characters following the '\c' in the
12638 arguments shall be ignored.

12639 \f Write a <form-feed> character.

12640 \n Write a <newline> character.

12641 \r Write a <carriage-return> character.

12642 \t Write a <tab> character.

12643 \v Write a <vertical-tab> character.

12644 \\ Write a backslash character.

12645 \0*num* Write an 8-bit value that is the zero, one, two, or three-digit octal number
12646 *num*.12647 **STDIN**

12648 Not used.

12649 **INPUT FILES**

12650 None.

12651 **ENVIRONMENT VARIABLES**12652 The following environment variables shall affect the execution of *echo*:12653 *LANG* Provide a default value for the internationalization variables that are unset or null.
12654 If *LANG* is unset or null, the corresponding value from the implementation-
12655 defined default locale shall be used. If any of the internationalization variables
12656 contains an invalid setting, the utility shall behave as if none of the variables had
12657 been defined.

12658 *LC_ALL* If set to a non-empty string value, override the values of all the other
12659 internationalization variables.

12660 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
12661 characters (for example, single-byte as opposed to multi-byte characters in
12662 arguments).

12663 *LC_MESSAGES*
12664 Determine the locale that should be used to affect the format and contents of
12665 diagnostic messages written to standard error.

12666 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

12667 **ASYNCHRONOUS EVENTS**
12668 Default.

12669 **STDOUT**
12670 The *echo* utility arguments shall be separated by single <space> characters and a <newline>
12671 character follows the last argument. Output transformations shall occur based on the escape
12672 sequences in the input. See the OPERANDS section.

12673 **STDERR**
12674 Used only for diagnostic messages.

12675 **OUTPUT FILES**
12676 None.

12677 **EXTENDED DESCRIPTION**
12678 None.

12679 **EXIT STATUS**
12680 The following exit values shall be returned:
12681 0 Successful completion.
12682 >0 An error occurred.

12683 **CONSEQUENCES OF ERRORS**
12684 Default.

12685 **APPLICATION USAGE**
12686 In the ISO/IEC 9945-2:1993 standard, it was not possible to use *echo* portably across all systems
12687 that were not XSI-conformant unless both *-n* (as the first argument) and escape sequences were
12688 omitted.

12689 The *printf* utility can be used portably to emulate any of the traditional behaviors of the *echo*
12690 utility as follows:

12691 • The historic System V *echo* and the current requirements in this volume of
12692 IEEE Std. 1003.1-200x are equivalent to:

12693 `printf "%b\n" "$*"`

12694 • The BSD *echo* is equivalent to:

12695 `if ["X$1" = "X-n"]`
12696 `then`
12697 `shift`
12698 `printf "%s" "$*"`
12699 `else`
12700 `printf "%s\n" "$*"`

- 12701 *fi*
- 12702 New applications are encouraged to use *printf* instead of *echo*.
- 12703 **EXAMPLES**
- 12704 None.
- 12705 **RATIONALE**
- 12706 The *echo* utility has not been made obsolescent because of its extremely widespread use in historical applications. Portable applications that wish to do prompting without <newline>s or that could possibly be expecting to echo a *-n*, should use the new *printf* utility derived from the Ninth Edition system.
- 12707
- 12708
- 12709
- 12710 As specified, *echo* writes its arguments in the simplest of ways. The two different historical versions of *echo* vary in fatally incompatible ways.
- 12711
- 12712 The BSD *echo* checks the first argument for the string *-n* which causes it to suppress the <newline> character that would otherwise follow the final argument in the output.
- 12713
- 12714 The System V *echo* does not support any options, but allows escape sequences within its operands, as described in the OPERANDS section.
- 12715
- 12716 The *echo* utility does not support Utility Syntax Guideline 10 because historical applications depend on *echo* to echo *all* of its arguments, except for the *-n* option in the BSD version.
- 12717
- 12718 **FUTURE DIRECTIONS**
- 12719 None.
- 12720 **SEE ALSO**
- 12721 *printf*
- 12722 **CHANGE HISTORY**
- 12723 First released in Issue 2.
- 12724 **Issue 4**
- 12725 Aligned with the ISO/IEC 9945-2:1993 standard.
- 12726 **Issue 5**
- 12727 In the OPTIONS section, the last sentence is changed to indicate that implementations “do not” support any options; in the previous issue this said “need not”.
- 12728
- 12729 **Issue 6**
- 12730 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:
- 12731
- 12732
 - A set of character sequences is defined as *string* operands.
- 12733
 - *LC_CTYPE* is added to the list of environment variables affecting *echo*.
- 12734
 - In the OPTIONS section, implementations shall not support any options.

12735 NAME

12736 ed — edit text

12737 SYNOPSIS

12738 ed [-p *string*][-s][*file*]

12739 DESCRIPTION

12740 The *ed* utility is a line-oriented text editor that uses two modes: *command mode* and *input mode*.
 12741 In command mode the input characters shall be interpreted as commands, and in input mode
 12742 they shall be interpreted as text. See the EXTENDED DESCRIPTION section.

12743 OPTIONS

12744 The *ed* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
 12745 Utility Syntax Guidelines.

12746 The following options shall be supported:

12747 **-p *string*** Use *string* as the prompt string when in command mode. By default, there shall be
 12748 no prompt string.

12749 **-s** Suppress the writing of byte counts by **e**, **E**, **r**, and **w** commands and of the **'!'**
 12750 prompt after a *!command*.

12751 OPERANDS

12752 The following operand shall be supported:

12753 ***file*** If the *file* argument is given, *ed* shall simulate an **e** command on the file named by
 12754 the path name, *file*, before accepting commands from the standard input. If the *file*
 12755 operand is **'-'**, the results are unspecified.

12756 STDIN

12757 The standard input shall be a text file consisting of commands, as described in the EXTENDED
 12758 DESCRIPTION section.

12759 INPUT FILES

12760 The input files shall be text files.

12761 ENVIRONMENT VARIABLES

12762 The following environment variables shall affect the execution of *ed*:

12763 **HOME** Determine the path name of the user's home directory.

12764 **LANG** Provide a default value for the internationalization variables that are unset or null.
 12765 If **LANG** is unset or null, the corresponding value from the implementation-
 12766 defined default locale shall be used. If any of the internationalization variables
 12767 contains an invalid setting, the utility shall behave as if none of the variables had
 12768 been defined.

12769 **LC_ALL** If set to a non-empty string value, override the values of all the other
 12770 internationalization variables.

12771 **LC_COLLATE**

12772 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 12773 character collating elements within regular expressions.

12774 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 12775 characters (for example, single-byte as opposed to multi-byte characters in
 12776 arguments and input files) and the behavior of character classes within regular
 12777 expressions.

- 12778 **LC_MESSAGES**
- 12779 Determine the locale that should be used to affect the format and contents of
- 12780 diagnostic messages written to standard error and informative messages written to
- 12781 standard output.
- 12782 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 12783 **ASYNCHRONOUS EVENTS**
- 12784 The *ed* utility shall take the standard action for all signals (see the ASYNCHRONOUS EVENTS
- 12785 section in Section 1.11 (on page 2224)) with the following exceptions:
- 12786 **SIGINT** The *ed* utility shall interrupt its current activity, write the string "?\n" to standard
- 12787 output, and return to command mode (see the EXTENDED DESCRIPTION
- 12788 section).
- 12789 **SIGHUP** If the buffer is not empty and has changed since the last write, the *ed* utility shall
- 12790 attempt to write a copy of the buffer in a file. First, the file named **ed.hup** in the
- 12791 current directory shall be used; if that fails, the file named **ed.hup** in the directory
- 12792 named by the *HOME* environment variable shall be used. In any case, the *ed* utility
- 12793 shall exit without returning to command mode.
- 12794 **SIGQUIT** The *ed* utility shall ignore this event.
- 12795 **STDOUT**
- 12796 Various editing commands and the prompting feature (see **-p**) write to standard output, as
- 12797 described in the EXTENDED DESCRIPTION section.
- 12798 **STDERR**
- 12799 Used only for diagnostic messages.
- 12800 **OUTPUT FILES**
- 12801 The output files shall be text files whose formats are dependent on the editing commands given.
- 12802 **EXTENDED DESCRIPTION**
- 12803 The *ed* utility shall operate on a copy of the file it is editing; changes made to the copy shall have
- 12804 no effect on the file until a **w** (write) command is given. The copy of the text is called the *buffer*.
- 12805 Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a
- 12806 single-character *command*, possibly followed by parameters to that command. These addresses
- 12807 specify one or more lines in the buffer. Every command that requires addresses has default
- 12808 addresses, so that the addresses very often can be omitted. If the **-p** option is specified, the
- 12809 prompt string shall be written to standard output before each command is read.
- 12810 In general, only one command can appear on a line. Certain commands allow text to be input.
- 12811 This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be
- 12812 in *input mode*. In this mode, no commands shall be recognized; all input is merely collected.
- 12813 Input mode is terminated by entering a line consisting of two characters: a period ('.')
- 12814 followed by a <newline> character. This line is not considered part of the input text.
- 12815 **Regular Expressions in ed**
- 12816 The *ed* utility shall support basic regular expressions, as described in the Base Definitions
- 12817 volume of IEEE Std. 1003.1-200x, Section 9.3, Basic Regular Expressions. Since regular
- 12818 expressions in *ed* are always matched against single lines, never against any larger section of
- 12819 text, there is no way for a regular expression to match a <newline> character. A null RE shall be
- 12820 equivalent to the last RE encountered.
- 12821 Regular expressions are used in addresses to specify lines, and in some commands (for example,
- 12822 the **s** substitute command) to specify portions of a line to be substituted.

12823

Addresses in ed

12824

Addressing in *ed* relates to the current line. Generally, the current line is the last line affected by a command. The current line number is the address of the current line. If the edit buffer is not empty, the initial value for the current line shall be the last line in the edit buffer; otherwise, zero.

12825

12826

12827

Addresses shall be constructed as follows:

12828

1. The period character (`' . '`) shall address the current line.

12829

2. The dollar sign character (`' $ '`) shall address the last line of the edit buffer.

12830

3. The positive decimal number *n* shall address the *n*th line of the edit buffer.

12831

4. The apostrophe-x character pair (`" ' x "`) shall address the line marked with the mark name character *x*, which shall be a lowercase letter from the portable character set. It shall be an error if the character has not been set to mark a line or if the line that was marked is not currently present in the edit buffer.

12832

12833

12834

12835

5. A BRE enclosed by slash characters (`' / '`) shall address the first line found by searching forwards from the line following the current line toward the end of the edit buffer and stopping at the first line containing a string matching the BRE. The BRE consisting of a null BRE delimited by a pair of slash characters shall address the next line containing the last BRE encountered. In addition, the second slash can be omitted at the end of a command line. Within the BRE, a backslash-slash pair (`" \ / "`) shall represent a literal slash instead of the BRE delimiter. If necessary, the search shall wrap around to the beginning of the buffer and continue up to and including the current line, so that the entire buffer is searched.

12836

12837

12838

12839

12840

12841

12842

12843

6. A BRE enclosed by question-mark characters (`' ? '`) shall address the first line found by searching backwards from the line preceding the current line toward the beginning of the edit buffer and stopping at the first line containing a string matching the BRE. The BRE consisting of a null BRE delimited by a pair of question-mark characters (`" ?? "`) shall address the previous line containing the last BRE encountered. In addition, the second question-mark can be omitted at the end of a command line. Within the BRE, a backslash-question-mark pair (`" \ ? "`) shall represent a literal question mark instead of the BRE delimiter. If necessary, the search shall wrap around to the end of the buffer and continue up to and including the current line, so that the entire buffer is searched.

12844

12845

12846

12847

12848

12849

12850

12851

12852

7. A plus-sign (`' + '`) or hyphen character (`' - '`) followed by a decimal number shall address the current line plus or minus the number. A plus-sign or hyphen character not followed by a decimal number shall address the current line plus or minus 1.

12853

12854

12855

Addresses can be followed by zero or more address offsets, optionally <blank>-separated.

12856

Address offsets are constructed as follows:

12857

- A plus-sign or hyphen character followed by a decimal number shall add or subtract, respectively, the indicated number of lines to or from the address. A plus-sign or hyphen character not followed by a decimal number shall add or subtract 1 to or from the address.

12858

12859

12860

- A decimal number shall add the indicated number of lines to the address.

12861

It shall not be an error for an intermediate address value to be less than zero or greater than the last line in the edit buffer. It shall be an error for the final address value to be less than zero or greater than the last line in the edit buffer. It shall be an error if a search for a BRE fails to find a matching line.

12862

12863

12864

12865

Commands accept zero, one, or two addresses. If more than the required number of addresses are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than the required number of addresses are provided to a command, the addresses specified first

12866

12867

12868 shall be evaluated and then discarded until the maximum number of valid addresses remain, for
12869 the specified command.

12870 Addresses shall be separated from each other by a comma (',') or semicolon character (';').
12871 In the case of a semicolon separator, the current line ('.') shall be set to the first address, and
12872 only then will the second address be calculated. This feature can be used to determine the
12873 starting line for forwards and backwards searches; see rules 5. and 6.

12874 Addresses can be omitted on either side of the comma or semicolon separator, in which case the
12875 resulting address pairs shall be as follows:

12876

Specified	Resulting
,	1 , \$
, addr	1 ,a ddr
addr ,	addr , addr
;	. ; \$
; addr	. ; addr
addr ;	addr ; addr

12877

12878

12879

12880

12881

12882

12883 Any <blank> characters included between addresses, address separators, or address offsets shall
12884 be ignored.

12885

Commands in ed

12886 In the following list of *ed* commands, the default addresses are shown in parentheses. The
12887 number of addresses shown in the default shall be the number expected by the command. The
12888 parentheses are not part of the address; they show that the given addresses are the default.

12889 It is generally invalid for more than one command to appear on a line. However, any command
12890 (except **e**, **E**, **f**, **q**, **Q**, **r**, **w**, and **!**) can be suffixed by the letter **l**, **n**, or **p**; in which case, except for
12891 the **l**, **n**, and **p** commands, the command shall be executed and then the new current line shall be
12892 written as described below under the **l**, **n**, and **p** commands. When an **l**, **n**, or **p** suffix is used
12893 with an **l**, **n**, or **p** command, the command shall write to standard output as described below, but
12894 it is unspecified whether the suffix writes the current line again in the requested format or
12895 whether the suffix has no effect. For example, the **pl** command (base **p** command with an **l**
12896 suffix) shall either write just the current line or write it twice—once as specified for **p** and once
12897 as specified for **l**. Also, the **g**, **G**, **v**, and **V** commands shall take a command as a parameter.

12898 Each address component can be preceded by zero or more <blank> characters. The command
12899 letter can be preceded by zero or more <blank> characters. If a suffix letter (**l**, **n**, or **p**) is given,
12900 the application shall ensure that it immediately follows the command.

12901 The **e**, **E**, **f**, **r**, and **w** commands shall take an optional *file* parameter, separated from the
12902 command letter by one or more <blank> characters.

12903 If changes have been made in the buffer since the last **w** command that wrote the entire buffer,
12904 *ed* shall warn the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands.
12905 The *ed* utility shall write the string:

12906 "?\n"

12907 (followed by an explanatory message if *help mode* has been enabled via the **H** command) to
12908 standard output and shall continue in command mode with the current line number unchanged.
12909 If the **e** or **q** command is repeated with no intervening command, it shall take effect.

12910 If a terminal disconnect is detected:

12911 • If the buffer is not empty and has changed since the last write, the *ed* utility shall attempt to
 12912 write a copy of the buffer to a file named **ed.hup** in the current directory. If this write fails, *ed*
 12913 shall attempt to write a copy of the buffer to a file name **ed.hup** in the directory named by the
 12914 *HOME* environment variable. If both these attempts fail, *ed* shall exit without saving the
 12915 buffer.

12916 • The *ed* utility shall not write the file to the currently remembered path name or return to
 12917 command mode, and shall terminate with a non-zero exit status.

12918 If an end-of-file is detected on standard input:

12919 • If the *ed* utility is in input mode, *ed* shall terminate input mode and return to command mode.
 12920 It is unspecified if any partially entered lines (that is, input text without a terminating
 12921 <newline> character) are discarded from the input text.

12922 • If the *ed* utility is in command mode, it shall act as if a **q** command had been entered.

12923 If the closing delimiter of an RE or of a replacement string (for example, ' / ') in a **g**, **G**, **s**, **v**, or **V**
 12924 command would be the last character before a <newline> character, that delimiter can be
 12925 omitted, in which case the addressed line shall be written. For example, the following pairs of
 12926 commands are equivalent:

```
12927 s/s1/s2    s/s1/s2/p
12928 g/s1       g/s1/p
12929 ?s1       ?s1?
```

12930 If an invalid command is entered, *ed* shall write the string:

12931 "?\n"

12932 (followed by an explanatory message if *help mode* has been enabled via the **H** command) to
 12933 standard output and shall continue in command mode with the current line number unchanged.

12934 Append Command

12935 *Synopsis:* (.)a
 12936 <text>
 12937 .

12938 The **a** command shall read the given text and append it after the addressed line; the current line
 12939 number shall become the address of the last inserted line or, if there were none, the addressed
 12940 line. Address 0 shall be valid for this command; it shall cause the appended text to be placed at
 12941 the beginning of the buffer.

12942 Change Command

12943 *Synopsis:* (. , .)c
 12944 <text>
 12945 .

12946 The **c** command shall delete the addressed lines, then accept input text that replaces these lines;
 12947 the current line shall be set to the address of the last line input; or, if there were none, at the line
 12948 after the last line deleted; if the lines deleted were originally at the end of the buffer, the current
 12949 line number shall be set to the address of the new last line; if no lines remain in the buffer, the
 12950 current line number shall be set to zero. Address 0 shall be valid for this command; it shall be
 12951 interpreted as if address 1 were specified.

12952 **Delete Command**12953 *Synopsis:* (.,.)d

12954 The **d** command shall delete the addressed lines from the buffer. The address of the line after the
 12955 last line deleted shall become the current line number; if the lines deleted were originally at the
 12956 end of the buffer, the current line number shall be set to the address of the new last line; if no
 12957 lines remain in the buffer, the current line number shall be set to zero.

12958 **Edit Command**12959 *Synopsis:* e [*file*]

12960 The **e** command shall delete the entire contents of the buffer and then read in the file named by
 12961 the path name *file*. The current line number shall be set to the address of the last line of the
 12962 buffer. If no path name is given, the currently remembered path name, if any, shall be used (see
 12963 the **f** command). The number of bytes read shall be written to standard output, unless the **-s**
 12964 option was specified, in the following format:

12965 "%d\n", <number of bytes read>

12966 The name *file* shall be remembered for possible use as a default path name in subsequent **e**, **E**, **r**,
 12967 and **w** commands. If *file* is replaced by **'!'**, the rest of the line shall be taken to be a shell
 12968 command line whose output is to be read. Such a shell command line shall not be remembered
 12969 as the current *file*. All marks shall be discarded upon the completion of a successful **e** command.
 12970 If the buffer has changed since the last time the entire buffer was written, the user shall be
 12971 warned, as described previously.

12972 **Edit Without Checking Command**12973 *Synopsis:* E [*file*]

12974 The **E** command shall possess all properties and restrictions of the **e** command except that the
 12975 editor shall not check to see whether any changes have been made to the buffer since the last **w**
 12976 command.

12977 **File Name Command**12978 *Synopsis:* f [*file*]

12979 If *file* is given, the **f** command shall change the currently remembered path name to *file*; whether
 12980 the name is changed or not, it shall then write the (possibly new) currently remembered path
 12981 name to the standard output in the following format:

12982 "%s\n", <pathname>

12983 The current line number shall be unchanged.

12984 **Global Command**12985 *Synopsis:* (1,\$)g/RE/command list

12986 In the **g** command, the first step shall be to mark every line that matches the given *RE*. Then,
 12987 going sequentially from the beginning of the file to the end of the file, the given *command list*
 12988 shall be executed for each marked line, with the current line number set to the address of that
 12989 line. Any line modified by the *command list* shall be unmarked. When the **g** command completes,
 12990 the current line number shall have the value assigned by the last command in the *command list*.
 12991 If there were no matching lines, the current line number shall not be changed. A single command
 12992 or the first of a list of commands shall appear on the same line as the global command. All lines

12993 of a multi-line list except the last line shall be ended with a backslash; the **a**, **i**, and **c** commands
 12994 and associated input are permitted. The `'.'` terminating input mode can be omitted if it would
 12995 be the last line of the *command list*. An empty *command list* shall be equivalent to the **p** command.
 12996 The use of the **g**, **G**, **v**, **V**, and **!** commands in the *command list* produces undefined results. Any
 12997 character other than `<space>` or `<newline>` can be used instead of a slash to delimit the *RE*.
 12998 Within the *RE*, the *RE* delimiter itself can be used as a literal character if it is preceded by a
 12999 backslash.

13000 **Interactive Global Command**

13001 *Synopsis:* `(1 , $)G/RE/`

13002 In the **G** command, the first step shall be to mark every line that matches the given *RE*. Then,
 13003 for every such line, that line shall be written, the current line number shall be set to the address
 13004 of that line, and any one command (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) shall
 13005 be read and executed. A `<newline>` character shall act as a null command (causing no action to
 13006 be taken on the current line); an `'&'` shall cause the re-execution of the most recent non-null
 13007 command executed within the current invocation of **G**. Note that the commands input as part
 13008 of the execution of the **G** command can address and affect any lines in the buffer. The final value
 13009 of the current line number shall be the value set by the last command successfully executed.
 13010 (Note that the last command successfully executed shall be the **G** command itself if a command
 13011 fails or the null command is specified.) If there were no matching lines, the current line number
 13012 shall not be changed. The **G** command can be terminated by a SIGINT signal. Any character
 13013 other than `<space>` or `<newline>` can be used instead of a slash to delimit the *RE* and the
 13014 replacement. Within the *RE*, the *RE* delimiter itself can be used as a literal character if it is
 13015 preceded by a backslash.

13016 **Help Command**

13017 *Synopsis:* `h`

13018 The **h** command shall write a short message to standard output that explains the reason for the
 13019 most recent `'?'` notification. The current line number shall be unchanged.

13020 **Help-Mode Command**

13021 *Synopsis:* `H`

13022 The **H** command shall cause *ed* to enter a mode in which help messages (see the **h** command)
 13023 shall be written to standard output for all subsequent `'?'` notifications. The **H** command
 13024 alternatively shall turn this mode on and off; it is initially off. If the help-mode is being turned
 13025 on, the **H** command also explains the previous `'?'` notification, if there was one. The current
 13026 line number shall be unchanged.

13027 **Insert Command**

13028 *Synopsis:* `(.)i`
 13029 `<text>`
 13030 `.`

13031 The **i** command shall insert the given text before the addressed line; the current line is set to the
 13032 last inserted line or, if there was none, to the addressed line. This command differs from the **a**
 13033 command only in the placement of the input text. Address 0 shall be valid for this command; it
 13034 shall be interpreted as if address 1 were specified.

13035 **Join Command**13036 *Synopsis:* (. , .+1) j

13037 The **j** command shall join contiguous lines by removing the appropriate <newline> characters. If
 13038 exactly one address is given, this command shall do nothing. If lines are joined, the current line
 13039 number shall be set to the address of the joined line; otherwise, the current line number shall be
 13040 unchanged.

13041 **Mark Command**13042 *Synopsis:* (.) k x

13043 The **k** command shall mark the addressed line with name *x*, which the application shall ensure is
 13044 a lowercase letter from the portable character set. The address " *x* " shall then refer to this line;
 13045 the current line number shall be unchanged.

13046 **List Command**13047 *Synopsis:* (. , .) l

13048 The **l** command shall write to standard output the addressed lines in a visually unambiguous
 13049 form. The characters listed in the Base Definitions volume of IEEE Std. 1003.1-200x, Table 5-1,
 13050 Escape Sequences and Associated Actions ('\', '\a', '\b', '\f', '\r', '\t', '\v') shall
 13051 be written as the corresponding escape sequence; the '\n' in that table is not applicable. Non-
 13052 printable characters not in the table shall be written as one three-digit octal number (with a
 13053 preceding backslash character) for each byte in the character (most significant byte first). If the
 13054 size of a byte on the system is greater than nine bits, the format used for non-printable characters
 13055 is implementation-defined.

13056 Long lines shall be folded, with the point of folding indicated by writing backslash/<newline>
 13057 character; the length at which folding occurs is unspecified, but should be appropriate for the
 13058 output device. The end of each line shall be marked with a '\$', and '\$' characters within the
 13059 text shall be written with a preceding backslash. An **l** command can be appended to any other
 13060 command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**. The current line number shall be set to the address of
 13061 the last line written.

13062 **Move Command**13063 *Synopsis:* (. , .) address

13064 The **m** command shall reposition the addressed lines after the line addressed by *address*.
 13065 Address 0 shall be valid for *address* and cause the addressed lines to be moved to the beginning
 13066 of the buffer. It shall be an error if *address* falls within the range of moved lines. The
 13067 current line number shall be set to the address of the last line moved.

13068 **Number Command**13069 *Synopsis:* (. , .) n

13070 The **n** command shall write to standard output the addressed lines, preceding each line by its
 13071 line number and a <tab> character; the current line number shall be set to the address of the last
 13072 line written. The **n** command can be appended to any command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

13073 **Print Command**13074 *Synopsis:* (.,.)p

13075 The **p** command shall write to standard output the addressed lines; the current line number shall
 13076 be set to the address of the last line written. The **p** command can be appended to any command
 13077 other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

13078 **Prompt Command**13079 *Synopsis:* P

13080 The **P** command shall cause *ed* to prompt with an asterisk ('*') (or *string*, if **-p** is specified) for
 13081 all subsequent commands. The **P** command alternatively shall turn this mode on and off; it shall
 13082 be initially on if the **-p** option is specified; otherwise, off. The current line number shall be
 13083 unchanged.

13084 **Quit Command**13085 *Synopsis:* q

13086 The **q** command shall cause *ed* to exit. If the buffer has changed since the last time the entire
 13087 buffer was written, the user shall be warned, as described previously.

13088 **Quit Without Checking Command**13089 *Synopsis:* Q

13090 The **Q** command shall cause *ed* to exit without checking whether changes have been made in the
 13091 buffer since the last **w** command.

13092 **Read Command**13093 *Synopsis:* (\$)r [*file*]

13094 The **r** command shall read in the file named by the path name *file* and append it after the
 13095 addressed line. If no *file* argument is given, the currently remembered path name, if any, shall be
 13096 used (see the **e** and **f** commands). The currently remembered path name shall not be changed
 13097 unless there is no remembered path name. Address 0 shall be valid for **r** and shall cause the file
 13098 to be read at the beginning of the buffer. If the read is successful, and **-s** was not specified, the
 13099 number of bytes read shall be written to standard output in the following format:

13100 "%d\n", <number of bytes read>

13101 The current line number shall be set to the address of the last line read in. If *file* is replaced by
 13102 '!', the rest of the line shall be taken to be a shell command line whose output is to be read.
 13103 Such a shell command line shall not be remembered as the current path name.

13104 **Substitute Command**13105 *Synopsis:* (.,.)s/RE/replacement/flags

13106 The **s** command shall search each addressed line for an occurrence of the specified *RE* and
 13107 replace either the first or all (non-overlapped) matched strings with the *replacement*; see the
 13108 following description of the **g** suffix. It is an error if the substitution fails on every addressed
 13109 line. Any character other than <space> or <newline> can be used instead of a slash to delimit the
 13110 *RE* and the replacement. Within the *RE*, the *RE* delimiter itself can be used as a literal character if
 13111 it is preceded by a backslash. The current line shall be set to the address of the last line on which
 13112 a substitution occurred.

13113 An ampersand ('&') appearing in the *replacement* shall be replaced by the string matching the
 13114 RE on the current line. The special meaning of '&' in this context can be suppressed by
 13115 preceding it by backslash. As a more general feature, the characters '\n', where *n* is a digit,
 13116 shall be replaced by the text matched by the corresponding back-reference expression. When the
 13117 character '%' is the only character in the *replacement*, the *replacement* used in the most recent
 13118 substitute command shall be used as the *replacement* in the current substitute command; if there
 13119 was no previous substitute command, the use of '%' in this manner shall be an error. The '%'
 13120 shall lose its special meaning when it is in a replacement string of more than one character or is
 13121 preceded by a backslash. For each backslash ('\') encountered in scanning *replacement* from
 13122 beginning to end, the following character shall lose its special meaning (if any). It is unspecified
 13123 what special meaning is given to any character other than '&', '\', '%', or digits.

13124 A line can be split by substituting a <newline> character into it. The application shall ensure it
 13125 escapes the <newline> character in the *replacement* by preceding it by backslash. Such
 13126 substitution cannot be done as part of a **g** or **v** *command list*. The current line number shall be set
 13127 to the address of the last line on which a substitution is performed. If no substitution is
 13128 performed, the current line number shall be unchanged. If a line is split, a substitution shall be
 13129 considered to have been performed on each of the new lines for the purpose of determining the
 13130 new current line number. A substitution shall be considered to have been performed even if the
 13131 replacement string is identical to the string that it replaces.

13132 The application shall ensure that the value of *flags* is zero or more of:

13133 **count** Substitute for the *count*th occurrence only of the *RE* found on each addressed line.

13134 **g** Globally substitute for all non-overlapping instances of the *RE* rather than just the first
 13135 one. If both **g** and *count* are specified, the results are unspecified.

13136 **l** Write to standard output the final line in which a substitution was made. The line shall
 13137 be written in the format specified for the **l** command.

13138 **n** Write to standard output the final line in which a substitution was made. The line shall
 13139 be written in the format specified for the **n** command.

13140 **p** Write to standard output the final line in which a substitution was made. The line shall
 13141 be written in the format specified for the **p** command.

13142 Copy Command

13143 *Synopsis:* (.,.)*taddress*

13144 The **t** command shall be equivalent to the **m** command, except that a copy of the addressed lines
 13145 shall be placed after address *address* (which can be 0); the current line number shall be set to the
 13146 address of the last line added.

13147 Undo Command

13148 *Synopsis:* u

13149 The **u** command shall nullify the effect of the most recent command that modified anything in
 13150 the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **u**, **v**, **G**, or **V** command. All changes
 13151 made to the buffer by a **g**, **G**, **v**, or **V** global command shall be undone as a single change; if no
 13152 changes were made by the global command (such as with **g/RE/p**), the **u** command shall have
 13153 no effect. The current line number shall be set to the value it had immediately before the
 13154 command being undone started.

13155 **Global Non-Matched Command**13156 *Synopsis:* (1,\$)v/RE/command list13157 This command shall be equivalent to the global command **g** except that the lines that are marked
13158 during the first step shall be those that do not match the *RE*.13159 **Interactive Global Not-Matched Command**13160 *Synopsis:* (1,\$)V/RE/13161 This command shall be equivalent to the interactive global command **G** except that the lines that
13162 are marked during the first step shall be those that do not match the *RE*.13163 **Write Command**13164 *Synopsis:* (1,\$)w [file]13165 The **w** command shall write the addressed lines into the file named by the path name *file*. The
13166 command shall create the file, if it does not exist, or shall replace the contents of the existing file.
13167 The currently remembered path name shall not be changed unless there is no remembered path
13168 name. If no path name is given, the currently remembered path name, if any, shall be used (see
13169 the **e** and **f** commands); the current line number shall be unchanged. If the command is
13170 successful, the number of bytes written shall be written to standard output, unless the **-s** option
13171 was specified, in the following format:

13172 "%d\n", <number of bytes written>

13173 If *file* begins with **'!'**, the rest of the line shall be taken to be a shell command line whose
13174 standard input shall be the addressed lines. Such a shell command line shall not be remembered
13175 as the current path name. This usage of the write command with **'!'** shall not be considered as
13176 a "last **w** command that wrote the entire buffer", as described previously; thus, this alone shall
13177 not prevent the warning to the user if an attempt is made to destroy the editor buffer via the **e** or
13178 **q** commands.13179 **Line Number Command**13180 *Synopsis:* (\$)=13181 The line number of the addressed line shall be written to standard output in the following
13182 format:

13183 "%d\n", <line number>

13184 The current line number shall be unchanged by this command.

13185 **Shell Escape Command**13186 *Synopsis:* !command13187 The remainder of the line after the **'!'** shall be sent to the command interpreter to be
13188 interpreted as a shell command line. Within the text of that shell command line, the unescaped
13189 character **'%'** shall be replaced with the remembered path name; if a **'!'** appears as the first
13190 character of the command, it shall be replaced with the text of the previous shell command
13191 executed via **'!'**. Thus, **"!!"** shall repeat the previous *!command*. If any replacements of **'%'** or
13192 **'!'** are performed, the modified line shall be written to the standard output before *command* is
13193 executed. The **'!'** command shall write:

13194 "!\n"

13195 to standard output upon completion, unless the `-s` option is specified. The current line number
13196 shall be unchanged.

13197 **Null Command**

13198 *Synopsis:* (`+.1`)

13199 An address alone on a line shall cause the addressed line to be written. A `<newline>` character
13200 alone shall be equivalent to `"+.1p"`. The current line number shall be set to the address of the
13201 written line.

13202 **EXIT STATUS**

13203 The following exit values shall be returned:

13204 0 Successful completion without any file or command errors.

13205 >0 An error occurred.

13206 **CONSEQUENCES OF ERRORS**

13207 When an error in the input script is encountered, or when an error is detected that is a
13208 consequence of the data (not) present in the file or due to an external condition such as a read or
13209 write error:

13210 • If the standard input is a terminal device file, all input shall be flushed, and a new command
13211 read.

13212 • If the standard input is a regular file, *ed* shall terminate with a non-zero exit status.

13213 **APPLICATION USAGE**

13214 Because of the extremely terse nature of the default error messages, the prudent script writer
13215 begins the *ed* input commands with an **H** command, so that if any errors do occur at least some
13216 clue as to the cause is made available.

13217 In previous versions, an obsolescent `-` option was described. This is no longer specified.
13218 Applications should use the `-s` option. Using `-` as a *file* operand now produces unspecified
13219 results. This allows implementations to continue to support the former required behavior.

13220 **EXAMPLES**

13221 None.

13222 **RATIONALE**

13223 The initial description of this utility was adapted from the SVID. It contains some features not
13224 found in Version 7 or BSD-derived systems. Some of the differences between the POSIX and
13225 BSD *ed* utilities include, but need not be limited to:

13226 • The BSD `-` option does not suppress the `'!'` prompt after a `!` command.

13227 • BSD does not support the special meanings of the `'%'` and `'!'` characters within a `!`
13228 command.

13229 • BSD does not support the *addresses* `' ; '` and `' , '`.

13230 • BSD allows the command/suffix pairs **pp**, **ll**, and so on, which are unspecified in this volume
13231 of IEEE Std. 1003.1-200x.

13232 • BSD does not support the `'!'` character part of the **e**, **r**, or **w** commands.

13233 • A failed **g** command in BSD sets the line number to the last line searched if there are no
13234 matches.

13235 • BSD does not default the *command list* to the **p** command.

- 13236 • BSD does not support the **G**, **h**, **H**, **n**, or **V** commands.
- 13237 • On BSD, if there is no inserted text, the insert command changes the current line to the
- 13238 referenced line -1 ; that is, the line before the specified line.
- 13239 • On BSD, the *join* command with only a single address changes the current line to that
- 13240 address.
- 13241 • BSD does not support the **P** command; moreover, in BSD it is synonymous with the **p**
- 13242 command.
- 13243 • BSD does not support the *undo* of the commands **j**, **m**, **r**, **s**, or **t**.
- 13244 • The Version 7 *ed* command **W**, and the BSD *ed* commands **W**, **wq**, and **z** are not present in this
- 13245 volume of IEEE Std. 1003.1-200x.
- 13246 The $-s$ option was added to allow the functionality of the now withdrawn $-$ option in a manner
- 13247 compatible with the Utility Syntax Guidelines.
- 13248 In early proposals there was a limit, {ED_FILE_MAX}, that described the historical limitations of
- 13249 some *ed* utilities in their handling of large files; some of these have had problems with files larger
- 13250 than 100 000 bytes. It was this limitation that prompted much of the desire to include a *split*
- 13251 command in this volume of IEEE Std. 1003.1-200x. Since this limit was removed, this volume of
- 13252 IEEE Std. 1003.1-200x requires that implementations document the file size limits imposed by *ed*
- 13253 in the conformance document. The limit {ED_LINE_MAX} was also removed; therefore, the
- 13254 global limit {LINE_MAX} is used for input and output lines.
- 13255 The manner in which the **l** command writes non-printable characters was changed to avoid the
- 13256 historical backspace-overstrike method. On video display terminals, the overstrike is ambiguous
- 13257 because most terminals simply replace overstruck characters, making the **l** format not useful for
- 13258 its intended purpose of unambiguously understanding the content of the line. The historical
- 13259 backslash escapes were also ambiguous. (The string "a\0011" could represent a line containing
- 13260 those six characters or a line containing the three characters 'a', a byte with a binary value of 1,
- 13261 and a 1.) In the format required here, a backslash appearing in the line is written as "\\ " so that
- 13262 the output is truly unambiguous. The method of marking the ends of lines was adopted from the
- 13263 *ex* editor and is required for any line ending in <space>; the '\$' is placed on all lines so that a
- 13264 real '\$' at the end of a line cannot be misinterpreted.
- 13265 Systems with bytes too large to fit into three octal digits must devise other means of displaying
- 13266 non-printable characters. Consideration was given to requiring that the number of octal digits be
- 13267 large enough to hold a byte, but this seemed to be too confusing for applications on the vast
- 13268 majority of systems where three digits are adequate. It would be theoretically possible for the
- 13269 application to use the *getconf* utility to find out the CHAR_BIT value and deal with such an
- 13270 algorithm; however, there is really no portable way that an application can use the octal values
- 13271 of the bytes across various coded character sets, so the additional specification was not
- 13272 worthwhile.
- 13273 The description of how a NUL is written was removed. The NUL character cannot be in text
- 13274 files, and this volume of IEEE Std. 1003.1-200x should not dictate behavior in the case of
- 13275 undefined, erroneous input.
- 13276 Unlike some of the other editing utilities, the file names accepted by the **E**, **e**, **R**, and **r** commands
- 13277 are not patterns.
- 13278 Early proposals stated that the $-p$ option worked only when standard input was associated with
- 13279 a terminal device. This has been changed to conform to historical implementations, thereby
- 13280 allowing applications to interpose themselves between a user and the *ed* utility.

- 13281 The form of the substitute command that uses the **n** suffix was limited in some historical
13282 documentation (where this was described incorrectly as “backreferencing”). This limit has been
13283 omitted because there is no reason an editor processing lines of {LINE_MAX} length should have
13284 this restriction. The command **s/x/X/2 047** should be able to substitute the 2 047th occurrence of **x**
13285 on a line.
- 13286 The use of printing commands with printing suffixes (such as **pn**, **lp**, and so on) was made
13287 unspecified because BSD-based systems allow this, whereas System V does not.
- 13288 Some BSD-based systems exit immediately upon receipt of end-of-file if all of the lines in the file
13289 have been deleted. Since this volume of IEEE Std. 1003.1-200x refers to the **q** command in this
13290 instance, such behavior is not allowed.
- 13291 Some historical implementations returned exit status zero even if command errors had occurred;
13292 this is not allowed by this volume of IEEE Std. 1003.1-200x.
- 13293 Some historical implementations contained a bug that allowed a single period to be entered in
13294 input mode as <backslash> <period> <newline>. This is not allowed by the *ed* because there is
13295 no description of escaping any of the characters in input mode; backslashes are entered into the
13296 buffer exactly as typed. The typical method of entering a single period has been to precede it
13297 with another character and then use the substitute command to delete that character.
- 13298 It is difficult under some modes of some versions of historical operating system terminal drivers
13299 to distinguish between an end-of-file condition and terminal disconnect. The ISO POSIX-2
13300 standard does not require implementations to distinguish between the two situations, which
13301 permits historical implementations of the *ed* utility on historical platforms to conform.
13302 Implementations are encouraged to distinguish between the two, if possible, and take
13303 appropriate action on terminal disconnect.
- 13304 Historically, *ed* accepted a zero address for the **a** and **r** commands in order to insert text at the
13305 start of the edit buffer. When the buffer was empty the command **.=** returned zero.
13306 IEEE Std. 1003.1-200x requires conformance to historical practice.
- 13307 For consistency with the **a** and **r** commands and better user functionality, the **i** and **c** commands
13308 must also accept an address of 0, in which case **0i** is treated as **1i** and likewise for the **c**
13309 command.
- 13310 All of the following are valid addresses:
- 13311 **+++** Three lines after the current line.
- 13312 **/pattern/-** One line before the next occurrence of pattern.
- 13313 **-2** Two lines before the current line.
- 13314 **3 ——— 2** Line one (note the intermediate negative address).
- 13315 **1 2 3** Line six.
- 13316 Any number of addresses can be provided to commands taking addresses; for example,
13317 "**1,2,3,4,5p**" prints lines 4 and 5, because two is the greatest valid number of addresses
13318 accepted by the **print** command. This, in combination with the semicolon delimiter, permits
13319 users to create commands based on ordered patterns in the file. For example, the command
13320 "**3;/foo/;+2p**" will display the first line after line 3 that contains the pattern *foo*, plus the next
13321 two lines. Note that the address "**3;**" must still be evaluated before being discarded, because
13322 the search origin for the **/foo/** command depends on this.
- 13323 Historically, *ed* disallowed address chains, as discussed above, consisting solely of comma or
13324 semicolon separators; for example, "**,, ,**" or "**;; ;**" were considered an error. For consistency of
13325 address specification, this restriction is removed. The following table lists some of the address

13326 forms now possible:

	Address	Addr1	Addr2	Status	Comment
13327	7,	7	7	Historical	
13328	7,5,	5	5	Historical	
13329	7,5,9	5	9	Historical	
13330	7,9	7	9	Historical	
13331	7,+	7	8	Historical	
13332	,	1	\$	Historical	
13333	,7	1	7	Extension	
13334	,,	\$	\$	Extension	
13335	,;	\$	\$	Extension	
13336	7;	7	7	Historical	
13337	7;5;	5	5	Historical	
13338	7;5;9	5	9	Historical	
13339	7;5,9	5	9	Historical	
13340	7;\$;4	\$	4	Historical	Valid, but erroneous.
13341	7;9	7	9	Historical	
13342	7;+	7	8	Historical	
13343	;	.	\$	Historical	
13344	;7	.	7	Extension	
13345	;;	\$	\$	Extension	
13346	;,	\$	\$	Extension	

13348 Historically, values could be added to addresses by including them after one or more <blank>
 13349 characters; for example, "3 - 5p" wrote the seventh line of the file, and "/foo/ 5" was the
 13350 same as "5 /foo/". However, only absolute values could be added; for example, "5 /foo/"
 13351 was an error. IEEE Std. 1003.1-200x requires conformance to historical practice.

13352 Historically, *ed* accepted the '^' character as an address, in which case it was identical to the
 13353 hyphen character. IEEE Std. 1003.1-200x does not require or prohibit this behavior.

13354 FUTURE DIRECTIONS

13355 None.

13356 SEE ALSO

13357 *ex, sed, sh, vi*

13358 CHANGE HISTORY

13359 First released in Issue 2.

13360 Issue 4

13361 Aligned with the ISO/IEC 9945-2:1993 standard.

13362 Issue 5

13363 In the OPTIONS section, the meaning of -s and - is clarified.

13364 Second FUTURE DIRECTION added.

13365 Issue 6

13366 The obsolescent single-minus form has been removed.

13367 A second APPLICATION USAGE note has been added.

13368 The Open Group corrigenda item U025/2 has been applied, correcting the description of the Edit
 13369 section.

- 13370 The *ed* utility is updated to align with the IEEE P1003.2b draft standard. This includes addition of
13371 the treatment of the SIGQUIT signal, changes to *ed* addressing, changes to processing when
13372 end-of-file is detected and when terminal disconnect is detected.
- 13373 The normative text is reworded to avoid use of the term “must” for application requirements.

13374 **NAME**

13375 env — set the environment for command invocation

13376 **SYNOPSIS**

13377 env [-i][name=value]... [utility [argument...]]

13378 **DESCRIPTION**13379 The *env* utility shall obtain the current environment, modify it according to its arguments, then
13380 invoke the utility named by the *utility* operand with the modified environment.13381 Optional arguments shall be passed to *utility*.13382 If no *utility* operand is specified, the resulting environment shall be written to the standard
13383 output, with one *name=value* pair per line.13384 **OPTIONS**13385 The *env* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
13386 12.2, Utility Syntax Guidelines.

13387 The following options shall be supported:

13388 **-i** Invoke *utility* with exactly the environment specified by the arguments; the
13389 inherited environment shall be ignored completely.13390 **OPERANDS**

13391 The following operands shall be supported:

13392 *name=value* Arguments of the form *name=value* shall modify the execution environment, and
13393 shall be placed into the inherited environment before the *utility* is invoked.13394 *utility* The name of the utility to be invoked. If the *utility* operand names any of the
13395 special built-in utilities in Section 2.15 (on page 2276), the results are undefined.13396 *argument* A string to pass as an argument for the invoked utility.13397 **STDIN**

13398 Not used.

13399 **INPUT FILES**

13400 None.

13401 **ENVIRONMENT VARIABLES**13402 The following environment variables shall affect the execution of *env*:13403 *LANG* Provide a default value for the internationalization variables that are unset or null.
13404 If *LANG* is unset or null, the corresponding value from the implementation-
13405 defined default locale shall be used. If any of the internationalization variables
13406 contains an invalid setting, the utility shall behave as if none of the variables had
13407 been defined.13408 *LC_ALL* If set to a non-empty string value, override the values of all the other
13409 internationalization variables.13410 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
13411 characters (for example, single-byte as opposed to multi-byte characters in
13412 arguments).13413 *LC_MESSAGES*13414 Determine the locale that should be used to affect the format and contents of
13415 diagnostic messages written to standard error.

- 13416 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 13417 **PATH** Determine the location of the *utility*, as described in the Base Definitions volume of
 13418 IEEE Std. 1003.1-200x, Chapter 8, Environment Variables. If *PATH* is specified as a
 13419 *name=value* operand to *env*, the *value* given shall be used in the search for *utility*.
- 13420 **ASYNCHRONOUS EVENTS**
- 13421 Default.
- 13422 **STDOUT**
- 13423 If no *utility* operand is specified, each *name=value* pair in the resulting environment shall be
 13424 written in the form:
- 13425 "%s=%s\n", <name>, <value>
- 13426 If the *utility* operand is specified, the *env* utility shall not write to standard output.
- 13427 **STDERR**
- 13428 Used only for diagnostic messages.
- 13429 **OUTPUT FILES**
- 13430 None.
- 13431 **EXTENDED DESCRIPTION**
- 13432 None.
- 13433 **EXIT STATUS**
- 13434 If the *utility* utility is invoked, the exit status of *env* shall be the exit status of *utility*; otherwise,
 13435 the *env* utility shall exit with one of the following values:
- 13436 0 The *env* utility completed successfully.
- 13437 1–125 An error occurred in the *env* utility.
- 13438 126 The utility specified by *utility* was found but could not be invoked.
- 13439 127 The utility specified by *utility* could not be found.
- 13440 **CONSEQUENCES OF ERRORS**
- 13441 Default.
- 13442 **APPLICATION USAGE**
- 13443 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if
 13444 an error occurs so that applications can distinguish “failure to find a utility” from “invoked
 13445 utility exited with an error indication”. The value 127 was chosen because it is not commonly
 13446 used for other meanings; most utilities use small values for “normal error conditions” and the
 13447 values above 128 can be confused with termination due to receipt of a signal. The value 126 was
 13448 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
 13449 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
 13450 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
 13451 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
 13452 any other reason.
- 13453 Historical implementations of the *env* utility use the *execvp()* or *execlp()* functions defined in the
 13454 System Interfaces volume of IEEE Std. 1003.1-200x to invoke the specified utility; this provides
 13455 better performance and keeps users from having to escape characters with special meaning to
 13456 the shell. Therefore, shell functions, special built-ins, and built-ins that are only provided by the
 13457 shell are not found.

13458 **EXAMPLES**

13459 The following command:

```
13460           env -i PATH=/mybin mygrep xyz myfile
```

13461 invokes the command *mygrep* with a new *PATH* value as the only entry in its environment. In
13462 this case, *PATH* is used to locate *mygrep*, which then must reside in **/mybin**.

13463 **RATIONALE**

13464 As with all other utilities that invoke other utilities, this volume of IEEE Std. 1003.1-200x only
13465 specifies what *env* does with standard input, standard output, standard error, input files, and
13466 output files. If a utility is executed, it is not constrained by the specification of input and output
13467 by *env*.

13468 The **-i** option was added to allow the functionality of the withdrawn **-** option in a manner
13469 compatible with the Utility Syntax Guidelines.

13470 Some have suggested that *env* is redundant since the same effect is achieved by:

```
13471           name=value ... utility [ argument ... ]
```

13472 The example is equivalent to *env* when an environment variable is being added to the
13473 environment of the command, but not when the environment is being set to the given value.

13474 The *env* utility also writes out the current environment if invoked without arguments. There is
13475 sufficient functionality beyond what the example provides to justify inclusion of *env*.

13476 **FUTURE DIRECTIONS**

13477 None.

13478 **SEE ALSO**

13479 Section 2.5 (on page 2241)

13480 **CHANGE HISTORY**

13481 First released in Issue 2.

13482 **Issue 4**

13483 Aligned with the ISO/IEC 9945-2:1993 standard.

13484 NAME

13485 ex — text editor

13486 SYNOPSIS

13487 UP `ex [-rR][-l][-s | -v][-c command]-t tagstring[-w size][file ...]`

13488

13489 DESCRIPTION

13490 The *ex* utility is a line-oriented text editor. There are two other modes of the editor—open and
 13491 visual—in which screen-oriented editing is available. This is described more fully by the *ex* **open**
 13492 and **visual** commands and in *vi*.

13493 This section uses the term *edit buffer* to describe the current working text. No specific
 13494 implementation is implied by this term. All editing changes are performed on the edit buffer,
 13495 and no changes to it shall affect any file until an editor command writes the file.

13496 Certain terminals do not have all the capabilities necessary to support the complete *ex* definition,
 13497 such as the full-screen editing commands (*visual mode* or *open mode*). When these commands
 13498 cannot be supported on such terminals, this condition shall not produce an error message such
 13499 as “not an editor command” or report a syntax error. The implementation may either accept the
 13500 commands and produce results on the screen that are the result of an unsuccessful attempt to
 13501 meet the requirements of this volume of IEEE Std. 1003.1-200x or report an error describing the
 13502 terminal-related deficiency.

13503 OPTIONS

13504 The *ex* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
 13505 Utility Syntax Guidelines.

13506 The following options shall be supported:

13507 **-c *command*** Specify an initial command to be executed in the first edit buffer loaded from an
 13508 existing file (see the EXTENDED DESCRIPTION section). Implementations may
 13509 support more than a single **-c** option. In such implementations, the specified
 13510 commands shall be executed in the order specified on the command line.

13511 **-l** (The letter ell.) Set lisp mode; indents appropriately for LISP code; the **O**, **{**, **[**, and
 13512 **]** commands in visual mode are modified to have meaning for LISP.

13513 **-r** Recover the named files (see the EXTENDED DESCRIPTION section). Recovery
 13514 information for a file shall be saved during an editor or system crash (for example,
 13515 when the editor is terminated by a signal which the editor can catch), or after the
 13516 use of an *ex* **preserve** command.

13517 A *crash* in this context is an unexpected failure of the system or utility that requires
 13518 restarting the failed system or utility. A system crash implies that any utilities
 13519 running at the time also crash. In the case of an editor or system crash, the number
 13520 of changes to the edit buffer (since the most recent **preserve** command) that will be
 13521 recovered is unspecified.

13522 If no *file* operands are given and the **-t** option is not specified, all other options, the
 13523 *EXINIT* variable, and any *.exrc* files shall be ignored; a list of all recoverable files
 13524 available to the invoking user shall be written, and the editor shall exit normally
 13525 without further action.

13526 **-R** Set **readonly** edit option.

13527 **-s** Prepare *ex* for batch use by taking the following actions:

- 13528 • Suppress writing prompts and informational (but not diagnostic) messages.
- 13529 • Ignore the value of *TERM* and any implementation default terminal type and
- 13530 assume the terminal is a type incapable of supporting open or visual modes;
- 13531 see the **visual** command and the description of *vi*.
- 13532 • Suppress the use of the *EXINIT* environment variable and the reading of any
- 13533 *.exrc* file; see the EXTENDED DESCRIPTION section.
- 13534 • Suppress autoindentation, ignoring the value of the **autoindent** edit option.
- 13535 *-t tagstring* Edit the file containing the specified *tagstring*; see *ctags*. The tags feature
- 13536 represented by *-t tagstring* and the **tag** command is optional. It shall be provided
- 13537 on any system that also provides a conforming implementation of *ctags*; otherwise,
- 13538 the use of *-t* produces undefined results. On any system, it shall be an error to
- 13539 specify more than a single *-t* option.
- 13540 *-v* Begin in visual mode (see *vi*).
- 13541 *-w size* Set the value of the *window* editor option to *size*.

13542 OPERANDS

13543 The following operand shall be supported:

13544 *file* A path name of a file to be edited.

13545 STDIN

13546 The standard input consists of a series of commands and input text, as described in the

13547 EXTENDED DESCRIPTION section. The implementation may limit each line of standard input

13548 to a length of {LINE_MAX}.

13549 If the standard input is not a terminal device, it shall be as if the *-s* option had been specified.

13550 If a read from the standard input returns an error, or if the editor detects an end-of-file condition

13551 from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

13552 INPUT FILES

13553 Input files shall be text files or files that would be text files except for an incomplete last line that

13554 is not longer than {LINE_MAX}-1 bytes in length and contains no NUL characters. By default,

13555 any incomplete last line shall be treated as if it had a trailing <newline> character. The editing of

13556 other forms of files may optionally be allowed by *ex* implementations.

13557 The *.exrc* files and source files shall be text files consisting of *ex* commands; see the EXTENDED

13558 DESCRIPTION section.

13559 By default, the editor shall read lines from the files to be edited without interpreting any of those

13560 lines as any form of editor command.

13561 ENVIRONMENT VARIABLES

13562 The following environment variables shall affect the execution of *ex*:

13563 *COLUMNS* Override the system-selected horizontal screen size. See the Base Definitions |

13564 volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables for valid |

13565 values and results when it is unset or null.

13566 *EXINIT* Determine a list of *ex* commands that are executed on editor start-up. See the

13567 EXTENDED DESCRIPTION section for more details of the initialization phase.

13568 *HOME* Determine a path name of a directory that shall be searched for an editor start-up

13569 file named *.exrc*; see the EXTENDED DESCRIPTION section.

- 13570 *LANG* Provide a default value for the internationalization variables that are unset or null.
 13571 If *LANG* is unset or null, the corresponding value from the implementation-
 13572 defined default locale shall be used. If any of the internationalization variables
 13573 contains an invalid setting, the utility shall behave as if none of the variables had
 13574 been defined.
- 13575 *LC_ALL* If set to a non-empty string value, override the values of all the other
 13576 internationalization variables.
- 13577 *LC_COLLATE* Determine the locale for the behavior of ranges, equivalence classes, and multi-
 13578 character collating elements within regular expressions.
 13579
- 13580 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 13581 characters (for example, single-byte as opposed to multi-byte characters in
 13582 arguments and input files), the behavior of character classes within regular
 13583 expressions, the classification of characters as uppercase or lowercase letters, the
 13584 case conversion of letters, and the detection of word boundaries.
- 13585 *LC_MESSAGES* Determine the locale that should be used to affect the format and contents of
 13586 diagnostic messages written to standard error.
 13587
- 13588 *LINES* Override the system-selected vertical screen size, used as the number of lines in a
 13589 screenful and the vertical screen size in visual mode. See the Base Definitions
 13590 volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables for valid
 13591 values and results when it is unset or null.
- 13592 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 13593 *PATH* Determine the search path for the shell command specified in the *ex* editor
 13594 commands **!**, **shell**, **read**, and **write**, and the open and visual mode command **!**; see
 13595 the description of command search and execution in Section 2.9.1.1 (on page 2257).
- 13596 *SHELL* Determine the preferred command line interpreter for use as the default value of
 13597 the **shell** edit option.
- 13598 *TERM* Determine the name of the terminal type. If this variable is unset or null, an
 13599 unspecified default terminal type shall be used.
- 13600 **ASYNCHRONOUS EVENTS**
- 13601 The following term is used in this and following sections to specify command and asynchronous
 13602 event actions:
- 13603 *complete write*
- 13604 A complete write is a write of the entire contents of the edit buffer to a file of a type
 13605 other than a terminal device, or the saving of the edit buffer caused by the user
 13606 executing the *ex* **preserve** command. Writing the contents of the edit buffer to a
 13607 temporary file that will be removed when the editor exits shall not be considered a
 13608 complete write.
- 13609 The following actions shall be taken upon receipt of signals:
- 13610 *SIGINT* If the standard input is not a terminal device, *ex* shall not write the file or return to
 13611 command or text input mode, and shall exit with a non-zero exit status.
- 13612 Otherwise, if executing an open or visual text input mode command, *ex* in receipt
 13613 of *SIGINT* shall behave identically to its receipt of the <ESC> character.

- 13614 Otherwise:
- 13615 1. If executing an *ex* text input mode command, all input lines that have been
 - 13616 completely entered shall be resolved into the edit buffer, and any partially
 - 13617 entered line shall be discarded.
 - 13618 2. If there is a currently executing command, it shall be aborted and a message
 - 13619 displayed. Unless otherwise specified by the *ex* or *vi* command descriptions,
 - 13620 it is unspecified whether any lines modified by the executing command
 - 13621 appear modified, or as they were before being modified by the executing
 - 13622 command, in the buffer.
- 13623 If the currently executing command was a motion command, its associated
- 13624 command shall be discarded.
- 13625 3. If in open or visual command mode, the terminal shall be alerted.
 - 13626 4. The editor shall then return to command mode.
- 13627 SIGCONT The screen shall be refreshed if in open or visual mode.
- 13628 SIGHUP If the edit buffer has been modified since the last complete write, *ex* shall attempt
- 13629 to save the edit buffer so that it can be recovered later using the **-r** option or the *ex*
- 13630 **recover** command. The editor shall not write the file or return to command or text
- 13631 input mode, and shall terminate with a non-zero exit status.
- 13632 SIGTERM Refer to SIGHUP.
- 13633 The action taken for all other signals is unspecified.
- 13634 **STDOUT**
- 13635 The standard output shall be used only for writing prompts to the user, for informational
- 13636 messages, and for writing lines from the file.
- 13637 **STDERR**
- 13638 Used only for diagnostic messages.
- 13639 **OUTPUT FILES**
- 13640 The output from *ex* shall be text files.
- 13641 **EXTENDED DESCRIPTION**
- 13642 Only the *ex* mode of the editor is described in this section. See *vi* for additional editing
- 13643 capabilities available in *ex*.
- 13644 When an error occurs, *ex* shall write a message. If the terminal supports a standout mode (such
- 13645 as inverse video), the message shall be written in standout mode. If the terminal does not
- 13646 support a standout mode, and the edit option **errorbells** is set, an alert action shall precede the
- 13647 error message.
- 13648 By default, *ex* shall start in command mode, which shall be indicated by a **:** prompt; see the
- 13649 **prompt** command. Text input mode can be entered by the **append**, **insert**, or **change** commands;
- 13650 it can be exited (and command mode re-entered) by typing a period (**' . '**) alone at the beginning
- 13651 of a line.

13652 **Initialization in ex and vi**

13653 The following symbols are used in this and following sections to specify locations in the edit
13654 buffer:

13655 *alternate and current path names*

13656 Two path names, named *current* and *alternate*, are maintained by the editor. Any *ex*
13657 commands that take file names as arguments shall set them as follows:

- 13658 1. If a *file* argument is specified to the *ex edit*, *ex*, or *recover* commands, or if an *ex tag*
13659 command replaces the contents of the edit buffer.
 - 13660 a. If the command replaces the contents of the edit buffer, the current path name
13661 shall be set to the *file* argument or the file indicated by the tag, and the alternate
13662 path name shall be set to the previous value of the current path name.
 - 13663 b. Otherwise, the alternate path name shall be set to the *file* argument.
- 13664 2. If a *file* argument is specified to the *ex next* command:
 - 13665 a. If the command replaces the contents of the edit buffer, the current path name
13666 shall be set to the first *file* argument, and the alternate path name shall be set to
13667 the previous value of the current path name.
- 13668 3. If a *file* argument is specified to the *ex file* command, the current path name shall be
13669 set to the *file* argument, and the alternate path name shall be set to the previous value
13670 of the current path name.
- 13671 4. If a *file* argument is specified to the *ex read* and *write* commands (that is, when
13672 reading or writing a file, and not to the program named by the *shell* edit option), or a
13673 *file* argument is specified to the *ex xit* command:
 - 13674 a. If the current path name has no value, the current path name shall be set to the
13675 *file* argument.
 - 13676 b. Otherwise, the alternate path name shall be set to the *file* argument.

13677 If the alternate path name is set to the previous value of the current path name when the
13678 current path name had no previous value, then the alternate path name shall have no value
13679 as a result.

13680 *current line*

13681 The line of the edit buffer referenced by the cursor. Each command description specifies the
13682 current line after the command has been executed, as the *current line value*. When the edit
13683 buffer contains no lines, the current line shall be zero; see **Addressing in ex** (on page 2571).

13684 *current column*

13685 The current screen column occupied by the cursor. (The columns shall be numbered
13686 beginning at 1.) Each command description specifies the current column after the command
13687 has been executed, as the *current column value*. This column is an *ideal* column that is
13688 remembered over the lifetime of the editor. The actual screen column upon which the cursor
13689 rests may be different from the current column; see the cursor positioning discussion in
13690 **Command Descriptions in vi** (on page 3201).

13691 *set to non-<blank>*

13692 A description for a current column value, meaning that the current column shall be set to
13693 the last screen column on which is displayed any part of the first non-<blank> character of
13694 the line. If the line has no non-<blank> characters, the current column shall be set to the last
13695 screen column on which is displayed any part of the last character in the line. If the line is
13696 empty, the current column shall be set to column position 1.

13697 The length of lines in the edit buffer may be limited to {LINE_MAX} bytes. In open and visual
 13698 mode, the length of lines in the edit buffer may be limited to the number of characters that will
 13699 fit in the display. If either limit is exceeded during editing, an error message shall be written. If
 13700 either limit is exceeded by a line read in from a file, an error message shall be written and the
 13701 edit session may be terminated.

13702 If the editor stops running due to any reason other than a user command, and the edit buffer has
 13703 been modified since the last complete write, it shall be equivalent to a SIGHUP asynchronous
 13704 event. If the system crashes, it shall be equivalent to a SIGHUP asynchronous event.

13705 During initialization (before the first file is copied into the edit buffer or any user commands
 13706 from the terminal are processed) the following shall occur:

13707 1. If the environment variable *EXINIT* is set, the editor shall execute the *ex* commands
 13708 contained in that variable.

13709 2. If the *EXINIT* variable is not set, and all of the following are true:

13710 a. The *HOME* environment variable is not null and not empty.

13711 b. The file *.exrc* in the directory referred to by the *HOME* environment variable:

13712 1. Exists

13713 2. Is owned by the same user ID as the real user ID of the process or the process
 13714 has appropriate privileges

13715 3. Is not writeable by anyone other than the owner

13716 the editor shall execute the *ex* commands contained in that file.

13717 3. If and only if all the following are true:

13718 a. The current directory is not referred to by the *HOME* environment variable.

13719 b. A command in the *EXINIT* environment variable or a command in the *.exrc* file in the
 13720 directory referred to by the *HOME* environment variable sets the editor option *exrc*.

13721 c. The *.exrc* file in the current directory:

13722 1. Exists

13723 2. Is owned by the same user ID as the real user ID of the process, or by one of a
 13724 set of implementation-defined user IDs

13725 3. Is not writeable by anyone other than the owner

13726 the editor shall attempt to execute the *ex* commands contained in that file.

13727 Lines in any *.exrc* file that contain no characters or only <blank> characters shall be ignored. If
 13728 any *.exrc* file exists, but is not read for ownership or permission reasons, it shall be an error.

13729 After the *EXINIT* variable and any *.exrc* files are processed, the first file specified by the user
 13730 shall be edited, as follows:

13731 1. If the user specified the *-t* option, the effect shall be as if the *ex tag* command was entered
 13732 with the specified argument, with the exception that if tag processing does not result in a
 13733 file to edit, the effect shall be as described in step 3. below.

13734 2. Otherwise, if the user specified any command line *file* arguments, the effect shall be as if
 13735 the *ex edit* command was entered with the first of those arguments as its *file* argument.

13736 3. Otherwise, the effect shall be as if the *ex edit* command was entered with a nonexistent file
 13737 name as its *file* argument. It is unspecified whether this action shall set the current path

13738 name. In an implementation where this action does not set the current path name, any
 13739 editor command using the current path name shall fail until an editor command sets the
 13740 current path name.

13741 If the `-r` option was specified the first time a file in the initial argument list or a file specified by
 13742 the `-t` option is edited, if recovery information has previously been saved about it, that
 13743 information shall be recovered and the editor shall behave as if the contents of the edit buffer
 13744 have already been modified. If there are multiple instances of the file to be recovered, the one
 13745 most recently saved shall be recovered, and an informational message that there are previous
 13746 versions of the file that can be recovered shall be written. If no recovery information about a file
 13747 is available, an informational message to this effect shall be written, and the edit shall proceed as
 13748 usual.

13749 If the `-` option was specified the first time a file that already exists (including a file that might
 13750 not exist but for which recovery information is available, when the `-r` option is specified)
 13751 replaces or initializes the contents of the edit buffer, the current line shall be set to the last line of
 13752 the edit buffer, the current column shall be set to non-`<blank>`, and the `ex` commands specified
 13753 with the `-c` option shall be executed. In this case, the current line and current column shall not be
 13754 set as described for the command associated with the replacement or initialization of the edit
 13755 buffer contents. However, if the `-t` option or a `tag` command is associated with this action, the `-c`
 13756 option commands shall be executed and then the movement to the tag shall be performed.

13757 The current argument list shall initially be set to the file names specified by the user on the
 13758 command line. If no file names are specified by the user, the current argument list shall be
 13759 empty. If the `-t` option was specified, it is unspecified whether any file name resulting from tag
 13760 processing shall be prepended to the current argument list. In the case where the file name is
 13761 added as a prefix to the current argument list, the current argument list reference shall be set to
 13762 that file name. In the case where the file name is not added as a prefix to the current argument
 13763 list, the current argument list reference shall logically be located before the first of the file names
 13764 specified on the command line (for example, a subsequent `ex next` command shall edit the first
 13765 file name from the command line). If the `-t` option was not specified, the current argument list
 13766 reference shall be to the first of the file names on the command line.

13767 Addressing in `ex`

13768 Addressing in `ex` relates to the current line and the current column; the address of a line is its 1-
 13769 based line number, the address of a column is its 1-based count from the beginning of the line.
 13770 Generally, the current line is the last line affected by a command. The current line number is the
 13771 address of the current line. In each command description, the effect of the command on the
 13772 current line number and the current column is described.

13773 Addresses are constructed as follows:

- 13774 1. The character `'.'` (period) shall address the current line.
- 13775 2. The character `'$'` shall address the last line of the edit buffer.
- 13776 3. The positive decimal number `n` shall address the `n`th line of the edit buffer.
- 13777 4. The address `"'x"` refers to the line marked with the mark name character `'x'`, which shall
 13778 be a lowercase letter from the portable character set or one of the characters `'\'` or `'\'`. It
 13779 shall be an error if the line that was marked is not currently present in the edit buffer or the
 13780 mark has not been set. Lines can be marked with the `ex mark` or `k` commands, or the `vi m`
 13781 command.
- 13782 5. A regular expression (RE) enclosed by slashes (`'/'`) shall address the first line found by
 13783 searching forwards from the line following the current line toward the end of the edit

13784 buffer and stopping at the first line containing a string matching the regular expression. As
 13785 stated in **Regular Expressions in ex** (on page 2601), an address consisting of a null regular
 13786 expression delimited by slashes "/" shall address the next line containing the last regular
 13787 expression encountered. In addition, the second slash can be omitted at the end of a
 13788 command line. If the **wrapsan** edit option is set, the search shall wrap around to the
 13789 beginning of the edit buffer and continue up to and including the current line, so that the
 13790 entire edit buffer is searched. Within the regular expression, the sequence "\/" shall
 13791 represent a literal slash instead of the regular expression delimiter.

13792 6. A regular expression enclosed in question marks ('?') shall address the first line found by
 13793 searching backwards from the line preceding the current line toward the beginning of the
 13794 edit buffer and stopping at the first line containing a string matching the regular
 13795 expression. The second question mark can be omitted at the end of a command line. If the
 13796 **wrapsan** edit option is set, the search shall wrap around from the beginning of the edit
 13797 buffer to the end of the edit buffer and continue up to and including the current line, so
 13798 that the entire edit buffer is searched. Within the regular expression, the sequence "\?"
 13799 shall represent a literal question mark instead of the RE delimiter.

13800 7. A plus sign ('+') or a minus sign ('-') followed by a decimal number shall address the
 13801 current line plus or minus the number. A '+' or '-' not followed by a decimal number
 13802 shall address the current line plus or minus 1.

13803 Addresses can be followed by zero or more address offsets, optionally <blank> character-
 13804 separated. Address offsets are constructed as follows:

13805 1. A '+' or '-' immediately followed by a decimal number shall add (subtract) the
 13806 indicated number of lines to (from) the address. A '+' or '-' not followed by a decimal
 13807 number shall add (subtract) 1 to (from) the address.

13808 2. A decimal number shall add the indicated number of lines to the address.

13809 It shall not be an error for an intermediate address value to be less than zero or greater than the
 13810 last line in the edit buffer. It shall be an error for the final address value to be less than zero or
 13811 greater than the last line in the edit buffer.

13812 Commands take zero, one, or two addresses; see the descriptions of *1addr* and *2addr* in
 13813 **Command Descriptions in ex** (on page 2578). If more than the required number of addresses
 13814 are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more
 13815 than the required number of addresses are provided to a command, the addresses specified first
 13816 shall be evaluated and then discarded until the maximum number of valid addresses remain.

13817 Addresses shall be separated from each other by a comma (',') or a semicolon (';'). If no
 13818 address is specified before or after a comma or semicolon separator, it shall be as if the address
 13819 of the current line was specified before or after the separator. In the case of a semicolon
 13820 separator, the current line ('.') shall be set to the first address, and only then will the next
 13821 address be calculated. This feature can be used to determine the starting line for forwards and
 13822 backwards searches (see rules 5. and 6.).

13823 A percent sign ('%') shall be equivalent to entering the two addresses "1, \$".

13824 Any delimiting <blank> characters between addresses, address separators, or address offsets
 13825 shall be discarded.

13826 **Command Line Parsing in ex**

13827 The following symbol is used in this and following sections to describe parsing behavior:

13828 *escape* If a character is referred to as “backslash escaped” or “<control>-V escaped,” it
 13829 shall mean that the character acquired or lost a special meaning by virtue of being
 13830 preceded, respectively, by a backslash or <control>-V character. Unless otherwise
 13831 specified, the escaping character shall be discarded at that time and shall not be
 13832 further considered for any purpose.

13833 Command-line parsing shall be done in the following steps. For each step, characters already
 13834 evaluated shall be ignored; that is, the phrase “leading character” refers to the next character
 13835 that has not yet been evaluated.

- 13836 1. Leading colon characters shall be skipped.
- 13837 2. Leading <blank> characters shall be skipped.
- 13838 3. If the leading character is a double-quote character, the characters up to and including the
 13839 next non-backslash-escaped <newline> character shall be discarded, and any subsequent
 13840 characters shall be parsed as a separate command.
- 13841 4. Leading characters that can be interpreted as addresses shall be evaluated; see **Addressing**
 13842 **in ex** (on page 2571).
- 13843 5. Leading <blank> characters shall be skipped.
- 13844 6. If the next character is a vertical-line character or a <newline> character:
 - 13845 a. If the next character is a <newline> character:
 - 13846 1. If *ex* is in open or visual mode, the current line shall be set to the last address
 13847 specified, if any.
 - 13848 2. Otherwise, if the last command was terminated by a vertical-line character, no
 13849 action shall be taken; for example, the command "||<newline>" shall
 13850 execute two implied commands, not three.
 - 13851 3. Otherwise, step 6.b. shall apply.
 - 13852 b. Otherwise, the implied command shall be the **print** command. The last #, **p**, and **l**
 13853 flags specified to any *ex* command shall be remembered and shall apply to this
 13854 implied command. Executing the *ex* **number**, **print**, or **list** command shall set the
 13855 remembered flags to #, nothing, and **l**, respectively, plus any other flags specified for
 13856 that execution of the **number**, **print**, or **list** command.
 If *ex* is not currently performing a **global** or **v** command, and no address or count is
 13857 specified, the current line shall be incremented by 1 before the command is executed.
 13858 If incrementing the current line would result in an address past the last line in the
 13859 edit buffer, the command shall fail, and the increment shall not happen.
 13860
 - 13861 c. The <newline> character or vertical-line character shall be discarded and any
 13862 subsequent characters shall be parsed as a separate command.
- 13863 7. The command name shall be comprised of the next character (if the character is not
 13864 alphabetic), or the next character and any subsequent alphabetic characters (if the
 13865 character is alphabetic), with the following exceptions:
 - 13866 a. Commands that consist of any prefix of the characters in the command name **delete**,
 13867 followed immediately by any of the characters **l**, **p**, **+**, **-**, or **#** shall be interpreted as a
 13868 **delete** command, followed by a <blank> character, followed by the characters that

13869 were not part of the prefix of the **delete** command. The maximum number of
 13870 characters shall be matched to the command name **delete**; for example, "de1" shall
 13871 not be treated as "de" followed by the flag **l**.

13872 b. Commands that consist of the character **k**, followed by a character that can be used
 13873 as the name of a mark, shall be equivalent to the mark command followed by a
 13874 <blank> character, followed by the character that followed the **k**.

13875 c. Commands that consist of the character **s**, followed by characters that could be
 13876 interpreted as valid options to the **s** command, shall be the equivalent of the **s**
 13877 command, without any pattern or replacement values, followed by a <blank>
 13878 character, followed by the characters after the **s**.

13879 8. The command name shall be matched against the possible command names, and a
 13880 command name that contains a prefix matching the characters specified by the user shall
 13881 be the executed command. In the case of commands where the characters specified by the
 13882 user could be ambiguous, the executed command shall be as follows:

13883
 13884
 13885
 13886
 13887
 13888

a	append	n	next	t	t
c	change	p	print	u	undo
ch	change	pr	print	un	undo
e	edit	r	read	v	v
m	move	re	read	w	write
ma	mark	s	s		

13889 Implementation extensions with names causing similar ambiguities shall not be checked
 13890 for a match until all possible matches for commands specified by IEEE Std. 1003.1-200x
 13891 have been checked.

13892 9. If the command is a **!** command, or if the command is a **read** command followed by zero
 13893 or more <blank> characters and a **!**, or if the command is a **write** command followed by
 13894 one or more <blank> characters and a **!**, the rest of the command shall include all
 13895 characters up to a non-backslash-escaped <newline> character. The <newline> character
 13896 shall be discarded and any subsequent characters shall be parsed as a separate *ex*
 13897 command.

13898 10. Otherwise, if the command is an **edit**, **ex**, or **next** command, or a **visual** command while in
 13899 open or visual mode, the next part of the command shall be parsed as follows:

13900 a. Any '!' character immediately following the command shall be skipped and be part
 13901 of the command.

13902 b. Any leading <blank> characters shall be skipped and be part of the command.

13903 c. If the next character is a '+', characters up to the first non-backslash-escaped
 13904 <newline> character or non-backslash-escaped <blank> character shall be skipped
 13905 and be part of the command.

13906 d. The rest of the command shall be determined by the steps specified in paragraph 12.

13907 11. Otherwise, if the command is a **global**, **open**, **s**, or **v** command, the next part of the
 13908 command shall be parsed as follows:

13909 a. Any leading <blank> characters shall be skipped and be part of the command.

13910 b. If the next character is not an alphanumeric, double-quote, <newline>, backslash, or
 13911 vertical-line character:

13912 1. The next character shall be used as a command delimiter.

- 13913 2. If the command is a **global**, **open**, or **v** command, characters up to the first
13914 non-backslash-escaped <newline> character, or first non-backslash-escaped
13915 delimiter character, shall be skipped and be part of the command.
- 13916 3. If the command is an **s** command, characters up to the first non-backslash-
13917 escaped <newline> character, or second non-backslash-escaped delimiter
13918 character, shall be skipped and be part of the command.
- 13919 c. If the command is a **global** or **v** command, characters up to the first non-backslash-
13920 escaped <newline> character shall be skipped and be part of the command.
- 13921 d. Otherwise, the rest of the command shall be determined by the steps specified in
13922 paragraph 12.

13923 12. Otherwise:

- 13924 a. If the command was a **map**, **unmap**, **abbreviate**, or **unabbreviate** command,
13925 characters up to the first non-<control>-V-escaped <newline>, vertical-line, or
13926 double-quote character shall be skipped and be part of the command.
- 13927 b. Otherwise, characters up to the first non-backslash-escaped <newline>, vertical-line,
13928 or double-quote character shall be skipped and be part of the command.
- 13929 c. If the command was an **append**, **change**, or **insert** command, and the step 12.b.
13930 ended at a vertical-line character, any subsequent characters, up to the next non-
13931 backslash-escaped <newline> character shall be used as input text to the command.
- 13932 d. If the command was ended by a double-quote character, all subsequent characters,
13933 up to the next non-backslash-escaped <newline> character, shall be discarded.
- 13934 e. The terminating <newline> or vertical-line character shall be discarded and any
13935 subsequent characters shall be parsed as a separate *ex* command.

13936 Command arguments shall be parsed as described by the Synopsis and Description of each
13937 individual *ex* command. This parsing shall not be <blank> character-sensitive, except for the **!**
13938 argument, which must follow the command name without intervening <blank> characters, and
13939 where it would otherwise be ambiguous. For example, *count* and *flag* arguments need not be
13940 <blank> character separated because "d22p" is not ambiguous, but *file* arguments to the *ex next*
13941 command must be separated by one or more <blank> characters. Any <blank> character in
13942 command arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands can be
13943 <control>-V-escaped, in which case the <blank> character shall not be used as an argument
13944 delimiter. Any <blank> character in the command argument for any other command can be
13945 backslash-escaped, in which case that <blank> character shall not be used as an argument
13946 delimiter.

13947 Within command arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands,
13948 any character can be <control>-V-escaped. All such escaped characters shall be treated literally
13949 and shall have no special meaning. Within command arguments for all other *ex* commands that
13950 are not regular expressions or replacement strings, any character that would otherwise have a
13951 special meaning can be backslash-escaped. Escaped characters shall be treated literally, without
13952 special meaning as shell expansion characters or '!', '%', and '#' expansion characters. See
13953 **Regular Expressions in ex** (on page 2601) and **Replacement Strings in ex** (on page 2602) for
13954 descriptions of command arguments that are regular expressions or replacement strings.

13955 Non-backslash-escaped '%' characters appearing in *file* arguments to any *ex* command shall be
13956 replaced by the current path name; unescaped '#' characters shall be replaced by the alternate
13957 path name. It shall be an error if '%' or '#' characters appear unescaped in an argument and
13958 their corresponding values are not set.

13959 Non-backslash-escaped '!' characters in the arguments to either the **ex!** command or the open
 13960 and visual mode **!** command, or in the arguments to the **ex read** command, where the first non-
 13961 <blank> character after the command name is a '!' character, or in the arguments to the **ex**
 13962 **write** command where the command name is followed by one or more <blank> characters and
 13963 the first non-<blank> character after the command name is a '!' character, shall be replaced
 13964 with the arguments to the last of those three commands as they appeared after all unescaped
 13965 '%', '#', and '!' characters were replaced. It shall be an error if '!' characters appear
 13966 unescaped in one of these commands and there has been no previous execution of one of these
 13967 commands.

13968 If an error occurs during the parsing or execution of an **ex** command:

- 13969 • An informational message to this effect shall be written. Execution of the **ex** command shall
 13970 stop, and the cursor (for example, the current line and column) shall not be further modified.
- 13971 • If the **ex** command resulted from a map expansion, all characters from that map expansion
 13972 shall be discarded, except as otherwise specified by the **map** command.
- 13973 • Otherwise, if the **ex** command resulted from the processing of an *EXINIT* environment
 13974 variable, a **.exrc** file, a **:source** command, a **-c** option, or a **+command** specified to an **ex edit**,
 13975 **ex, next**, or **visual** command, no further commands from the source of the commands shall
 13976 be executed.
- 13977 • Otherwise, if the **ex** command resulted from the execution of a buffer or a **global** or **v**
 13978 command, no further commands caused by the execution of the buffer or the **global** or **v**
 13979 command shall be executed.
- 13980 • Otherwise, if the **ex** command was not terminated by a <newline> character, all characters up
 13981 to and including the next non-backslash-escaped <newline> character shall be discarded.

13982 **Input Editing in ex**

13983 The following symbols are used in this and following sections to specify command actions.

13984 *word* In the POSIX locale, a word consists of a maximal sequence of letters, digits, and
 13985 underscores, delimited at both ends by characters other than letters, digits, or
 13986 underscores, or by the beginning or end of a line or the edit buffer.

13987 When accepting input characters from the user, in either **ex** command mode or **ex** text input
 13988 mode, **ex** shall enable canonical mode input processing, as defined in the System Interfaces
 13989 volume of IEEE Std. 1003.1-200x.

13990 If in **ex** text input mode:

- 13991 1. If the **number** edit option is set, **ex** shall prompt for input using the line number that would
 13992 be assigned to the line if it is entered, in the format specified for the **ex number** command.
- 13993 2. If the **autoindent** edit option is set, **ex** shall prompt for input using **autoindent** characters,
 13994 as described by the **autoindent** edit option. **autoindent** characters shall follow the line
 13995 number, if any.

13996 If in **ex** command mode:

- 13997 1. If the **prompt** edit option is set, input shall be prompted for using a single ':' character;
 13998 otherwise, there shall be no prompt.

13999 The input characters in the following sections shall have the following effects on the input line.

14000 **Scroll**

14001 *Synopsis:* eof

14002 See the description of the *stty eof* character in *stty*.

14003 If in *ex* command mode:

14004 If the *eof* character is the first character entered on the line, the line shall be evaluated as if it

14005 contained two characters: a <control>-D and a <newline> character.

14006 Otherwise, the *eof* character shall have no special meaning.

14007 If in *ex* text input mode:

14008 If the cursor follows an **autoindent** character, the **autoindent** characters in the line shall be

14009 modified so that a part of the next text input character will be displayed on the first column

14010 in the line after the previous **shiftwidth** edit option column boundary, and the user shall be

14011 prompted again for input for the same line.

14012 Otherwise, if the cursor follows a '0', which follows an **autoindent** character, and the '0'

14013 was the previous text input character, the '0' and all **autoindent** characters in the line shall

14014 be discarded, and the user shall be prompted again for input for the same line.

14015 Otherwise, if the cursor follows a '^', which follows an **autoindent** character, and the '^'

14016 was the previous text input character, the '^' and all **autoindent** characters in the line shall

14017 be discarded, and the user shall be prompted again for input for the same line. In addition,

14018 the **autoindent** level for the next input line shall be derived from the same line from which

14019 the **autoindent** level for the current input line was derived.

14020 Otherwise, if there are no **autoindent** or text input characters in the line, the *eof* character

14021 shall be discarded.

14022 Otherwise, the *eof* character shall have no special meaning.

14023 <newline>

14024 *Synopsis:* <newline>

14025 <control>-J

14026 If in *ex* command mode:

14027 Cause the command line to be parsed; <control>-J shall be mapped to the <newline>

14028 character for this purpose.

14029 If in *ex* text input mode:

14030 Terminate the current line. If there are no characters other than **autoindent** characters on the

14031 line, all characters on the line shall be discarded.

14032 Prompt for text input on a new line after the current line. If the **autoindent** edit option is set,

14033 an appropriate number of **autoindent** characters shall be added as a prefix to the line as

14034 described by the *ex autoindent* edit option.

- 14035 **<backslash>**
- 14036 *Synopsis:* <backslash>
- 14037 Allow the entry of a subsequent <newline> or <control>-J as a literal character, removing any special meaning that it may have to the editor during text input mode. The backslash character shall be retained and evaluated when the command line is parsed, or retained and included when the input text becomes part of the edit buffer.
- 14038
- 14039
- 14040
- 14041 **<control>-V**
- 14042 *Synopsis:* <control>-V
- 14043 Allow the entry of any subsequent character as a literal character, removing any special meaning that it may have to the editor during text input mode. The <control>-V character shall be discarded before the command line is parsed or the input text becomes part of the edit buffer.
- 14044
- 14045
- 14046 If the “literal next” functionality is performed by the underlying system, it is implementation-defined whether a character other than <control>-V performs this function.
- 14047
- 14048 **<control>-W**
- 14049 *Synopsis:* <control>-W
- 14050 Discard the <control>-W, and the word previous to it in the input line, including any <blank> characters following the word and preceding the <control>-W. If the “word erase” functionality is performed by the underlying system, it is implementation-defined whether a character other than <control>-W performs this function.
- 14051
- 14052
- 14053
- 14054 **Command Descriptions in ex**
- 14055 The following symbols are used in this section to represent command modifiers. Some of these modifiers can be omitted, in which case the specified defaults shall be used.
- 14056
- 14057 *1addr* A single line address, given in any of the forms described in **Addressing in ex** (on page 2571); the default shall be the current line (‘ . ’), unless otherwise specified.
- 14058
- 14059 If the line address is zero, it shall be an error, unless otherwise specified in the following command descriptions.
- 14060
- 14061 If the edit buffer is empty, and the address is specified with a command other than =, **append**, **insert**, **open**, **put**, **read**, or **visual**, or the address is not zero, it shall be an error.
- 14062
- 14063
- 14064 *2addr* Two addresses specifying an inclusive range of lines. If no addresses are specified, the default for *2addr* shall be the current line only (“ . . ”), unless otherwise specified in the following command descriptions. If one address is specified, *2addr* shall specify that line only, unless otherwise specified in the following command descriptions.
- 14065
- 14066
- 14067
- 14068
- 14069 It shall be an error if the first address is greater than the second address.
- 14070 If the edit buffer is empty, and the two addresses are specified with a command other than the **!**, **write**, **wq**, or **xit** commands, or either address is not zero, it shall be an error.
- 14071
- 14072
- 14073 *count* A positive decimal number. If *count* is specified, it shall be equivalent to specifying an additional address to the command, unless otherwise specified by the following command descriptions. The additional address shall be equal to the last address specified to the command (either explicitly or by default) plus *count*–1.
- 14074
- 14075
- 14076

14077 If this would result in an address greater than the last line of the edit buffer, it shall
 14078 be corrected to equal the last line of the edit buffer.

14079 *flags* One or more of the characters '+' , '-' , '#' , 'p' , or 'l' (ell). The flag characters
 14080 can be <blank>-separated, and in any order or combination. The characters '#' ,
 14081 'p' , and 'l' shall cause lines to be written in the format specified by the **print**
 14082 command with the specified *flags*.

14083 The lines to be written are as follows:

- 14084 1. All edit buffer lines written during the execution of the *ex* &, ~, **list**, **number**,
 14085 **open**, **print**, **s**, **visual**, and **z** commands shall be written as specified by *flags*.
- 14086 2. After the completion of an *ex* command with a flag as an argument, the
 14087 current line shall be written as specified by *flags*, unless the current line was
 14088 the last line written by the command.

14089 The characters '+' and '-' cause the value of the current line after the execution
 14090 of the *ex* command to be adjusted by the offset address as described in **Addressing**
 14091 **in ex** (on page 2571). This adjustment shall occur before the current line is written
 14092 as described in 2. above.

14093 The default for *flags* shall be none.

14094 *buffer* One of a number of named areas for holding text. The named buffers are specified
 14095 by the alphanumeric characters of the POSIX locale. There shall also be one
 14096 "unnamed" buffer. When no buffer is specified for editor commands that use a
 14097 buffer, the unnamed buffer shall be used. Commands that store text into buffers
 14098 shall store the text as it was before the command took effect, and shall store text
 14099 occurring earlier in the file before text occurring later in the file, regardless of how
 14100 the text region was specified. Commands that store text into buffers shall store the
 14101 text into the unnamed buffer as well as any specified buffer.

14102 In *ex* commands, buffer names are specified as the name by itself. In open or visual
 14103 mode commands the name is preceded by a double quote ('"') character.

14104 If the specified buffer name is an uppercase character, and the buffer contents are
 14105 to be modified, the buffer shall be appended to rather than being overwritten. If
 14106 the buffer is not being modified, specifying the buffer name in lowercase and
 14107 uppercase shall have identical results.

14108 There shall also be buffers named by the numbers 1 through 9. In open and visual
 14109 mode, if a region of text including characters from more than a single line is being
 14110 modified by the *vi* **c** or **d** commands, the motion character associated with the **c** or
 14111 **d** commands specifies that the buffer text shall be in line mode, or the commands
 14112 %, /, ?, (,), **N**, **n**, {, or } are used to define a region of text for the **c** or **d** commands,
 14113 the contents of buffers 1 through 8 shall be moved into the buffer named by the
 14114 next numerically greater value, the contents of buffer 9 shall be discarded, and the
 14115 region of text shall be copied into buffer 1. This shall be in addition to copying the
 14116 text into a user-specified buffer or unnamed buffer, or both. Numeric buffers can
 14117 be specified as a source buffer for open and visual mode commands; however,
 14118 specifying a numeric buffer as the write target of an open or visual mode
 14119 command shall have unspecified results.

14120 The text of each buffer shall have the characteristic of being in either line or
 14121 character mode. Appending text to a non-empty buffer shall set the mode to match
 14122 the characteristic of the text being appended. Appending text to a buffer shall
 14123 cause the creation of at least one additional line in the buffer. All text stored into

14124 buffers by *ex* commands shall be in line mode. The *ex* commands that use buffers
 14125 as the source of text specify individually how buffers of different modes are
 14126 handled. Each open or visual mode command that uses buffers for any purpose
 14127 specifies individually the mode of the text stored into the buffer and how buffers
 14128 of different modes are handled.

14129 *file* Command text used to derive a path name. The default shall be the current path
 14130 name, as defined previously, in which case, if no current path name has yet been
 14131 established it shall be an error, except where specifically noted in the individual
 14132 command descriptions that follow. If the command text contains any of the
 14133 characters '~', '{', '[', '*', '?', '\$', '\', ' ', ' ', ' ', ' ', and '\', it shall be
 14134 subjected to the process of "shell expansions", as described below; if more than a
 14135 single path name results and the command expects only one, it shall be an error.

14136 The process of shell expansions in the editor shall be done as follows. The *ex* utility
 14137 shall pass two arguments to the program named by the shell edit option; the first
 14138 shall be `-c`, and the second shall be the string "echo" and the command text as a
 14139 single argument. The standard output and standard error of that command shall
 14140 replace the command text.

14141 **!** A character that can be appended to the command name to modify its operation,
 14142 as detailed in the individual command descriptions. With the exception of the *ex*
 14143 **read**, **write**, and **!** commands, the '!' character shall only act as a modifier if there
 14144 are no <blank> characters between it and the command name.

14145 *remembered search direction*

14146 The *vi* commands **N** and **n** begin searching in a forwards or backwards direction in
 14147 the edit buffer based on a remembered search direction, which is initially unset,
 14148 and is set by the *ex* **global**, **v**, **s**, and **tag** commands, and the *vi* / and ? commands.

14149 **Abbreviate**

14150 *Synopsis:* `ab[breviate][lhs rhs]`

14151 If *lhs* and *rhs* are not specified, write the current list of abbreviations and do nothing more.

14152 Implementations may restrict the set of characters accepted in *lhs* or *rh*, except that printable
 14153 characters and <blank> characters shall not be restricted. Additional restrictions shall be
 14154 implementation-defined.

14155 In both *lhs* and *rhs*, any character may be escaped with a <control>-V, in which case the
 14156 character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be
 14157 discarded.

14158 In open and visual text input mode, if a non-word or <ESC> character that is not escaped by a
 14159 <control>-V character is entered after a word character, a check shall be made for a set of
 14160 characters matching *lhs*, in the text input entered during this command. If it is found, the effect
 14161 shall be as if *rhs* was entered instead of *lhs*.

14162 The set of characters that are checked is defined as follows:

- 14163 1. If there are no characters inserted before the word and non-word or <ESC> characters that
 14164 triggered the check, the set of characters shall consist of the word character.
- 14165 2. If the character inserted before the word and non-word or <ESC> characters that triggered
 14166 the check is a word character, the set of characters shall consist of the characters inserted
 14167 immediately before the triggering characters that are word characters, plus the triggering
 14168 word character.

14169 3. If the character inserted before the word and non-word or <ESC> characters that triggered
 14170 the check is not a word character, the set of characters shall consist of the characters that
 14171 were inserted before the triggering characters that are neither <blank> characters nor word
 14172 characters, plus the triggering word character.

14173 It is unspecified whether the *lhs* argument entered for the *ex* **abbreviate** and **unabbreviate**
 14174 commands is replaced in this fashion. Regardless of whether or not the replacement occurs, the
 14175 effect of the command shall be as if the replacement had not occurred.

14176 *Current line*: Unchanged.

14177 *Current column*: Unchanged.

14178 **Append**

14179 *Synopsis*: [*laddr*] a[ppend][!]

14180 Enter text input mode; the input text shall be placed after the specified line. If line zero is
 14181 specified, the text shall be placed at the beginning of the edit buffer.

14182 This command shall be affected by the **number** and **autoindent** edit options; following the
 14183 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
 14184 duration of this command only.

14185 *Current line*: Set to the last input line; if no lines were input, set to the specified line, or to the
 14186 first line of the edit buffer if a line of zero was specified, or zero if the edit buffer is empty.

14187 *Current column*: Set to non-<blank>.

14188 **Arguments**

14189 *Synopsis*: ar[gs]

14190 Write the current argument list, with the current argument-list entry, if any, between '[' and
 14191 ']' characters.

14192 *Current line*: Unchanged.

14193 *Current column*: Unchanged.

14194 **Change**

14195 *Synopsis*: [*2addr*] c[hange][!][*count*]

14196 Enter *ex* text input mode; the input text shall replace the specified lines. The specified lines shall
 14197 be copied into the unnamed buffer, which shall become a line mode buffer.

14198 This command shall be affected by the **number** and **autoindent** edit options; following the
 14199 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
 14200 duration of this command only.

14201 *Current line*: Set to the last input line; if no lines were input, set to the line before the first
 14202 address, or to the first line of the edit buffer if there are no lines preceding the first address, or to
 14203 zero if the edit buffer is empty.

14204 *Current column*: Set to non-<blank>.

14205 **Change Directory**

14206 *Synopsis:* chd[ir][!][*directory*]
 14207 cd[!][*directory*]

14208 Change the current working directory to *directory*.

14209 If no *directory* argument is specified, and the *HOME* environment variable is set to a non-null
 14210 and non-empty value, *directory* shall default to the value named in the *HOME* environment
 14211 variable. If the *HOME* environment variable is empty or is undefined, the default value of
 14212 *directory* is implementation-defined.

14213 If no '!' is appended to the command name, and the edit buffer has been modified since the
 14214 last complete write, and the current path name does not begin with a '/', it shall be an error.

14215 *Current line:* Unchanged.

14216 *Current column:* Unchanged.

14217 **Copy**

14218 *Synopsis:* [*laddr*] co[py] [*laddr*] [*flags*]
 14219 [*laddr*] t [*laddr*] [*flags*]

14220 Copy the specified lines after the specified destination line; line zero specifies that the lines shall
 14221 be placed at the beginning of the edit buffer.

14222 *Current line:* Set to the last line copied.

14223 *Current column:* Set to non-<blank>.

14224 **Delete**

14225 *Synopsis:* [*laddr*] d[elete][*buffer*][*count*][*flags*]

14226 Delete the specified lines into a buffer (defaulting to the unnamed buffer), which shall become a
 14227 line-mode buffer.

14228 Flags can immediately follow the command name; see **Command Line Parsing in ex** (on page
 14229 2573).

14230 *Current line:* Set to the line following the deleted lines, or to the last line in the edit buffer if that
 14231 line is past the end of the edit buffer, or to zero if the edit buffer is empty.

14232 *Current column:* Set to non-<blank>.

14233 **Edit**

14234 *Synopsis:* e[dit][!][+*command*][*file*]
 14235 ex[!][+*command*][*file*]

14236 If no '!' is appended to the command name, and the edit buffer has been modified since the
 14237 last complete write, it shall be an error.

14238 If *file* is specified, replace the current contents of the edit buffer with the current contents of *file*,
 14239 and set the current path name to *file*. If *file* is not specified, replace the current contents of the
 14240 edit buffer with the current contents of the file named by the current path name. If for any
 14241 reason the current contents of the file cannot be accessed, the edit buffer shall be empty.

14242 The *+command* option shall be <blank> character-delimited; <blank> characters within
 14243 *+command* can be escaped by preceding them with a backslash character. The *+command* shall be
 14244 interpreted as an *ex* command immediately after the contents of the edit buffer have been

- 14245 replaced and the current line and column have been set.
- 14246 If the edit buffer is empty:
- 14247 *Current line*: Set to 0.
- 14248 *Current column*: Set to 1.
- 14249 Otherwise, if executed while in *ex* command mode or if the *+command* argument is specified:
- 14250 *Current line*: Set to the last line of the edit buffer.
- 14251 *Current column*: Set to non-<blank>.
- 14252 Otherwise, if *file* is omitted or results in the current path name:
- 14253 *Current line*: Set to the first line of the edit buffer.
- 14254 *Current column*: Set to non-<blank>.
- 14255 Otherwise, if *file* is the same as the last file edited, the line and column shall be set as follows; if
- 14256 the file was previously edited, the line and column may be set as follows:
- 14257 *Current line*: Set to the last value held when that file was last edited. If this value is not a valid
- 14258 line in the new edit buffer, set to the first line of the edit buffer.
- 14259 *Current column*: If the current line was set to the last value held when the file was last edited, set
- 14260 to the last value held when the file was last edited. Otherwise, or if the last value is not a valid
- 14261 column in the new edit buffer, set to non-<blank>.
- 14262 Otherwise:
- 14263 *Current line*: Set to the first line of the edit buffer.
- 14264 *Current column*: Set to non-<blank>.
- 14265 **File**
- 14266 *Synopsis*: f[*file*][*file*]
- 14267 If a *file* argument is specified, the alternate path name shall be set to the current path name, and
- 14268 the current path name shall be set to *file*.
- 14269 Write an informational message. If the file has a current path name, it shall be included in this
- 14270 message; otherwise, the message shall indicate that there is no current path name. If the edit
- 14271 buffer contains lines, the current line number and the number of lines in the edit buffer shall be
- 14272 included in this message; otherwise, the message shall indicate that the edit buffer is empty. If
- 14273 the edit buffer has been modified since the last complete write, this fact shall be included in this
- 14274 message. If the **readonly** edit option is set, this fact shall be included in this message. The
- 14275 message may contain other unspecified information.
- 14276 *Current line*: Unchanged.
- 14277 *Current column*: Unchanged.

14278 **Global**

14279 *Synopsis:* [2addr] g[lobal] /pattern/ [commands]
 14280 [2addr] v /pattern/ [commands]

14281 The optional '!' character after the **global** command shall be the same as executing the **v**
 14282 command.

14283 If *pattern* is empty (for example, "//") or not specified, the last regular expression used in the
 14284 editor command shall be used as the *pattern*. The *pattern* can be delimited by slashes (shown in
 14285 the Synopsis), as well as any non-alphanumeric or non-<blank> character other than backslash,
 14286 vertical line, double quote, or <newline>.

14287 If no lines are specified, the lines shall default to the entire file.

14288 The **global** and **v** commands are logically two-pass operations. First, mark the lines within the
 14289 specified lines that match (**global**) or do not match (**v** or **global!**) the specified pattern. Second,
 14290 execute the *ex* commands given by *commands*, with the current line ('.') set to each marked
 14291 line. If an error occurs during this process, or the contents of the edit buffer are replaced (for
 14292 example, by the *ex* **:edit** command) an error message shall be written and no more commands
 14293 resulting from the execution of this command shall be processed.

14294 Multiple *ex* commands can be specified by entering multiple commands on a single line using a
 14295 vertical line to delimit them, or one per line, by escaping each <newline> with a backslash.

14296 If no commands are specified:

- 14297 1. If in *ex* command mode, it shall be as if the **print** command were specified.
- 14298 2. Otherwise, no command shall be executed.

14299 For the **append**, **change**, and **insert** commands, the input text shall be included as part of the
 14300 command, and the terminating period can be omitted if the command ends the list of
 14301 commands. The **open** and **visual** commands can be specified as one of the commands, in which
 14302 case each marked line shall cause the editor to enter open or visual mode. If open or visual mode
 14303 is exited using the *vi* **Q** command, the current line shall be set to the next marked line, and open
 14304 or visual mode reentered, until the list of marked lines is exhausted.

14305 The **global**, **v**, and **undo** commands cannot be used in *commands*. Marked lines may be deleted
 14306 by commands executed for lines occurring earlier in the file than the marked lines. In this case,
 14307 no commands shall be executed for the deleted lines.

14308 If the remembered search direction is not set, the **global** and **v** commands shall set it to forward.

14309 The **autoprint** and **autoindent** edit options shall be inhibited for the duration of the **g** or **v**
 14310 command.

14311 *Current line:* If no commands executed, set to the last marked line. Otherwise, as specified for
 14312 the executed *ex* commands.

14313 *Current column:* If no commands are executed, set to non-<blank>; otherwise, as specified for the
 14314 individual *ex* commands.

14315 **Insert**14316 *Synopsis:* [*laddr*] i[nsert][!]14317 Enter *ex* text input mode; the input text shall be placed before the specified line. If the line is zero
14318 or 1, the text shall be placed at the beginning of the edit buffer.14319 This command shall be affected by the **number** and **autoindent** edit options; following the
14320 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
14321 duration of this command only.14322 *Current line:* Set to the last input line; if no lines were input, set to the line before the specified
14323 line, or to the first line of the edit buffer if there are no lines preceding the specified line, or zero
14324 if the edit buffer is empty.14325 *Current column:* Set to non-<blank>.14326 **Join**14327 *Synopsis:* [*2addr*] j[oin][!][*count*][*flags*]14328 If *count* is specified:14329 If no address was specified, the **join** command shall behave as if *2addr* were the current line
14330 and the current line plus *count* (.,. + *count*).14331 If one address was specified, the **join** command shall behave as if *2addr* were the specified
14332 address and the specified address plus *count* (*addr*,*addr* + *count*).14333 If two addresses were specified, the **join** command shall behave as if an additional address,
14334 equal to the last address plus *count* - 1 (*addr1*,*addr2*,*addr2* + *count* - 1), was specified.14335 If this would result in a second address greater than the last line of the edit buffer, it shall be
14336 corrected to be equal to the last line of the edit buffer.14337 If no *count* is specified:14338 If no address was specified, the **join** command shall behave as if *2addr* were the current line
14339 and the next line (.,. + 1).14340 If one address was specified, the **join** command shall behave as if *2addr* were the specified
14341 address and the next line (*addr*,*addr* + 1).14342 Join the text from the specified lines together into a single line, which shall replace the specified
14343 lines.14344 If a '!' character is appended to the command name, the **join** shall be without modification of
14345 any line, independent of the current locale.14346 Otherwise, in the POSIX locale, set the current line to the first of the specified lines, and then, for
14347 each subsequent line, proceed as follows:

- 14348 1. Discard leading <space>s from the line to be joined.
- 14349 2. If the line to be joined is now empty, delete it, and skip steps 3 through 5.
- 14350 3. If the current line ends in a <blank> character, or the first character of the line to be joined
14351 is a ') ' character, join the lines without further modification.
- 14352 4. If the last character of the current line is a ' . ' , join the lines with two <space> characters
14353 between them.

14354 5. Otherwise, join the lines with a single <space> character between them.

14355 *Current line*: Set to the first line specified.

14356 *Current column*: Set to non-<blank>.

14357 **List**

14358 *Synopsis*: [2*addr*] l[*ist*][*count*][*flags*]

14359 This command shall be equivalent to the *ex* command:

14360 [2*addr*] p[*rint*][*count*] l[*flags*]

14361 See **Print** (on page 2590).

14362 **Map**

14363 *Synopsis*: map[!][*lhs rhs*]

14364 If *lhs* and *rhs* are not specified:

- 14365 1. If '!' is specified, write the current list of text input mode maps.
- 14366 2. Otherwise, write the current list of command mode maps.
- 14367 3. Do nothing more.

14368 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable
 14369 characters and <blank> characters shall not be restricted. Additional restrictions shall be
 14370 implementation-defined. In both *lhs* and *rhs*, any character can be escaped with a <control>-V, in
 14371 which case the character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V
 14372 shall be discarded.

14373 If the character '!' is appended to the **map** command name, the mapping shall be effective
 14374 during open or visual text input mode rather than **open** or **visual** command mode. This allows
 14375 *lhs* to have two different **map** definitions at the same time: one for command mode and one for
 14376 text input mode.

14377 For command mode mappings:

14378 When the *lhs* is entered as any part of a *vi* command in open or visual mode (but not as part
 14379 of the arguments to the command), the action shall be as if the corresponding *rhs* had been
 14380 entered.

14381 If any character in the command, other than the first, is escaped using a <control>-V
 14382 character, that character shall not be part of a match to an *lhs*.

14383 It is unspecified whether implementations shall support **map** commands where the *lhs* is
 14384 more than a single character in length, where the first character of the *lhs* is printable.

14385 If *lhs* contains more than one character and the first character is '#', followed by a sequence
 14386 of digits corresponding to a numbered function key, then when this function key is typed it
 14387 shall be mapped to *rhs*. Characters other than digits following a '#' character also
 14388 represent the function key named by the characters in the *lhs* following the '#' and may be
 14389 mapped to *rhs*. It is unspecified how function keys are named or what function keys are
 14390 supported.

14391 For text input mode mappings:

- 14392 When the *lhs* is entered as any part of text entered in open or visual text input modes, the
14393 action shall be as if the corresponding *rhs* had been entered.
- 14394 If any character in the input text is escaped using a <control>-V character, that character shall
14395 not be part of a match to an *lhs*.
- 14396 It is unspecified whether the *lhs* argument entered for the **map** or **unmap** commands is
14397 replaced in this fashion. Regardless of whether or not the replacement occurs, the effect of
14398 the command shall be as if the replacement had not occurred.
- 14399 If only part of the *lhs* is entered, it is unspecified how long the editor will wait for additional,
14400 possibly matching characters before treating the already entered characters as not matching the
14401 *lhs*.
- 14402 The *rhs* characters shall themselves be subject to remapping, unless otherwise specified by the
14403 **remap** edit option, except that if the characters in *lhs* occur as prefix characters in *rhs*, those
14404 characters shall not be remapped.
- 14405 On block-mode terminals, the mapping need not occur immediately (for example, it may occur
14406 after the terminal transmits a group of characters to the system), but it shall achieve the same
14407 results as if it occurred immediately.
- 14408 *Current line*: Unchanged.
- 14409 *Current column*: Unchanged.
- 14410 **Mark**
- 14411 *Synopsis:* [laddr] ma[*rk*] *character*
14412 [laddr] k *character*
- 14413 Implementations shall support *character* values of a single lowercase letter of the POSIX locale
14414 and the characters ' ' and ' ' '; support of other characters is implementation-defined.
- 14415 If executing the **vi m** command, set the specified mark to the current line and 1-based numbered
14416 character referenced by the current column, if any; otherwise, column position 1.
- 14417 Otherwise, set the specified mark to the specified line and 1-based numbered first non-<blank>
14418 character in the line, if any; otherwise, the last character in the line, if any; otherwise, column
14419 position 1.
- 14420 The mark shall remain associated with the line until the mark is reset or the line is deleted. If a
14421 deleted line is restored by a subsequent **undo** command, any marks previously associated with
14422 the line, which have not been reset, shall be restored as well. Any use of a mark not associated
14423 with a current line in the edit buffer shall be an error.
- 14424 The marks ' and ' shall be set as described previously, immediately before the following events
14425 occur in the editor:
- 14426 1. The use of ' \$ ' as an *ex* address
 - 14427 2. The use of a positive decimal number as an *ex* address
 - 14428 3. The use of a search command as an *ex* address
 - 14429 4. The use of a mark reference as an *ex* address
 - 14430 5. The use of the following open and visual mode commands: <control>-[, %, (,), [,], {, }.
 - 14431 6. The use of the following open and visual mode commands: ', **G**, **H**, **L**, **M**, **z** if the current
14432 line will change as a result of the command

14433 7. The use of the open and visual mode commands: /, ?, N, ', n if the current line or column
14434 will change as a result of the command

14435 8. The use of the ex mode commands: z, undo, global, v

14436 For rules 1., 2., 3., and 4., the ' and ' marks shall not be set if the ex command is parsed as
14437 specified by rule 6.a. in **Command Line Parsing in ex** (on page 2573).

14438 For rules 5., 6., and 7., the ' and ' marks shall not be set if the commands are used as motion
14439 commands in open and visual mode.

14440 For rules 1., 2., 3., 4., 5., 6., 7., and 8., the ' and ' marks shall not be set if the command fails.

14441 The ' and ' marks shall be set as described previously, each time the contents of the edit buffer
14442 are replaced (including the editing of the initial buffer), if in open or visual mode, or if in ex
14443 mode and the edit buffer is not empty, before any commands or movements (including
14444 commands or movements specified by the -c or -t options or the +command argument) are
14445 executed on the edit buffer. If in open or visual mode, the marks shall be set as if executing the vi
14446 m command; otherwise, as if executing the ex mark command.

14447 When changing from ex mode to open or visual mode, if the ' and ' marks are not already set,
14448 the ' and ' marks shall be set as described previously.

14449 *Current line:* Unchanged.

14450 *Current column:* Unchanged.

14451 Move

14452 *Synopsis:* [2addr] m[ove] laddr [flags]

14453 Move the specified lines after the specified destination line. A destination of line zero specifies
14454 that the lines shall be placed at the beginning of the edit buffer. It shall be an error if the
14455 destination line is within the range of lines to be moved.

14456 *Current line:* Set to the last of the moved lines.

14457 *Current column:* Set to non-<blank>.

14458 Next

14459 *Synopsis:* n[ext][!][+command][file ...]

14460 If no '!' is appended to the command name, and the edit buffer has been modified since the
14461 last complete write, it shall be an error, unless the file is successfully written as specified by the
14462 **autowrite** option.

14463 If one or more files is specified:

- 14464 1. Set the argument list to the specified file names.
- 14465 2. Set the current argument list reference to be the first entry in the argument list.
- 14466 3. Set the current path name to the first file name specified.

14467 Otherwise:

- 14468 1. It shall be an error if there are no more file names in the argument list after the file name
14469 currently referenced.
- 14470 2. Set the current path name and the current argument list reference to the file name after the
14471 file name currently referenced in the argument list.

14472 Replace the contents of the edit buffer with the contents of the file named by the current path
14473 name. If for any reason the contents of the file cannot be accessed, the edit buffer shall be empty.

14474 This command shall be affected by the **autowrite** and **writeany** edit options.

14475 The *+command* option shall be <blank> character-delimited; <blank> characters can be escaped
14476 by preceding them with a backslash character. The *+command* shall be interpreted as an *ex*
14477 command immediately after the contents of the edit buffer have been replaced and the current
14478 line and column have been set.

14479 *Current line*: Set as described for the **edit** command.

14480 *Current column*: Set as described for the **edit** command.

14481 **Number**

14482 *Synopsis*: [2addr] nu[mber][count][flags]
14483 [2addr] #[count][flags]

14484 These commands shall be equivalent to the *ex* command:

14485 [2addr] p[rint][count] #[flags]

14486 See **Print** (on page 2590).

14487 **Open**

14488 *Synopsis*: [1addr] o[pen] /pattern/ [flags]

14489 This command need not be supported on block-mode terminals or terminals with insufficient
14490 capabilities. If standard input, standard output, or standard error are not terminal devices, the
14491 results are unspecified.

14492 Enter open mode.

14493 The trailing delimiter can be omitted from pattern at the end of the command line. If pattern is
14494 empty (for example, "//") or not specified, the last regular expression used in the editor shall be
14495 used as the pattern. The pattern can be delimited by slashes (shown in the Synopsis), as well as
14496 any alphanumeric, or non-<blank> character other than backslash, vertical line, double quote, or
14497 <newline> character.

14498 *Current line*: Set to the specified line.

14499 *Current column*: Set to non-<blank>.

14500 **Preserve**

14501 *Synopsis*: pre[serve]

14502 Save the edit buffer in a form that can later be recovered by using the **-r** option or by using the *ex*
14503 **recover** command. After the file has been preserved, a mail message shall be sent to the user.
14504 This message shall be readable by invoking the *mailx* utility. The message shall contain the name
14505 of the file, the time of preservation, and an *ex* command that could be used to recover the file.
14506 Additional information may be included in the mail message.

14507 *Current line*: Unchanged.

14508 *Current column*: Unchanged.

14509 **Print**14510 *Synopsis:* [2addr] p[rint][count][flags]14511 Write the addressed lines. The behavior is unspecified if the number of columns on the display is
14512 less than the number of columns required to write any single character in the lines being written.14513 Non-printable characters, except for the <tab> character, shall be written as implementation-
14514 defined multi-character sequences.14515 If the # flag is specified or the **number** edit option is set, each line shall be preceded by its line
14516 number in the following format:

14517 "%6dΔΔ", <line number>

14518 If the **l** flag is specified or the **list** edit option is set:14519 1. The characters listed in the Base Definitions volume of IEEE Std. 1003.1-200x, Table 5-1,
14520 Escape Sequences and Associated Actions shall be written as the corresponding escape
14521 sequence.14522 2. Non-printable characters not in the Base Definitions volume of IEEE Std. 1003.1-200x,
14523 Table 5-1, Escape Sequences and Associated Actions shall be written as one three-digit
14524 octal number (with a preceding backslash) for each byte in the character (most significant
14525 byte first). If the size of a byte on the system is greater than 9 bits, the format used for non-
14526 printable characters is implementation-defined.14527 3. The end of each line shall be marked with a '\$', and literal '\$' characters within the line
14528 shall be written with a preceding backslash.14529 Long lines shall be folded; the length at which folding occurs is unspecified, but should be
14530 appropriate for the output terminal, considering the number of columns of the terminal.14531 If a line is folded, and the **l** flag is not specified and the **list** edit option is not set, it is unspecified
14532 whether a multi-column character at the folding position is separated; it shall not be discarded.14533 *Current line:* Set to the last written line.14534 *Current column:* Unchanged if the current line is unchanged; otherwise, set to non-<blank>.14535 **Put**14536 *Synopsis:* [laddr] pu[t][buffer]14537 Append text from the specified buffer (by default, the unnamed buffer) to the specified line; line
14538 zero specifies that the text shall be placed at the beginning of the edit buffer. Each portion of a
14539 line in the buffer shall become a new line in the edit buffer, regardless of the mode of the buffer.14540 *Current line:* Set to the last line entered into the edit buffer.14541 *Current column:* Set to non-<blank>.14542 **Quit**14543 *Synopsis:* q[uit][!]

14544 If no '!' is appended to the command name:

14545 1. If the edit buffer has been modified since the last complete write, it shall be an error.

14546 2. If there are file names in the argument list after the file name currently referenced, and the
14547 last command was not a **quit**, **wq**, **xit**, or **ZZ** (see **Exit** (on page 3235)) command, it shall be
14548 an error.

14549 Otherwise, terminate the editing session.

14550 **Read**

14551 *Synopsis:* `[laddr] r[ead][!][file]`

14552 If '!' is not the first non-<blank> character to follow the command name, a copy of the
14553 specified file shall be appended into the edit buffer after the specified line; line zero specifies that
14554 the copy shall be placed at the beginning of the edit buffer. The number of lines and bytes read
14555 shall be written. If no *file* is named, the current path name shall be the default. If there is no
14556 current path name, then *file* shall become the current path name. If there is no current path name
14557 or *file* operand, it shall be an error. Specifying a *file* that is not of type regular shall have
14558 unspecified results.

14559 Otherwise, if *file* is preceded by '!', the rest of the line after the '!' shall have '%', '#', and
14560 '!' characters expanded as described in **Command Line Parsing in ex** (on page 2573).

14561 The *ex* utility shall then pass two arguments to the program named by the *shell* edit option; the
14562 first shall be `-c` and the second shall be the expanded arguments to the **read** command as a
14563 single argument. The standard input of the program shall be set to the standard input of the *ex*
14564 program when it was invoked. The standard error and standard output of the program shall be
14565 appended into the edit buffer after the specified line.

14566 Each line in the copied file or program output (as delimited by <newline> characters or the end
14567 of the file or output if it is not immediately preceded by a <newline> character), shall be a
14568 separate line in the edit buffer. Any occurrences of <carriage-return> and <newline> character
14569 pairs in the output shall be treated as single <newline> characters.

14570 The special meaning of the '!' following the **read** command can be overridden by escaping it
14571 with a backslash character.

14572 *Current line:* If no lines are added to the edit buffer, unchanged. Otherwise, if in open or visual
14573 mode, set to the first line entered into the edit buffer. Otherwise, set to the last line entered into
14574 the edit buffer.

14575 *Current column:* Set to non-<blank>.

14576 **Recover**

14577 *Synopsis:* `rec[over][!] file`

14578 If no '!' is appended to the command name, and the edit buffer has been modified since the
14579 last complete write, it shall be an error.

14580 If no *file* operand is specified, then the current path name shall be used. If there is no current
14581 path name or *file* operand, it shall be an error.

14582 If no recovery information has previously been saved about *file*, the **recover** command shall
14583 behave identically to the **edit** command, and an informational message to this effect shall be
14584 written.

14585 Otherwise, set the current path name to *file*, and replace the current contents of the edit buffer
14586 with the recovered contents of *file*. If there are multiple instances of the file to be recovered, the
14587 one most recently saved shall be recovered, and an informational message that there are
14588 previous versions of the file that can be recovered shall be written. The editor shall behave as if
14589 the contents of the edit buffer have already been modified.

14590 *Current file:* Set as described for the **edit** command.

14591 *Current column*: Set as described for the **edit** command.

14592 **Rewind**

14593 *Synopsis*: `rew[ind][!]`

14594 If no `'!'` is appended to the command name, and the edit buffer has been modified since the
14595 last complete write, it shall be an error, unless the file is successfully written as specified by the
14596 **autowrite** option.

14597 If the argument list is empty, it shall be an error.

14598 The current argument list reference and the current path name shall be set to the first file name
14599 in the argument list.

14600 Replace the contents of the edit buffer with the contents of the file named by the current path
14601 name. If for any reason the contents of the file cannot be accessed, the edit buffer shall be empty.

14602 This command shall be affected by the **autowrite** and **writeany** edit options.

14603 *Current line*: Set as described for the **edit** command.

14604 *Current column*: Set as described for the **edit** command.

14605 **Set**

14606 *Synopsis*: `se[t][option=[value]] ...][nooption ...][option? ...][all]`

14607 When no arguments are specified, write the value of the **term** edit option and those options
14608 whose values have been changed from the default settings; when the argument *all* is specified,
14609 write all of the option values.

14610 Giving an option name followed by the character `'?'` shall cause the current value of that
14611 option to be written. The `'?'` can be separated from the option name by zero or more `<blank>`
14612 characters. The `'?'` shall be necessary only for Boolean valued options. Boolean options can be
14613 given values by the form **set option** to turn them on or **set nooption** to turn them off; string and
14614 numeric options can be assigned by the form **set option=value**. Any `<blank>` characters in strings
14615 can be included as is by preceding each `<blank>` with an escaping backslash. More than one
14616 option can be set or listed by a single set command by specifying multiple arguments, each
14617 separated from the next by one or more `<blank>` characters.

14618 See **Edit Options in ex** (on page 2602) for details about specific options.

14619 *Current line*: Unchanged.

14620 *Current column*: Unchanged.

14621 **Shell**

14622 *Synopsis*: `sh[ell]`

14623 Invoke the program named in the **shell** edit option with the single argument `-i` (interactive
14624 mode). Editing shall be resumed when the program exits.

14625 *Current line*: Unchanged.

14626 *Current column*: Unchanged.

14627 **Source**

14628 *Synopsis:* so[urce] file

14629 Read and execute *ex* commands from *file*. Lines in the file that contain no characters or only
14630 <blank> characters shall be ignored.

14631 *Current line:* As specified for the individual *ex* commands.

14632 *Current column:* As specified for the individual *ex* commands.

14633 **Substitute**

14634 *Synopsis:* [2addr] s[substitute][/*pattern/repl*/[*options*][*count*][*flags*]]

14635 [2addr] &[*options*][*count*][*flags*]]

14636 [2addr] ~[*options*][*count*][*flags*]]

14637 Replace the first instance of the pattern *pattern* by the string *repl* on each specified line. (See
14638 **Regular Expressions in ex** (on page 2601) and **Replacement Strings in ex** (on page 2602).) Any
14639 non-alphabetic, non-<blank> delimiter other than '\\', '|', double quote, or <newline>
14640 character can be used instead of '/'. Backslash characters can be used to escape delimiters,
14641 backslash characters, and other special characters.

14642 The trailing delimiter can be omitted from *pattern* or from *repl* at the end of the command line. If
14643 both *pattern* and *repl* are not specified or are empty (for example, "//"), the last **s** command
14644 shall be repeated. If only *pattern* is not specified or is empty, the last regular expression used in
14645 the editor shall be used as the pattern. If only *repl* is not specified or is empty, the pattern shall be
14646 replaced by nothing. If the entire replacement pattern is '%', the last replacement pattern to an
14647 **s** command shall be used.

14648 Entering a <carriage-return> in *repl* (which requires an escaping backslash in *ex* mode and an
14649 escaping <control>-V in open or *vi* mode) shall split the line at that point, creating a new line in
14650 the edit buffer. The <carriage-return> shall be discarded.

14651 If options include the letter 'g' (**global**), all non-overlapping instances of the pattern in the line
14652 shall be replaced.

14653 If options includes the letter 'c' (**confirm**), then before each substitution the line shall be
14654 written; the written line shall reflect all previous substitutions. On the following line, <space>
14655 characters shall be written beneath the characters from the line that are before the *pattern* to be
14656 replaced, and '^' characters written beneath the characters included in the *pattern* to be
14657 replaced. The *ex* utility shall then wait for a response from the user. An affirmative response
14658 shall cause the substitution to be done, while any other input shall not make the substitution. An
14659 affirmative response shall consist of a line with the affirmative response (as defined by the
14660 current locale) at the beginning of the line. This line shall be subject to editing in the same way as
14661 the *ex* command line.

14662 If interrupted (see the ASYNCHRONOUS EVENTS section), any modifications confirmed by the
14663 user shall be preserved in the edit buffer after the interrupt.

14664 If the remembered search direction is not set, the **s** command shall set it to forward.

14665 In the second Synopsis, the **&** command shall repeat the previous substitution, as if the **&**
14666 command were replaced by:

14667 *s/pattern/repl/*

14668 where *pattern* and *repl* are as specified in the previous **s**, **&**, or **~** command.

14669 In the third Synopsis, the ~ command shall repeat the previous substitution, as if the '~' were
14670 replaced by:

14671 *s/pattern/repl/*

14672 where *pattern* shall be the last regular expression specified to the editor, and *repl* shall be from
14673 the previous substitution (including & and ~) command.

14674 These commands shall be affected by the *LC_MESSAGES* environment variable.

14675 *Current line*: Set to the last line in which a substitution occurred, or, unchanged if no
14676 substitution occurred.

14677 *Current column*: Set to non-<blank>.

14678 Suspend

14679 *Synopsis:* su[spend][!]
14680 st[op][!]

14681 Allow control to return to the invoking process; *ex* shall suspend itself as if it had received the
14682 SIGTSTP signal. The suspension shall occur only if job control is enabled in the invoking shell
14683 (see the description of *set -m*).

14684 These commands shall be affected by the **autowrite** and **writeany** edit options.

14685 The current **susp** character (see *stty*) shall have the same affect as the **suspend** command.

14686 Tag

14687 *Synopsis:* ta[g][!] *tagstring*

14688 The results are unspecified if the format of a tags file is not as specified by the *ctags* utility (see
14689 *ctags*) description.

14690 The **tag** command shall search for *tagstring* in the tag files referred to by the **tag** edit option, in
14691 the order they are specified, until a reference to *tagstring* is found. Files shall be searched from
14692 beginning to end. If no reference is found, it shall be an error and an error message to this effect
14693 shall be written. If the reference is not found, or if an error occurs while processing a file referred
14694 to in the **tag** edit option, it shall be an error, and an error message shall be written at the first
14695 occurrence of such an error.

14696 Otherwise, if the tags file contained a pattern, the pattern shall be treated as a regular expression
14697 used in the editor; for example, for the purposes of the **s** command.

14698 If the *tagstring* is in a file with a different name than the current path name, set the current path
14699 name to the name of that file, and replace the contents of the edit buffer with the contents of that
14700 file. In this case, if no '!' is appended to the command name, and the edit buffer has been
14701 modified since the last complete write, it shall be an error, unless the file is successfully written
14702 as specified by the **autowrite** option.

14703 This command shall be affected by the **autowrite**, **tag**, **taglength**, and **writeany** edit options.

14704 *Current line*: If the tags file contained a line number, set to that line number. If the line number is
14705 larger than the last line in the edit buffer, an error message shall be written and the current line
14706 shall be set as specified for the **edit** command.

14707 If the tags file contained a pattern, set to the first occurrence of the pattern in the file. If no
14708 matching pattern is found, an error message shall be written and the current line shall be set as
14709 specified for the **edit** command.

14710 *Current column:* If the tags file contained a line-number reference and that line-number was not
 14711 larger than the last line in the edit buffer, or if the tags file contained a pattern and that pattern
 14712 was found, set to non-<blank>. Otherwise, set as specified for the **edit** command.

14713 **Unabbreviate**

14714 *Synopsis:* una[bbrev] lhs

14715 If *lhs* is not an entry in the current list of abbreviations (see **Abbreviate** (on page 2580)), it shall
 14716 be an error. Otherwise, delete *lhs* from the list of abbreviations.

14717 *Current line:* Unchanged.

14718 *Current column:* Unchanged.

14719 **Undo**

14720 *Synopsis:* u[ndo]

14721 Reverse the changes made by the last command that modified the contents of the edit buffer,
 14722 including **undo**. For this purpose, the **global**, **v**, **open**, and **visual** commands, and commands
 14723 resulting from buffer executions and mapped character expansions, are considered single
 14724 commands.

14725 If no action that can be undone preceded the **undo** command, it shall be an error.

14726 If the **undo** command restores lines that were marked, the mark shall also be restored unless it
 14727 was reset subsequent to the deletion of the lines.

14728 *Current line:*

- 14729 1. If lines are added or changed in the file, set to the first line added or changed.
- 14730 2. Set to the line before the first line deleted, if it exists.
- 14731 3. Set to 1 if the edit buffer is not empty.
- 14732 4. Set to zero.

14733 *Current column:* Set to non-<blank>.

14734 **Unmap**

14735 *Synopsis:* unm[ap][!] lhs

14736 If '!' is appended to the command name, and if *lhs* is not an entry in the list of text input mode
 14737 map definitions, it shall be an error. Otherwise, delete *lhs* from the list of text input mode map
 14738 definitions.

14739 If no '!' is appended to the command name, and if *lhs* is not an entry in the list of command
 14740 mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of command mode
 14741 map definitions.

14742 *Current line:* Unchanged.

14743 *Current column:* Unchanged.

- 14744 **Version**
- 14745 *Synopsis:* ve[rsion]
- 14746 Write a message containing version information for the editor. The format of the message is
14747 unspecified.
- 14748 *Current line:* Unchanged.
- 14749 *Current column:* Unchanged.
- 14750 **Visual**
- 14751 *Synopsis:* [*laddr*] vi[sual][*type*][*count*][*flags*]
- 14752 If *ex* is currently in open or visual mode, the Synopsis and behavior of the visual command shall
14753 be the same as the **edit** command, as specified by **Edit** (on page 2582).
- 14754 Otherwise, this command need not be supported on block-mode terminals or terminals with
14755 insufficient capabilities. If standard input, standard output, or standard error are not terminal
14756 devices, the results are unspecified.
- 14757 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in
14758 **window** (on page 2609)). If the '^' type character was also specified, the **window** edit option
14759 shall be set before being used by the type character.
- 14760 Enter visual mode. If *type* is not specified, it shall be as if a *type* of '+' was specified. The *type*
14761 shall cause the following effects:
- 14762 + Place the beginning of the specified line at the top of the display.
- 14763 - Place the end of the specified line at the bottom of the display.
- 14764 . Place the beginning of the specified line in the middle of the display.
- 14765 ^ If the specified line is less than or equal to the value of the **window** edit option, set the line
14766 to 1; otherwise, decrement the line by the value of the **window** edit option minus 1. Place
14767 the beginning of this line as close to the bottom of the displayed lines as possible, while still
14768 displaying the value of the **window** edit option number of lines.
- 14769 *Current line:* Set to the specified line.
- 14770 *Current column:* Set to non-<blank>.
- 14771 **Write**
- 14772 *Synopsis:* [*2addr*] w[rite][!][>>][*file*]
14773 [*2addr*] w[rite][!][*file*]
14774 [*2addr*] wq[!][>>][*file*]
- 14775 If no lines are specified, the lines shall default to the entire file.
- 14776 The command **wq** shall be equivalent to a **write** command followed by a **quit** command; **wq!**
14777 shall be equivalent to **write!** followed by **quit**. In both cases, if the **write** command fails, the
14778 **quit** shall not be attempted.
- 14779 If the command name is not followed by one or more <blank> characters, or *file* is not preceded
14780 by a '!' character, the **write** shall be to a file.
- 14781 1. If the >> argument is specified, and the file already exists, the lines shall be appended to
14782 the file instead of replacing its contents. If the >> argument is specified, and the file does
14783 not already exist, it is unspecified whether the write shall proceed as if the >> argument

- 14784 had not been specified or if the write shall fail.
- 14785 2. If the **readonly** edit option is set (see **readonly** (on page 2606)), the **write** shall fail.
- 14786 3. If *file* is specified, and is not the current path name, and the file exists, the **write** shall fail.
- 14787 4. If *file* is not specified, the current path name shall be used. If there is no current path name,
14788 the **write** command shall fail.
- 14789 5. If the current path name is used, and the current path name has been changed by the **file**
14790 or **read** commands, and the file exists, the **write** shall fail. If the **write** is successful,
14791 subsequent **writes** shall not fail for this reason (unless the current path name is changed
14792 again).
- 14793 6. If the whole edit buffer is not being written, and the file to be written exists, the **write** shall
14794 fail.
- 14795 For rules 1., 2., 4., and 5., the **write** can be forced by appending the character **'!'** to the
14796 command name.
- 14797 For rules 2., 4., and 5., the **write** can be forced by setting the **writeany** edit option.
- 14798 Additional, implementation-defined tests may cause the **write** to fail.
- 14799 If the edit buffer is empty, a file without any contents shall be written.
- 14800 An informational message shall be written noting the number of lines and bytes written.
- 14801 Otherwise, if the command is followed by one or more <blank> characters, and file is preceded
14802 by **'!'**, the rest of the line after the **'!'** shall have **'%'**, **'#'**, and **'!'** characters expanded as
14803 described in **Command Line Parsing in ex** (on page 2573).
- 14804 The *ex* utility shall then pass two arguments to the program named by the **shell** edit option; the
14805 first shall be **-c** and the second shall be the expanded arguments to the **write** command as a
14806 single argument. The specified lines shall be written to the standard input of the command. The
14807 standard error and standard output of the program, if any, shall be written as described for the
14808 **print** command. If the last character in that output is not a <newline> character, a <newline>
14809 shall be written at the end of the output.
- 14810 The special meaning of the **'!'** following the **write** command can be overridden by escaping it
14811 with a backslash character.
- 14812 *Current line:* Unchanged.
- 14813 *Current column:* Unchanged.
- 14814 **Write and Exit**
- 14815 *Synopsis:* [2*addr*] x[*it*][**!**][*file*]
- 14816 If the edit buffer has not been modified since the last complete **write**, **xit** shall be equivalent to
14817 the **quit** command, or if a **'!'** is appended to the command name, to **quit!**.
- 14818 Otherwise, **xit** shall be equivalent to the **wq** command, or if a **'!'** is appended to the command
14819 name, to **wq!**.
- 14820 *Current line:* Unchanged.
- 14821 *Current line:* Unchanged.

14822 **Yank**14823 *Synopsis:* `[laddr] ya[nk][buffer][count]`14824 Copy the specified lines to the specified buffer (by default, the unnamed buffer), which shall
14825 become a line-mode buffer.14826 *Current line:* Unchanged.14827 *Current line:* Unchanged.14828 **Adjust Window**14829 *Synopsis:* `[laddr] z[!][type ...][count][flags]`14830 If no line is specified, the current line shall be the default; if *type* is omitted as well, the current
14831 line value shall first be incremented by 1. If incrementing the current line would cause it to be
14832 greater than the last line in the edit buffer, it shall be an error.14833 If there are <blank> characters between the *type* argument and the preceding *z* command name
14834 or optional '!' character, it shall be an error.14835 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in
14836 **window** (on page 2609)). If *count* is omitted, it shall default to 2 times the value of the **scroll** edit
14837 option, or if ! was specified, the number of lines in the display minus 1.14838 If *type* is omitted, then *count* lines starting with the specified line shall be written. Otherwise,
14839 *count* lines starting with the line specified by the *type* argument shall be written.14840 The *type* argument shall change the lines to be written. The possible values of *type* are as follows:

14841 – The specified line shall be decremented by the following value:

14842 $((\text{number of ``-'' characters}) \times \text{count}) - 1$ 14843 If the calculation would result in a number less than 1, it shall be an error. Write lines from
14844 the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit
14845 buffer has been written.

14846 + The specified line shall be incremented by the following value:

14847 $((\text{number of ``+'' characters}) - 1) \times \text{count} + 1$ 14848 If the calculation would result in a number greater than the last line in the edit buffer, it
14849 shall be an error. Write lines from the edit buffer, starting at the new value of line, until
14850 *count* lines or the last line in the edit buffer has been written.14851 =, . If more than a single ' .' or '=' is specified, it shall be an error. The following steps shall be
14852 taken:14853 1. If *count* is zero, nothing shall be written.14854 2. Write as many of the *N* lines before the current line in the edit buffer as exist. If *count*
14855 or '!' was specified, *N* shall be:14856 $(\text{count} - 1) / 2$ 14857 Otherwise, *N* shall be:14858 $(\text{count} - 3) / 2$ 14859 If *N* is a number less than 3, no lines shall be written.

- 14860 3. If '=' was specified as the type character, write a line consisting of the smaller of the
14861 number of columns in the display divided by two, or 40 '-' characters.
- 14862 4. Write the current line.
- 14863 5. Repeat step 3.
- 14864 6. Write as many of the *N* lines after the current line in the edit buffer as exist. *N* shall be
14865 defined as in step 2. If *N* is a number less than 3, no lines shall be written. current line
14866 in the edit buffer as exist. If count is less than 3, no lines shall be written.
- 14867 ^ The specified line shall be decremented by the following value:
14868 $((\text{number of ``^`` characters}) + 1) \times \text{count} - 1$
- 14869 If the calculation would result in a number less than 1, it shall be an error. Write lines from
14870 the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit
14871 buffer has been written.
- 14872 *Current line:* Set to the last line written, unless the type is =, in which case, set to the specified
14873 line.
- 14874 *Current column:* Set to non-<blank>.
- 14875 **Escape**
- 14876 *Synopsis:* ! *command*
14877 [*addr*] ! *command*
- 14878 The contents of the line after the '!' shall have '%', '#', and '!' characters expanded as
14879 described in **Command Line Parsing in ex** (on page 2573). If the expansion causes the text of the
14880 line to change, it shall be redisplayed, preceded by a single '!' character.
- 14881 The *ex* utility shall execute the program named by the **shell** edit option. It shall pass two
14882 arguments to the program; the first shall be **-c**, and the second shall be the expanded arguments
14883 to the ! command as a single argument.
- 14884 If no lines are specified, the standard input, standard output, and standard error of the program
14885 shall be set to the standard input, standard output, and standard error of the *ex* program when it
14886 was invoked. In addition, a warning message shall be written if the edit buffer has been
14887 modified since the last complete write, and the **warn** edit option is set.
- 14888 If lines are specified, they shall be passed to the program as standard input, and the standard
14889 output and standard error of the program shall replace those lines in the edit buffer. Each line in
14890 the program output (as delimited by <newline> characters or the end of the output if it is not
14891 immediately preceded by a <newline> character), shall be a separate line in the edit buffer. Any
14892 occurrences of <carriage-return> and <newline> character pairs in the output shall be treated as
14893 single <newline> characters. The specified lines shall be copied into the unnamed buffer before
14894 they are replaced, and the unnamed buffer shall become a line-mode buffer.
- 14895 If in *ex* mode, a single '!' character shall be written when the program completes.
- 14896 This command shall be affected by the **shell** and **warn** edit options. If no lines are specified, this
14897 command shall be affected by the **autowrite** and **writeany** edit options. If lines are specified, this
14898 command shall be affected by the **autoprint** edit option.
- 14899 *Current line:*
- 14900 1. If no lines are specified, unchanged.

- 14901 2. Otherwise, set to the last line read in, if any lines are read in.
- 14902 3. Otherwise, set to the line before the first line of the lines specified, if that line exists.
- 14903 4. Otherwise, set to the first line of the edit buffer if the edit buffer is not empty.
- 14904 5. Otherwise, set to zero.
- 14905 *Current column*: If no lines are specified, unchanged. Otherwise, set to non-<blank>.
- 14906 **Shift Left**
- 14907 *Synopsis*: [*2addr*] <[< ...][*count*][*flags*]
- 14908 Shift the specified lines to the start of the line; the number of column positions to be shifted shall be the number of command characters times the value of the **shiftwidth** edit option. Only leading <blank> characters shall be deleted or changed into other <blank> characters in shifting; other characters shall not be affected.
- 14909
- 14910
- 14911
- 14912 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode buffer.
- 14913
- 14914 This command shall be affected by the **autoprint** edit option.
- 14915 *Current line*: Set to the last line in the lines specified.
- 14916 *Current column*: Set to non-<blank>.
- 14917 **Shift Right**
- 14918 *Synopsis*: [*2addr*] >[> ...][*count*][*flags*]
- 14919 Shift the specified lines away from the start of the line; the number of column positions to be shifted shall be the number of command characters times the value of the **shiftwidth** edit option. The shift shall be accomplished by adding <blank> characters as a prefix to the line or changing leading <blank> characters into other <blank> characters. Empty lines shall not be changed.
- 14920
- 14921
- 14922
- 14923 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode buffer.
- 14924
- 14925 This command shall be affected by the **autoprint** edit option.
- 14926 *Current line*: Set to the last line in the lines specified.
- 14927 *Current column*: Set to non-<blank>.
- 14928 <control>-D
- 14929 *Synopsis*: <control>-D
- 14930 Write the next *n* lines, where *n* is the minimum of the values of the **scroll** edit option and the number of lines after the current line in the edit buffer. If the current line is the last line of the edit buffer it shall be an error.
- 14931
- 14932
- 14933 *Current line*: Set to the last line written.
- 14934 *Current column*: Set to non-<blank>.

14935 **Write Line Number**14936 *Synopsis:* [1addr] = [flags]14937 If *line* is not specified, it shall default to the last line in the edit buffer. Write the line number of
14938 the specified line.14939 *Current line:* Unchanged.14940 *Current column:* Unchanged.14941 **Execute**14942 *Synopsis:* [2addr] @ *buffer*14943 [2addr] * *buffer*14944 If no buffer is specified or is specified as '@' or '* ', the last buffer executed shall be used. If no
14945 previous buffer has been executed, it shall be an error.14946 For each line specified by the addresses, set the current line ('.') to the specified line, and
14947 execute the contents of the named *buffer* (as they were at the time the @ command was executed)
14948 as *ex* commands. For each line of a line-mode buffer, and all but the last line of a character-mode
14949 buffer, the *ex* command parser shall behave as if the line was terminated by a <newline>
14950 character.14951 If an error occurs during this process, or a line specified by the addresses does not exist when the
14952 current line would be set to it, or more than a single line was specified by the addresses, and the
14953 contents of the edit buffer are replaced (for example, by the *ex:edit* command) an error message
14954 shall be written, and no more commands resulting from the execution of this command shall be
14955 processed.14956 *Current line:* As specified for the individual *ex* commands.14957 *Current column:* As specified for the individual *ex* commands.14958 **Regular Expressions in ex**14959 The *ex* utility shall support regular expressions that are a superset of the basic regular
14960 expressions described in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 9.3, Basic
14961 Regular Expressions. A null regular expression ("//") shall be equivalent to the last regular
14962 expression encountered.14963 Regular expressions can be used in addresses to specify lines and, in some commands (for
14964 example, the **substitute** command), to specify portions of a line to be substituted.

14965 The following constructs can be used to enhance the basic regular expressions:

14966 \< < Match the beginning of a *word*. (See the definition of *word* at the beginning of **Command**
14967 **Descriptions in ex** (on page 2578).)14968 \> Match the end of a *word*.14969 ~ Match the replacement part of the last **substitute** command. The tilde ('~') character can
14970 be escaped in a regular expression to become a normal character with no special meaning.
14971 The backslash shall be discarded.14972 When the editor option **magic** is not set, the only characters with special meanings shall be '^'
14973 at the beginning of a pattern, '\$' at the end of a pattern, and '\'. The characters '.', '*',
14974 '[', and '~' shall be treated as ordinary characters unless preceded by a '\'; when preceded
14975 by a '\' they shall regain their special meaning, or in the case of backslash, be handled as a
14976 single backslash. Backslashes used to escape other characters shall be discarded.

14977 **Replacement Strings in ex**

14978 The character '&' ('\&' if the editor option **magic** is not set) in the replacement string shall
 14979 stand for the text matched by the pattern to be replaced. The character '~' ('\~' if **magic** is not
 14980 set) shall be replaced by the replacement part of the previous **substitute** command. The
 14981 sequence '\n', where *n* is an integer, shall be replaced by the text matched by the pattern
 14982 enclosed in the *n*th set of parentheses '\(' and '\)'.
 14983

14984 The strings '\l', '\u', '\L', and '\U' can be used to modify the case of elements in the
 14985 replacement string (using the '\&' or "\"digit) notation. The string '\l' ('\u') shall cause
 14986 the character that follows to be converted to lowercase (uppercase). The string '\L' ('\U')
 14987 shall cause all characters subsequent to it to be converted to lowercase (uppercase) as they are
 14988 inserted by the substitution until the string '\e' or '\E', or the end of the replacement string,
 is encountered.

14989 Otherwise, any character following a backslash shall be treated as that literal character, and the
 14990 escaping backslash shall be discarded.

14991 An example of case conversion with the **s** command is as follows:

```
14992 :p
14993 The cat sat on the mat.
14994 :s/\<.at\>/\u&/gp
14995 The Cat Sat on the Mat.
14996 :s/S\(.*\)M/S\U\1\eM/p
14997 The Cat SAT ON THE Mat.
```

14998 **Edit Options in ex**

14999 The **ex** utility has a number of options that modify its behavior. These options have default
 15000 settings, which can be changed using the **set** command.

15001 Options are Boolean unless otherwise specified.

15002 **autoindent, ai**

15003 [Default *unset*]

15004 If **autoindent** is set, each line in input mode shall be indented (using first as many <tab>
 15005 characters as possible, as determined by the editor option **tabstop**, and then using <space>
 15006 characters) to align with another line, as follows:

- 15007 1. If in open or visual mode and the text input is part of a line-oriented command (see the
 15008 EXTENDED DESCRIPTION in *vi*), align to the first column. Otherwise, if in open or
 15009 visual mode, indentation for each line shall be set as follows:
 - 15010 a. If a line was previously inserted as part of this command, it shall be set to the
 15011 indentation of the last inserted line by default, or as otherwise specified for the
 15012 <control>-D character in **Input Mode Commands in vi** (on page 3235).
 - 15013 b. Otherwise, it shall be set to the indentation of the previous current line, if any;
 15014 otherwise, to the first column.
- 15015 2. For the **ex a, i,** and **c** commands, indentation for each line shall be set as follows:
 - 15016 a. If a line was previously inserted as part of this command, it shall be set to the
 15017 indentation of the last inserted line by default, or as otherwise specified for the *eof*
 15018 character in **Scroll** (on page 2577).

- 15019 b. Otherwise, if the command is the **ex a** command, it shall be set to the line appended
15020 after, if any; otherwise to the first column.
- 15021 c. Otherwise, if the command is the **ex i** command, it shall be set to the line inserted
15022 before, if any; otherwise to the first column.
- 15023 d. Otherwise, if the command is the **ex c** command, it shall be set to the indentation of
15024 the line replaced.

15025 **autoprint, ap**

15026 [Default *set*]

15027 If **autoprint** is set, the current line shall be written after each **ex** command that modifies the
15028 contents of the current edit buffer, and after each **tag** command for which the tag search pattern
15029 was found or tag line number was valid, unless:

- 15030 1. The command was executed while in open or visual mode.
- 15031 2. The command was executed as part of a **global** or **v** command or @ buffer execution.
- 15032 3. The command was the form of the **read** command that reads a file into the edit buffer.
- 15033 4. The command was the **append**, **change**, or **insert** command.
- 15034 5. The command was not terminated by a <newline> character.
- 15035 6. The current line shall be written by a flag specified to the command; for example, **delete #**
15036 shall write the current line as specified for the flag modifier to the **delete** command, and
15037 not as specified by the **autoprint** edit option.

15038 **autowrite, aw**

15039 [Default *unset*]

15040 If **autowrite** is set, and the edit buffer has been modified since it was last completely written to
15041 any file, the contents of the edit buffer shall be written as if the **ex write** command had been
15042 specified without arguments, before each command affected by the **autowrite** edit option is
15043 executed. Appending the character '!' to the command name of any of the **ex** commands
15044 except '!' shall prevent the write. If the write fails, it shall be an error and the command shall
15045 not be executed.

15046 **beautify, bf**

15047 XSI [Default *unset*]

15048 If **beautify** is set, all non-printable characters, other than <tab>, <newline>, and <form-feed>
15049 characters, shall be discarded from text read in from files.

15050 **directory, dir**

15051 [Default *implementation-defined*]

15052 The value of this option specifies the directory in which the editor buffer is to be placed. If this
15053 directory is not writable by the user, the editor shall quit.

- 15054 **edcompatible, ed**
15055 [Default *unset*]
- 15056 Causes the presence of **g** and **c** suffixes on substitute commands to be remembered, and toggled
15057 by repeating the suffixes.
- 15058 **errorbells, eb**
15059 [Default *unset*]
- 15060 If the editor is in *ex* mode, and the terminal does not support a standout mode (such as inverse
15061 video), and **errorbells** is set, error messages shall be preceded by alerting the terminal.
- 15062 **exrc**
15063 [Default *unset*]
- 15064 If **exrc** is set, *ex* shall access any **.exrc** file in the current directory, as described in **Initialization in**
15065 **ex and vi** (on page 2569). If **exrc** is not set, *ex* shall ignore any **.exrc** file in the current directory
15066 during initialization, unless the current directory is that named by the *HOME* environment
15067 variable.
- 15068 **ignorecase, ic**
15069 [Default *unset*]
- 15070 If **ignorecase** is set, characters that have uppercase and lowercase representations shall have
15071 those representations considered as equivalent for purposes of regular expression comparison.
- 15072 The **ignorecase** edit option shall affect all remembered regular expressions; for example,
15073 unsetting the **ignorecase** edit option shall cause a subsequent *vi n* command to search for the
15074 last basic regular expression in a case-sensitive fashion.
- 15075 **lisp**
15076 [Default *unset*]
- 15077 **autoindent** mode and the (), { }, [[, and]] commands in visual mode are suitably modified for
15078 LISP code.
- 15079 **list**
15080 [Default *unset*]
- 15081 If **list** is set, edit buffer lines written while in *ex* command mode shall be written as specified for
15082 the **print** command with the **l** flag specified. In open or visual mode, each edit buffer line shall
15083 be displayed as specified for the *ex print* command with the **l** flag specified. In open or visual
15084 text input mode, when the cursor does not rest on any character in the line, it shall rest on the
15085 ' \$ ' marking the end of the line.

- 15086 **magic**
 15087 [Default *set*]
- 15088 If **magic** is set, modify the interpretation of characters in regular expressions and substitution
 15089 replacement strings (see **Regular Expressions in ex** (on page 2601) and **Replacement Strings in**
 15090 **ex** (on page 2602)).
- 15091 **mesg**
 15092 [Default *set*]
- 15093 If **mesg** is set, the permission for others to use the **write** or **talk** commands to write to the
 15094 terminal shall be turned on while in open or visual mode. The shell-level command *mesg n* shall
 15095 take precedence over any setting of the *ex mesg* option; that is, if **mesg y** was issued before the
 15096 editor started (or in a shell escape), such as:
- 15097 :!*mesg y*
- 15098 the **mesg** option in *ex* shall suppress incoming messages, but the **mesg** option shall not enable
 15099 incoming messages if **mesg n** was issued.
- 15100 **number, nu**
 15101 [Default *unset*]
- 15102 If **number** is set, edit buffer lines written while in *ex* command mode shall be written with line
 15103 numbers, in the format specified by the **print** command with the # flag specified. In *ex* text input
 15104 mode, each line shall be preceded by the line number it will have in the file.
- 15105 In open or visual mode, each edit buffer line shall be displayed with a preceding line number, in
 15106 the format specified by the *ex print* command with the # flag specified. This line number shall
 15107 not be considered part of the line for the purposes of evaluating the current column; that is,
 15108 column position 1 shall be the first column position after the format specified by the **print**
 15109 command.
- 15110 **paragraphs, para**
 15111 [Default in the POSIX locale *IPLPPPQPP LIpplpipbp*]
- 15112 The **paragraphs** edit option shall define additional paragraph boundaries for the open and visual
 15113 mode commands. The **paragraphs** edit option can be set to a character string consisting of zero
 15114 or more character pairs. It shall be an error to set it to an odd number of characters.
- 15115 **prompt**
 15116 [Default *set*]
- 15117 If **prompt** is set, *ex* command mode input shall be prompted for with a colon (':'); when unset,
 15118 no prompt shall be written.

15119 **readonly**15120 [Default *see text*]

15121 If **readonly** edit option is set, read-only mode shall be enabled (see **Write** (on page 2596)). The
 15122 **readonly** edit option shall be initialized to set if either of the following conditions are true:

- 15123 • The command-line option **-R** was specified.
- 15124 • Performing actions equivalent to the *access()* function called with the following arguments
 15125 indicates that the file lacks write permission:
 - 15126 1. The current path name is used as the *path* argument.
 - 15127 2. The constant **W_OK** is used as the *amode* argument.

15128 The **readonly** edit option may be initialized to set for other, implementation-defined reasons. |
 15129 The **readonly** edit option shall not be initialized to unset based on any special privileges of the |
 15130 user or process. The **readonly** edit option shall be reinitialized each time that the contents of the |
 15131 edit buffer are replaced (for example, by an **edit** or **next** command) unless the user has explicitly |
 15132 set it, in which case it shall remain set until the user explicitly unsets it. Once unset, it shall again |
 15133 be reinitialized each time that the contents of the edit buffer are replaced. |

15134 **redraw**15135 [Default *unset*]

15136 The editor simulates an intelligent terminal on a dumb terminal. (Since this is likely to require a
 15137 large amount of output to the terminal, it is useful only at high transmission speeds.) |

15138 **remap**15139 [Default *set*]

15140 If **remap** is set, map translation shall allow for maps defined in terms of other maps; translation
 15141 shall continue until a final product is obtained. If unset, only a one-step translation shall be done.

15142 **report**

15143 [Default 5]

15144 The value of this **report** edit option specifies what number of lines being added, copied, deleted,
 15145 or modified in the edit buffer will cause an informational message to be written to the user. The
 15146 following conditions shall cause an informational message. The message shall contain the
 15147 number of lines added, copied, deleted, or modified, but is otherwise unspecified.

- 15148 • An *ex* or *vi* editor command, other than **open**, **undo**, or **visual**, that modifies at least the value
 15149 of the **report** edit option number of lines, and which is not part of an *ex* **global** or *v*
 15150 command, or *ex* or *vi* buffer execution, shall cause an informational message to be written.
- 15151 • An *ex* **yank** or *vi* **y** or **Y** command, that copies at least the value of the **report** edit option plus
 15152 1 number of lines, and which is not part of an *ex* **global** or *v* command, or *ex* or *vi* buffer
 15153 execution, shall cause an informational message to be written.
- 15154 • An *ex* **global**, *v*, **open**, **undo**, or **visual** command or *ex* or *vi* buffer execution, that adds or
 15155 deletes a total of at least the value of the **report** edit option number of lines, and which is not
 15156 part of an *ex* **global** or *v* command, or *ex* or *vi* buffer execution, shall cause an informational
 15157 message to be written. (For example, if 3 lines were added and 8 lines deleted during an *ex*
 15158 **visual** command, 5 would be the number compared against the **report** edit option after the
 15159 command completed.

15160 **scroll, scr**

15161 [Default (number of lines in the display -1)/2]

15162 The value of the **scroll** edit option shall determine the number of lines scrolled by by the *ex*
15163 <control>-D and **z** commands. For the *vi* <control>-D and <control>-U commands, it shall be the
15164 initial number of lines to scroll when no previous <control>-D or <control>-U command has
15165 been executed.

15166 **sections**15167 [Default in the POSIX locale `NHSHH HUnhsh`]

15168 The **sections** edit option shall define additional section boundaries for the open and visual mode
15169 commands. The **sections** edit option can be set to a character string consisting of zero or more
15170 character pairs; it shall be an error to set it to an odd number of characters.

15171 **shell, sh**15172 [Default from the environment variable *SHELL*]

15173 The value of this option shall be a string. The default shall be taken from the *SHELL*
15174 environment variable. If the *SHELL* environment variable is null or empty, the *sh* (see *sh*) utility
15175 shall be the default.

15176 **shiftwidth, sw**

15177 [Default 8]

15178 The value of this option shall give the width in columns of an indentation level used during
15179 autoindentation and by the shift commands (< and >).

15180 **showmatch, sm**15181 [Default *unset*]

15182 The functionality described for the **showmatch** edit option need not be supported on block-
15183 mode terminals or terminals with insufficient capabilities.

15184 If **showmatch** is set, in open or visual mode, when a ') ' or ' } ' is typed, if the matching ' (' or
15185 ' { ' is currently visible on the display, the matching ' (' or ' { ' shall be flagged moving the
15186 cursor to its location for an unspecified amount of time.

15187 **showmode**15188 [Default *unset*]

15189 If **showmode** is set, in open or visual mode, the current mode that the editor is in shall be
15190 displayed on the last line of the display. Command mode and text input mode shall be
15191 differentiated; other unspecified modes and implementation-defined information may be
15192 displayed.

- 15193 **slowopen**
15194 [Default *unset*]
15195 If **slowopen** is set during open and visual text input modes, the editor shall not update portions
15196 of the display other than those screen columns that display the characters entered by the user
15197 (see **Input Mode Commands in vi** (on page 3235)).
- 15198 **tabstop, ts**
15199 [Default 8]
15200 The value of this edit option shall specify the column boundary used by a <tab> character in the
15201 display (see **autoprint, ap** (on page 2603) and **Input Mode Commands in vi** (on page 3235)).
- 15202 **taglength, tl**
15203 [Default zero]
15204 The value of this edit option shall specify the maximum number of characters that are
15205 considered significant in the user-specified tag name and in the tag name from the tags file. If the
15206 value is zero, all characters in both tag names shall be significant.
- 15207 **tags**
15208 [Default *see text*]
15209 The value of this edit option shall be a string of <blank> character-delimited path names of files
15210 used by the **tag** command. The default value is unspecified.
- 15211 **term**
15212 [Default from the environment variable *TERM*]
15213 The value of this edit option shall be a string. The default shall be taken from the *TERM* variable
15214 in the environment. If the *TERM* environment variable is empty or null, the default is
15215 unspecified. The editor shall use the value of this edit option to determine the type of the display
15216 device.
15217 The results are unspecified if the user changes the value of the term edit option after editor
15218 initialization.
- 15219 **terse**
15220 [Default *unset*]
15221 If **terse** is set, error messages may be less verbose. However, except for this caveat, error
15222 messages are unspecified. Furthermore, not all error messages need change for different settings
15223 of this option.
- 15224 **warn**
15225 [Default *set*]
15226 If **warn** is set, and the contents of the edit buffer have been modified since they were last
15227 completely written, the editor shall write a warning message before certain ! commands (see
15228 **Escape** (on page 2599)).

- 15229 **window**
- 15230 [Default *see text*]
- 15231 A value used in open and visual mode, by the <control>-B and <control>-F commands, and, in
15232 visual mode, to specify the number of lines displayed when the screen is repainted.
- 15233 If the **-w** command-line option is not specified, the default value shall be set to the value of the
15234 *LINES* environment variable. If the *LINES* environment variable is empty or null, the default
15235 shall be the number of lines in the display minus 1.
- 15236 Setting the **window** edit option to zero or to a value greater than the number of lines in the
15237 display minus 1 (either explicitly or based on the **-w** option or the *LINES* environment variable)
15238 shall cause the **window** edit option to be set to the number of lines in the display minus 1.
- 15239 The baud rate of the terminal line may change the default in an implementation-defined manner.
- 15240 **wrapmargin, wm**
- 15241 [Default 0]
- 15242 If the value of this edit option is zero, it shall have no effect.
- 15243 If not in the POSIX locale, the effect of this edit option is implementation-defined.
- 15244 Otherwise, it shall specify a number of columns from the ending margin of the terminal.
- 15245 During open and visual text input modes, for each character for which any part of the character
15246 is displayed in a column that is less than **wrapmargin** columns from the ending margin of the
15247 screen, the editor shall behave as follows:
- 15248 1. If the character triggering this event is a <blank> character, it, and all immediately
15249 preceding <blank> characters on the current line entered during the execution of the
15250 current text input command, shall be discarded, and the editor shall behave as if the user
15251 had entered a single <newline> character instead. In addition, if the next user-entered
15252 character is a <space> character, it shall be discarded as well.
 - 15253 2. Otherwise, if there are one or more <blank> characters on the current line immediately
15254 preceding the last group of inserted non-<blank> characters which was entered during the
15255 execution of the current text input command, the <blank> characters shall be replaced as if
15256 the user had entered a single <newline> character instead.
- 15257 If the **autoindent** edit option is set, and the events described in 1. or 2. are performed, any
15258 <blank> characters at or after the cursor in the current line shall be discarded.
- 15259 The ending margin shall be determined by the system or overridden by the user, as described for
15260 *COLUMNS* in in the ENVIRONMENT VARIABLES section and the Base Definitions volume of
15261 IEEE Std. 1003.1-200x, Chapter 8, Environment Variables.
- 15262 **wrapscan, ws**
- 15263 [Default *set*]
- 15264 If **wrapscan** is set, searches (the *ex* / or ? addresses, or open and visual mode /, ?, N, and **n**
15265 commands) shall wrap around the beginning or end of the edit buffer; when unset, searches
15266 shall stop at the beginning or end of the edit buffer.

15267 **writeany, wa**
 15268 [Default *unset*]
 15269 If **writeany** is set, some of the checks performed when executing the **ex write** commands shall be
 15270 inhibited, as described in editor option **autowrite**.

15271 **EXIT STATUS**

15272 The following exit values shall be returned:

15273 0 Successful completion.

15274 >0 An error occurred.

15275 **CONSEQUENCES OF ERRORS**

15276 When any error is encountered and the standard input is not a terminal device file, **ex** shall not
 15277 write the file or return to command or text input mode, and shall terminate with a non-zero exit
 15278 status.

15279 Otherwise, when an unrecoverable error is encountered, it shall be equivalent to a SIGHUP
 15280 asynchronous event.

15281 Otherwise, when an error is encountered, the editor shall behave as specified in **Command Line**
 15282 **Parsing in ex** (on page 2573).

15283 **APPLICATION USAGE**

15284 If a SIGSEGV signal is received while **ex** is saving a file, the file might not be successfully saved.

15285 The **next** command can accept more than one file, so usage such as:

15286 next `ls [abc]*`

15287 is valid; it would not be valid for the **edit** or **read** commands, for example, because they expect
 15288 only one file and unspecified results occur.

15289 **EXAMPLES**

15290 None.

15291 **RATIONALE**

15292 The **ex/vi** specification is based on the historical practice found in the 4 BSD and System V
 15293 implementations of **ex** and **vi**. A freely redistributable implementation of **ex/vi**, which is
 15294 tracking IEEE Std. 1003.1-200x fairly closely, and demonstrates the intended changes between
 15295 historical implementations and IEEE Std. 1003.1-200x, may be obtained by anonymous FTP
 15296 from:

15297 ftp://ftp.rdg.opengroup/pub/mirrors/nvi

15298 A *restricted editor* (both the historical *red* utility and modifications to **ex**) were considered and
 15299 rejected for inclusion. Neither option provided the level of security that users might expect.

15300 It is recognized that **ex** visual mode and related features would be difficult, if not impossible, to
 15301 implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor
 15302 addressing; thus, it is not a mandatory requirement that such features should work on all
 15303 terminals. It is the intention, however, that an **ex** implementation should provide the full set of
 15304 capabilities on all terminals capable of supporting them.

15305 **Options**

15306 The **-c** replacement for *+command* was inspired by the **-e** option of *sed*. Historically, all such
 15307 commands (see **edit** and **next** as well) were executed from the last line of the edit buffer. This
 15308 meant, for example, that `+/pattern` would fail unless the **wrapsan** option was set.
 15309 IEEE Std. 1003.1-200x requires conformance to historical practice. Historically, some
 15310 implementations restricted the *ex* commands that could be listed as part of the command line
 15311 arguments. For consistency, IEEE Std. 1003.1-200x does not permit these restrictions.

15312 In historical implementations of the editor, the **-R** option (and the **readonly** edit option) only
 15313 prevented overwriting of files; appending to files was still permitted, mapping loosely into the
 15314 *cs*h **noclobber** variable. Some implementations, however, have not followed this semantic, and
 15315 **readonly** does not permit appending either. IEEE Std. 1003.1-200x follows the latter practice,
 15316 believing that it is a more obvious and intuitive meaning of **readonly**.

15317 The **-s** option suppresses all interactive user feedback and is useful for editing scripts in batch
 15318 jobs. The list of specific effects is historical practice. The terminal type “incapable of supporting
 15319 open and visual modes” has historically been named “dumb”.

15320 The **-t** option was required because the *ctags* utility appears in IEEE Std. 1003.1-200x and the
 15321 option is available in all historical implementations of *ex*.

15322 Historically, the *ex* and *vi* utilities accepted a **-x** option, which did encryption based on the
 15323 algorithm found in the historical *crypt* utility. The **-x** option for encryption, and the associated
 15324 *crypt* utility, were omitted because the algorithm used was not specifiable and the export control
 15325 laws of some nations make it difficult to export cryptographic technology. In addition, it did not
 15326 historically provide the level of security that users might expect.

15327 **Standard Input**

15328 An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file
 15329 character, `<control>-D`, is historically an *ex* command.

15330 There was no maximum line length in historical implementations of *ex*. Specifically, as it was
 15331 parsed in chunks, the addresses had a different maximum length than the file names. Further,
 15332 the maximum line buffer size was declared as `{BUFSIZ}`, which was different lengths on
 15333 different systems. This version selected the value of `{LINE_MAX}` to impose a reasonable
 15334 restriction on portable usage of *ex* and to aid test suite writers in their development of realistic
 15335 tests that exercise this limit.

15336 **Input Files**

15337 It was an explicit decision by the standard developers that a `<newline>` character be added to
 15338 any file lacking one. It was believed that this feature of *ex* and *vi* was relied on by users in order
 15339 to make text files lacking a trailing `<newline>` more portable. It is recognized that this will
 15340 require a user-specified option or extension for implementations that permit *ex* and *vi* to edit
 15341 files of type other than text if such files are not otherwise identified by the system. It was agreed
 15342 that the ability to edit files of arbitrary type can be useful, but it was not considered necessary to
 15343 mandate that an *ex* or *vi* implementation be required to handle files other than text files.

15344 The paragraph in the INPUT FILES section, “By default, ...”, is intended to close a long-standing
 15345 security problem in *ex* and *vi*, that of the “*modeline*” or “*modelines*” edit option. This feature
 15346 allows any line in the first or last five lines of the file containing the strings `"ex:"` or `"vi:"`
 15347 (and, apparently, `"ei:"` or `"vx:"`) to be a line containing editor commands, and *ex* interprets all
 15348 the text up to the next `':'` or `<newline>` as a command. Consider the consequences, for
 15349 example, of an unsuspecting user using *ex* or *vi* as the editor when replying to a mail message in
 15350 which a line such as:

15351 `ex:! rm -rf :`

15352 appeared in the signature lines. The standard developers believed strongly that an editor should
15353 not by default interpret any lines of a file. Vendors are strongly urged to delete this feature from
15354 their implementations of *ex* and *vi*.

15355 **Asynchronous Events**

15356 The intention of the phrase “complete write” is that the entire edit buffer be written to stable
15357 storage. The note regarding temporary files is intended for implementations that use temporary
15358 files to back edit buffers unnamed by the user.

15359 Historically, SIGQUIT was ignored by *ex*, but was the equivalent of the **Q** command in visual
15360 mode; that is, it exited visual mode and entered *ex* mode. IEEE Std. 1003.1-200x permits, but does
15361 not require, this behavior. Historically, SIGINT was often used by *vi* users to terminate text
15362 input mode (<control>-C is often easier to enter than <ESC>). Some implementations of *vi*
15363 alerted the terminal on this event, and some did not. IEEE Std. 1003.1-200x requires that SIGINT
15364 behave identically to <ESC>, and that the terminal not be alerted.

15365 Historically, suspending the *ex* editor during text input mode was similar to SIGINT, as
15366 completed lines were retained, but any partial line discarded, and the editor returned to
15367 command mode. IEEE Std. 1003.1-200x is silent on this issue; implementations are encouraged to
15368 follow historical practice, where possible.

15369 Historically, the *vi* editor did not treat SIGTSTP as an asynchronous event, and it was therefore
15370 impossible to suspend the editor in visual text input mode. There are two major reasons for this.
15371 The first is that SIGTSTP is a broadcast signal on UNIX systems, and the chain of events where
15372 the shell *execs* an application that then *execs vi* usually caused confusion for the terminal state if
15373 SIGTSTP was delivered to the process group in the default manner. The second was that most
15374 implementations of the UNIX *curses* package are not reentrant, and the receipt of SIGTSTP at the
15375 wrong time will cause them to crash. IEEE Std. 1003.1-200x is silent on this issue;
15376 implementations are encouraged to treat suspension as an asynchronous event if possible.

15377 Historically, modifications to the edit buffer made before SIGINT interrupted an operation were
15378 retained; that is, anywhere from zero to all of the lines to be modified might have been modified
15379 by the time the SIGINT arrived. These changes were not discarded by the arrival of SIGINT.
15380 IEEE Std. 1003.1-200x permits this behavior, noting that the *undo* command is required to be able
15381 to undo these partially completed commands.

15382 The action taken for signals other than SIGINT, SIGCONT, SIGHUP, and SIGTERM is
15383 unspecified because some implementations attempt to save the edit buffer in a useful state when
15384 other signals are received.

15385 **Standard Error**

15386 For *ex/vi*, diagnostic messages are those messages reported as a result of a failed attempt to
15387 invoke *ex* or *vi*, such as invalid options or insufficient resources, or an abnormal termination
15388 condition. Diagnostic messages should not be confused with the error messages generated by
15389 inappropriate or illegal user commands.

15390 **Initialization in ex and vi**

15391 If an *ex* command (other than **cd**, **chdir**, or **source**) has a file name argument, one or both of the
15392 alternate and current path names will be set. Informally, they are set as follows:

- 15393 1. If the *ex* command is one that replaces the contents of the edit buffer, and it succeeds, the
15394 current path name will be set to the file name argument (the first file name argument in the
15395 case of the **next** command) and the alternate path name will be set to the previous current
15396 path name, if there was one.
- 15397 2. In the case of the file read/write forms of the **read** and **write** commands, if there is no
15398 current path name, the current path name will be set to the file name argument.
- 15399 3. Otherwise, the alternate path name will be set to the file name argument.

15400 For example, **:edit foo** and **:recover foo**, when successful, set the current path name, and, if there
15401 was a previous current path name, the alternate path name. The commands **:write**, **!command**,
15402 and **:edit** set neither the current or alternate path names. If the **:edit foo** command were to fail
15403 for some reason, the alternate path name would be set. The **read** and **write** commands set the
15404 alternate path name to their *file* argument, unless the current path name is not set, in which case
15405 they set the current path name to their *file* arguments. The alternate path name was not
15406 historically set by the **:source** command. IEEE Std. 1003.1-200x requires conformance to
15407 historical practice. Implementations adding commands that take file names as arguments are
15408 encouraged to set the alternate path name as described here.

15409 Historically, *ex* and *vi* read the **.exrc** file in the *\$HOME* directory twice, if the editor was executed
15410 in the *\$HOME* directory. IEEE Std. 1003.1-200x prohibits this behavior.

15411 Historically, the 4 BSD *ex* and *vi* read the *\$HOME* and local **.exrc** files if they were owned by the
15412 real ID of the user, or the **sourceany** option was set, regardless of other considerations. This was
15413 a security problem because it is possible to put normal UNIX system commands inside a **.exrc**
15414 file. IEEE Std. 1003.1-200x does not specify the **sourceany** option, and historical implementations
15415 are encouraged to delete it.

15416 The **.exrc** files must be owned by the real ID of the user, and not writeable by anyone other than
15417 the owner. The appropriate privileges exception is intended to permit users to acquire special
15418 privileges, but continue to use the **.exrc** files in their home directories.

15419 System V Release 3.2 and later *vi* implementations added the option **[no]exrc**. The behavior is
15420 that local **.exrc** files are read-only if the **exrc** option is set. The default for the **exrc** option was off,
15421 so by default, local **.exrc** files were not read. The problem this was intended to solve was that
15422 System V permitted users to give away files, so there is no possible ownership or writeability
15423 test to ensure that the file is safe. This is still a security problem on systems where users can give
15424 away files, but there is nothing additional that IEEE Std. 1003.1-200x can do. The
15425 implementation-defined exception is intended to permit groups to have local **.exrc** files that are
15426 shared by users, by creating pseudo-users to own the shared files.

15427 IEEE Std. 1003.1-200x does not mention system-wide *ex* and *vi* start-up files. While they exist in
15428 several implementations of *ex* and *vi*, they are not present in any implementations considered
15429 historical practice by IEEE Std. 1003.1-200x. Implementations that have such files should use
15430 them only if they are owned by the real user ID or an appropriate user (for example, root on
15431 UNIX systems) and if they are not writeable by any user other than their owner. System-wide
15432 start-up files should be read before the *EXINIT* variable, *\$HOME/.exrc* or local **.exrc** files are
15433 evaluated.

15434 Historically, any *ex* command could be entered in the *EXINIT* variable or the **.exrc** file, although
15435 ones requiring that the edit buffer already contain lines of text generally caused historical
15436 implementations of the editor to drop core. IEEE Std. 1003.1-200x requires that any *ex* command

15437 be permitted in the *EXINIT* variable and *.exrc* files, for simplicity of specification and
15438 consistency, although many of them will obviously fail under many circumstances.

15439 The initialization of the contents of the edit buffer uses the phrase “the effect shall be” with
15440 regard to various *ex* commands. The intent of this phrase is that edit buffer contents loaded
15441 during the initialization phase not be lost; that is, loading the edit buffer should fail if the *.exrc*
15442 file read in the contents of a file and did not subsequently write the edit buffer. An additional
15443 intent of this phrase is to specify that the initial current line and column is set as specified for the
15444 individual *ex* commands.

15445 Historically, the *-t* option behaved as if the tag search were a *+command*; that is, it was executed
15446 from the last line of the file specified by the tag. This resulted in the search failing if the pattern
15447 was a forward search pattern and the *wrapsan* edit option was not set. IEEE Std. 1003.1-200x
15448 does not permit this behavior, requiring that the search for the tag pattern be performed on the
15449 entire file, and, if not found, that the current line be set to a more reasonable location in the file.

15450 Historically, the empty edit buffer presented for editing when a file was not specified by the user
15451 was unnamed. This is permitted by IEEE Std. 1003.1-200x; however, implementations are
15452 encouraged to provide users a temporary file name for this buffer because it permits them the
15453 use of *ex* commands that use the current path name during temporary edit sessions.

15454 Historically, the file specified using the *-t* option was not part of the current argument list. This
15455 practice is permitted by IEEE Std. 1003.1-200x; however, implementations are encouraged to
15456 include its name in the current argument list for consistency.

15457 Historically, the *-c* command was generally not executed until a file that already exists was
15458 edited. IEEE Std. 1003.1-200x requires conformance to this historical practice. Commands that
15459 could cause the *-c* command to be executed include the *ex* commands *edit*, *next*, *recover*,
15460 *rewind*, and *tag*, and the *vi* commands *<control>-^* and *<control>-]*. Historically, reading a file
15461 into an edit buffer did not cause the *-c* command to be executed (even though it might set the
15462 current path name) with the exception that it did cause the *-c* command to be executed if: the
15463 editor was in *ex* mode, the edit buffer had no current path name, the edit buffer was empty, and
15464 no read commands had yet been attempted. For consistency and simplicity of specification,
15465 IEEE Std. 1003.1-200x does not permit this behavior.

15466 Historically, the *-r* option was the same as a normal edit session if there was no recovery
15467 information available for the file. This allowed users to enter:

```
15468 vi -r *.c
```

15469 and recover whatever files were recoverable. In some implementations, recovery was attempted
15470 only on the first file named, and the file was not entered into the argument list; in others,
15471 recovery was attempted for each file named. In addition, some historical implementations
15472 ignored *-r* if *-t* was specified or did not support command line *file* arguments with the *-t* option.
15473 For consistency and simplicity of specification, IEEE Std. 1003.1-200x disallows these special
15474 cases, and requires that recovery be attempted the first time each file is edited.

15475 Historically, *vi* initialized the ‘ and ’ marks, but *ex* did not. This meant that if the first command
15476 in *ex* mode was *visual* or if an *ex* command was executed first (for example, *vi +10 file*), *vi*
15477 was entered without the marks being initialized. Because the standard developers believed the marks
15478 to be generally useful, and for consistency and simplicity of specification, IEEE Std. 1003.1-200x
15479 requires that they always be initialized if in open or visual mode, or if in *ex* mode and the edit
15480 buffer is not empty. Not initializing it in *ex* mode if the edit buffer is empty is historical practice;
15481 however, it has always been possible to set (and use) marks in empty edit buffers in open and
15482 visual mode edit sessions.

15483 **Addressing**

15484 Historically, *ex* and *vi* accepted the additional addressing forms '`\/'` and '`\?'`. They were
 15485 equivalent to "`//`" and "`??`", respectively. They are not required by IEEE Std. 1003.1-200x,
 15486 mostly because nobody can remember whether they ever did anything different historically.

15487 Historically, *ex* and *vi* permitted an address of zero for several commands, and permitted the %
 15488 address in empty files for others. For consistency, IEEE Std. 1003.1-200x requires support for the
 15489 former in the few commands where it makes sense, and disallows it otherwise. In addition,
 15490 because IEEE Std. 1003.1-200x requires that % be logically equivalent to "`1,$`", it is also
 15491 supported where it makes sense and disallowed otherwise.

15492 Historically, the % address could not be followed by further addresses. For consistency and
 15493 simplicity of specification, IEEE Std. 1003.1-200x requires that additional addresses be
 15494 supported.

15495 All of the following are valid *addresses*:

15496 `+++` Three lines after the current line.

15497 `/re/-` One line before the next occurrence of *re*.

15498 `-2` Two lines before the current line.

15499 `3 —— 2` Line one (note intermediate negative address).

15500 `1 2 3` Line six.

15501 Any number of addresses can be provided to commands taking addresses; for example,
 15502 "`1,2,3,4,5p`" prints lines 4 and 5, because two is the greatest valid number of addresses
 15503 accepted by the **print** command. This, in combination with the semicolon delimiter, permits
 15504 users to create commands based on ordered patterns in the file. For example, the command
 15505 **3;/foo/;+2print** will display the first line after line 3 that contains the pattern *foo*, plus the next
 15506 two lines. Note that the address **3**; must be evaluated before being discarded because the search
 15507 origin for the **/foo/** command depends on this.

15508 Historically, values could be added to addresses by including them after one or more <blank>
 15509 characters; for example, **3 - 5p** wrote the seventh line of the file, and **/foo/ 5** was the same as
 15510 **/foo/+5**. However, only absolute values could be added; for example, **5 /foo/** was an error.
 15511 IEEE Std. 1003.1-200x requires conformance to historical practice. Address offsets are separately
 15512 specified from addresses because they could historically be provided to visual mode search
 15513 commands.

15514 Historically, any missing addresses defaulted to the current line. This was true for leading and
 15515 trailing comma-delimited addresses, and for trailing semicolon-delimited addresses. For
 15516 consistency, IEEE Std. 1003.1-200x requires it for leading semicolon addresses as well.

15517 Historically, *ex* and *vi* accepted the '`^`' character as both an address and as a flag offset for
 15518 commands. In both cases it was identical to the '`-`' character. IEEE Std. 1003.1-200x does not
 15519 require or prohibit this behavior.

15520 Historically, the enhancements to basic regular expressions could be used in addressing; for
 15521 example, '`~`', '`\<`', and '`\>`'. IEEE Std. 1003.1-200x requires conformance to historical
 15522 practice; that is, that regular expression usage be consistent, and that regular expression
 15523 enhancements be supported wherever regular expressions are used.

15524 **Command Line Parsing in ex**

15525 Historical *ex* command parsing was even more complex than that described here.
 15526 IEEE Std. 1003.1-200x requires the subset of the command parsing that the standard developers
 15527 believed was documented and that users could reasonably be expected to use in a portable
 15528 fashion, and that was historically consistent between implementations. (The discarded
 15529 functionality is obscure, at best.) Historical implementations will require changes in order to
 15530 comply with IEEE Std. 1003.1-200x; however, users are not expected to notice any of these
 15531 changes. Most of the complexity in *ex* parsing is to handle three special termination cases:

- 15532 1. The **!**, **global**, **v**, and the filter versions of the **read** and **write** commands are delimited by
 15533 <newline> characters (they can contain vertical-line characters that are usually shell pipes).
- 15534 2. The **ex**, **edit**, **next**, and **visual** in open and visual mode commands all take *ex* commands,
 15535 optionally containing vertical-line characters, as their first arguments.
- 15536 3. The **s** command takes a regular expression as its first argument, and uses the delimiting
 15537 characters to delimit the command.

15538 Historically, vertical-line characters in the *+command* argument of the **ex**, **edit**, **next**, **vi**, and
 15539 **visual** commands, and in the *pattern* and *replacement* parts of the **s** command, did not delimit the
 15540 command, and in the filter cases for **read** and **write**, and the **!**, **global**, and **v** commands, they did
 15541 not delimit the command at all. For example, the following commands are all valid:

```
15542 :edit +25 | s/abc/ABC/ file.c
15543 :s/ | /PIPE/
15544 :read !spell % | columnate
15545 :global/pattern/p | l
15546 :s/a/b/ | s/c/d | set
```

15547 Historically, empty or <blank> filled lines in *.exrc* files and *sourced* files (as well as *EXINIT*
 15548 variables and *ex* command scripts) were treated as default commands; that is, **print** commands.
 15549 IEEE Std. 1003.1-200x specifically requires that they be ignored when encountered in *.exrc* and
 15550 *sourced* files to eliminate a common source of new user error.

15551 Historically, *ex* commands with multiple adjacent (or <blank>-separated) vertical lines were
 15552 handled oddly when executed from *ex* mode. For example, the command `|||` <carriage-return>,
 15553 when the cursor was on line 1, displayed lines 2, 3, and 5 of the file. In addition, the command `|`
 15554 would only display the line after the next line, instead of the next two lines. The former worked
 15555 more logically when executed from *vi* mode, and displayed lines 2, 3, and 4.
 15556 IEEE Std. 1003.1-200x requires the *vi* behavior; that is, a single default command and line
 15557 number increment for each command separator, and trailing <newline> characters after
 15558 vertical-line separators are discarded.

15559 Historically, *ex* permitted a single extra colon as a leading command character; for example,
 15560 **:g/pattern/p** was a valid command. IEEE Std. 1003.1-200x generalizes this to require that any
 15561 number of leading colon characters be stripped.

15562 Historically, any prefix of the **delete** command could be followed without intervening <blank>
 15563 characters by a flag character because in the command **d p**, *p* is interpreted as the buffer *p*.
 15564 IEEE Std. 1003.1-200x requires conformance to historical practice.

15565 Historically, the **k** command could be followed by the mark name without intervening <blank>
 15566 characters. IEEE Std. 1003.1-200x requires conformance to historical practice.

15567 Historically, the **s** command could be immediately followed by flag and option characters; for
 15568 example, **s/e/E/|s|sgc3p** was a valid command. However, flag characters could not stand alone;
 15569 for example, the commands **sp** and **s l** would fail, while the command **sgp** and **s gl** would

15570 succeed. (Obviously, the '#' flag character was used as a delimiter character if it followed the
 15571 command.) Another issue was that option characters had to precede flag characters even when
 15572 the command was fully specified; for example, the command **s/e/E/pg** would fail, while the
 15573 command **s/e/E/gp** would succeed. IEEE Std. 1003.1-200x requires conformance to historical
 15574 practice.

15575 Historically, the first command name that had a prefix matching the input from the user was the
 15576 executed command; for example, **ve**, **ver**, and **vers** all executed the **version** command.
 15577 Commands were in a specific order, however, so that **a** matched **append**, not **abbreviate**.
 15578 IEEE Std. 1003.1-200x requires conformance to historical practice. The restriction on command
 15579 search order for implementations with extensions is to avoid the addition of commands such
 15580 that the historical prefixes would fail to work portably.

15581 Historical implementations of **ex** and **vi** did not correctly handle multiple **ex** commands,
 15582 separated by vertical-line characters, that entered or exited visual mode or the editor. Because
 15583 implementations of **vi** exist that do not exhibit this failure mode, IEEE Std. 1003.1-200x does not
 15584 permit it.

15585 The requirement that alphabetic command names consist of all following alphabetic characters
 15586 up to the next non-alphabetic character means that alphabetic command names must be
 15587 separated from their arguments by one or more non-alphabetic characters, normally a <blank>
 15588 or '!' character, except as specified for the exceptions, the **delete**, **k**, and **s** commands.

15589 Historically, the repeated execution of the **ex** default **print** commands (<control>-D, **eof**,
 15590 <newline>, <carriage-return>) erased any prompting character and displayed the next lines
 15591 without scrolling the terminal; that is, immediately below any previously displayed lines. This
 15592 provided a cleaner presentation of the lines in the file for the user. IEEE Std. 1003.1-200x does not
 15593 require this behavior because it may be impossible in some situations; however,
 15594 implementations are strongly encouraged to provide this semantic if possible.

15595 Historically, it was possible to change files in the middle of a command, and have the rest of the
 15596 command executed in the new file; for example:

```
15597 :edit +25 file.c | s/abc/ABC/ | 1
```

15598 was a valid command, and the substitution was attempted in the newly edited file.
 15599 IEEE Std. 1003.1-200x requires conformance to historical practice. The following commands are
 15600 examples that exercise the **ex** parser:

```
15601 echo 'foo | bar' > file1; echo 'foo/bar' > file2;
```

```
15602 vi
```

```
15603 :edit +1 | s/|/PIPE/ | w file1 | e file2 | 1 | s/\/SLASH/ | wq
```

15604 Historically, there was no protection in editor implementations to avoid **ex global**, **v**, **@**, or *****
 15605 commands changing edit buffers during execution of their associated commands. Because this
 15606 would almost invariably result in catastrophic failure of the editor, and implementations exist
 15607 that do exhibit these problems, IEEE Std. 1003.1-200x requires that changing the edit buffer
 15608 during a **global** or **v** command, or during a **@** or ***** command for which there will be more than a
 15609 single execution, be an error. Implementations supporting multiple edit buffers simultaneously
 15610 are strongly encouraged to apply the same semantics to switching between buffers as well.

15611 The **ex** command quoting required by IEEE Std. 1003.1-200x is a superset of the quoting in
 15612 historical implementations of the editor. For example, it was not historically possible to escape a
 15613 <blank> character in a file name; for example, **:edit foo\\\ bar** would report that too many file
 15614 names had been entered for the edit command, and there was no method of escaping a <blank>
 15615 in the first argument of an **edit**, **ex**, **next**, or **visual** command at all. IEEE Std. 1003.1-200x extends
 15616 historical practice, requiring that quoting behavior be made consistent across all **ex** commands,

15617 except for the **map**, **unmap**, **abbreviate**, and **unabbreviate** commands, which historically used
 15618 <control>-V instead of backslashes for quoting. For those four commands, IEEE Std. 1003.1-200x
 15619 requires conformance to historical practice.

15620 Backslash quoting in *ex* is non-intuitive. Backslash escapes are ignored unless they escape a
 15621 special character; for example, when performing *file* argument expansion, the string "\\%" is
 15622 equivalent to '\%', not "\<current path name>". This can be confusing for users because
 15623 backslash is usually one of the characters that causes shell expansion to be performed, and
 15624 therefore shell quoting rules must be taken into consideration. Generally, quoting characters are
 15625 only considered if they escape a special character, and a quoting character must be provided for
 15626 each layer of parsing for which the character is special. As another example, only a single
 15627 backslash is necessary for the '\1' sequence in substitute replacement patterns, because the
 15628 character '1' is not special to any parsing layer above it.

15629 <control>-V quoting in *ex* is slightly different from backslash quoting. In the four commands
 15630 where <control>-V quoting applies (**abbreviate**, **unabbreviate**, **map**, and **unmap**), any character
 15631 may be escaped by a <control>-V whether it would have a special meaning or not.
 15632 IEEE Std. 1003.1-200x requires conformance to historical practice.

15633 Historical implementations of the editor did not require delimiters within character classes to be
 15634 escaped; for example, the command :s/[/]// on the string "xxx/yyy" would delete the '/' from
 15635 the string. IEEE Std. 1003.1-200x disallows this historical practice for consistency and because it
 15636 places a large burden on implementations by requiring that knowledge of regular expressions be
 15637 built into the editor parser.

15638 Historically, quoting <newline> characters in *ex* commands was handled inconsistently. In most
 15639 cases, the <newline> always terminated the command, regardless of any preceding escape
 15640 character, because backslash characters did not escape <newline> characters for most *ex*
 15641 commands. However, some *ex* commands (for example, **s**, **map**, and **abbreviation**) permitted
 15642 <newline> characters to be escaped (although in the case of **map** and **abbreviation**, <control>-V
 15643 characters escaped them instead of backslashes). This was true in not only the command line,
 15644 but also **.exrc** and **sourced** files. For example, the command:

```
15645 map = foo<control-V><newline>bar
```

15646 would succeed, although it was sometimes difficult to get the <control>-V and the inserted
 15647 <newline> passed to the *ex* parser. For consistency and simplicity of specification,
 15648 IEEE Std. 1003.1-200x requires that it be possible to escape <newline> characters in *ex* commands
 15649 at all times, using backslashes for most *ex* commands, and using <control>-V characters for the
 15650 **map** and **abbreviation** commands. For example, the command **print<newline>list** is required to
 15651 be parsed as the single command **print<newline>list**. While this differs from historical practice,
 15652 IEEE Std. 1003.1-200x developers believed it unlikely that any script or user depended on the
 15653 historical behavior.

15654 Historically, an error in a command specified using the **-c** option did not cause the rest of the **-c**
 15655 commands to be discarded. IEEE Std. 1003.1-200x disallows this for consistency with mapped
 15656 keys, the **@**, **global**, **source**, and **v** commands, the *EXINIT* environment variable, and the **.exrc**
 15657 files.

15658 **Input Editing in ex**

15659 One of the common uses of the historical *ex* editor is over slow network connections. Editors
 15660 that run in canonical mode can require far less traffic to and from, and far less processing on, the
 15661 host machine, as well as more easily supporting block-mode terminals. For these reasons,
 15662 IEEE Std. 1003.1-200x requires that *ex* be implemented using canonical mode input processing,
 15663 as was done historically.

15664 IEEE Std. 1003.1-200x does not require the historical 4 BSD input editing characters “word erase”
 15665 or “literal next”. For this reason, it is unspecified how they are handled by *ex*, although they
 15666 must have the required effect. Implementations that resolve them after the line has been ended
 15667 using a <newline> or <control>-M character, and implementations that rely on the underlying
 15668 system terminal support for this processing, are both conforming. Implementations are strongly
 15669 urged to use the underlying system functionality, if at all possible, for compatibility with other
 15670 system text input interfaces.

15671 Historically, when the *eof* character was used to decrement the **autoindent** level, the cursor
 15672 moved to display the new end of the **autoindent** characters, but did not move the cursor to a
 15673 new line, nor did it erase the <control>-D character from the line. IEEE Std. 1003.1-200x does not
 15674 specify that the cursor remain on the same line or that the rest of the line is erased; however,
 15675 implementations are strongly encouraged to provide the best possible user interface; that is, the
 15676 cursor should remain on the same line, and any <control>-D character on the line should be
 15677 erased.

15678 IEEE Std. 1003.1-200x does not require the historical 4 BSD input editing character “reprint”,
 15679 traditionally <control>-R, which redisplayed the current input from the user. For this reason,
 15680 and because the functionality cannot be implemented after the line has been terminated by the
 15681 user, IEEE Std. 1003.1-200x makes no requirements about this functionality. Implementations are
 15682 strongly urged to make this historical functionality available, if possible.

15683 Historically, <control>-Q did not perform a literal next function in *ex*, as it did in *vi*.
 15684 IEEE Std. 1003.1-200x requires conformance to historical practice to avoid breaking historical *ex*
 15685 scripts and **.exrc** files.

15686 **eof**

15687 Whether the *eof* character immediately modifies the **autoindent** characters in the prompt is left
 15688 unspecified so that implementations can conform in the presence of systems that do not support
 15689 this functionality. Implementations are encouraged to modify the line and redisplay it
 15690 immediately, if possible.

15691 The specification of the handling of the *eof* character differs from historical practice only in that
 15692 *eof* characters are not discarded if they follow normal characters in the text input. Historically,
 15693 they were always discarded.

15694 **Command Descriptions in ex**

15695 Historically, several commands (for example, **global**, **v**, **visual**, **s**, **write**, **wq**, **yank**, **!**, **<**, **>**, **&**, and
 15696 **->**) were executable in empty files (that is, the default address(es) were 0), or permitted explicit
 15697 addresses of 0 (for example, 0 was a valid address, or 0,0 was a valid range). Addresses of 0, or
 15698 command execution in an empty file, make sense only for commands that add new text to the
 15699 edit buffer or write commands (because users may wish to write empty files).
 15700 IEEE Std. 1003.1-200x requires this behavior for such commands and disallows it otherwise, for
 15701 consistency and simplicity of specification.

15702 A count to an *ex* command has been historically corrected to be no greater than the last line in a
 15703 file; for example, in a five-line file, the command **1,6print** would fail, but the command **1print300**

- 15704 would succeed. IEEE Std. 1003.1-200x requires conformance to historical practice.
- 15705 Historically, the use of flags in *ex* commands could be obscure. General historical practice was as
 15706 described by IEEE Std. 1003.1-200x, but there were some special cases. For example, the **list**,
 15707 **number**, and **print** commands ignored trailing address offsets; for example, **3p** +++# would
 15708 display line 3, and 3 would be the current line after the execution of the command. The **open** and
 15709 **visual** commands ignored both the trailing offsets and the trailing flags. Also, flags specified to
 15710 the **open** and **visual** commands interacted badly with the **list** edit option, and setting and then
 15711 unsetting it during the open/visual session would cause *vi* to stop displaying lines in the
 15712 specified format. For consistency and simplicity of specification, IEEE Std. 1003.1-200x does not
 15713 permit any of these exceptions to the general rule.
- 15714 IEEE Std. 1003.1-200x uses the word *copy* in several places when discussing buffers. This is not
 15715 intended to imply implementation.
- 15716 Historically, *ex* users could not specify numeric buffers because of the ambiguity this would
 15717 cause; for example, in the command **3 delete 2**, it is unclear whether 2 is a buffer name or a
 15718 *count*. IEEE Std. 1003.1-200x requires conformance to historical practice by default, but does not
 15719 preclude extensions.
- 15720 Historically, the contents of the unnamed buffer were frequently discarded after commands that
 15721 did not explicitly affect it; for example, when using the **edit** command to switch files. For
 15722 consistency and simplicity of specification, IEEE Std. 1003.1-200x does not permit this behavior.
- 15723 The *ex* utility did not historically have access to the numeric buffers, and, furthermore, deleting
 15724 lines in *ex* did not modify their contents. For example, if, after doing a delete in *vi*, the user
 15725 switched to *ex*, did another delete, and then switched back to *vi*, the contents of the numeric
 15726 buffers would not have changed. IEEE Std. 1003.1-200x requires conformance to historical
 15727 practice. Numeric buffers are described in the *ex* utility in order to confine the description of
 15728 buffers to a single location in IEEE Std. 1003.1-200x.
- 15729 The metacharacters that trigger shell expansion in *file* arguments match historical practice, as
 15730 does the method for doing shell expansion. Implementations wishing to provide users with the
 15731 flexibility to alter the set of metacharacters are encouraged to provide a **shellmeta** string edit
 15732 option.
- 15733 Historically, *ex* commands executed from *vi* refreshed the screen when it did not strictly need to
 15734 do so; for example, **!date > /dev/null** does not require a screen refresh because the output of the
 15735 UNIX *date* command requires only a single line of the screen. IEEE Std. 1003.1-200x requires that
 15736 the screen be refreshed if it has been overwritten, but makes no requirements as to how an
 15737 implementation should make that determination. Implementations may prompt and refresh the
 15738 screen regardless.
- 15739 **Abbreviate**
- 15740 Historical practice was that characters that were entered as part of an abbreviation replacement
 15741 were subject to **map** expansions, the **showmatch** edit option, further abbreviation expansions,
 15742 and so on; that is, they were logically pushed onto the terminal input queue, and were not a
 15743 simple replacement. IEEE Std. 1003.1-200x requires conformance to historical practice.
 15744 Historical practice was that whenever a non-word character (that had not been escaped by a
 15745 <control>-V) was entered after a word character, *vi* would check for abbreviations. The check
 15746 was based on the type of the character entered before the word character of the word/non-word
 15747 pair that triggered the check. The word character of the word/non-word pair that triggered the
 15748 check and all characters entered before the trigger pair that were of that type were included in
 15749 the check, with the exception of <blank> characters, which always delimited the abbreviation.

15750 This means that, for the abbreviation to work, the *lhs* must end with a word character, there can
 15751 be no transitions from word to non-word characters (or *vice versa*) other than between the last
 15752 and next-to-last characters in the *lhs*, and there can be no <blank> characters in the *lhs*. In
 15753 addition, because of the historical quoting rules, it was impossible to enter a literal <control>-V
 15754 in the *lhs*. IEEE Std. 1003.1-200x requires conformance to historical practice. Historical
 15755 implementations did not inform users when abbreviations that could never be used were
 15756 entered; implementations are strongly encouraged to do so.

15757 For example, the following abbreviations will work:

```
15758 :ab (p REPLACE
15759 :ab p REPLACE
15760 :ab ((p REPLACE
```

15761 The following abbreviations will not work:

```
15762 :ab ( REPLACE
15763 :ab (pp REPLACE
```

15764 Historical practice is that words on the *vi* colon command line were subject to abbreviation
 15765 expansion, including the arguments to the **abbrev** (and more interestingly) the **unabbrev**
 15766 command. Because there are implementations that do not do abbreviation expansion for the first
 15767 argument to those commands, this is permitted, but not required, by IEEE Std. 1003.1-200x.
 15768 However, the following sequence:

```
15769 :ab foo bar
15770 :ab foo baz
```

15771 resulted in the addition of an abbreviation of "baz" for the string "bar" in historical *ex/vi*, and
 15772 the sequence:

```
15773 :ab foo1 bar
15774 :ab foo2 bar
15775 :unabbreviate foo2
```

15776 deleted the abbreviation "foo1", not "foo2". These behaviors are not permitted by
 15777 IEEE Std. 1003.1-200x because they clearly violate the expectations of the user.

15778 It was historical practice that <control>-V, not backslash, characters be interpreted as escaping
 15779 subsequent characters in the **abbreviate** command. IEEE Std. 1003.1-200x requires conformance
 15780 to historical practice; however, it should be noted that an abbreviation containing a <blank> will
 15781 never work.

15782 **Append**

15783 Historically, any text following a vertical-line command separator after an **append**, **change**, or
 15784 **insert** command became part of the insert text. For example, in the command:

```
15785 :g/pattern/append|stuff1
```

15786 a line containing the text "stuff1" would be appended to each line matching pattern. It was
 15787 also historically valid to enter:

```
15788 :append|stuff1
15789 stuff2
15790 .
```

15791 and the text on the *ex* command line would be appended along with the text inserted after it.
 15792 There was an historical bug, however, that the user had to enter two terminating lines (the ' .'
 15793 lines) to terminate text input mode in this case. IEEE Std. 1003.1-200x requires conformance to

15794 historical practice, but disallows the historical need for multiple terminating lines.

15795 **Change**

15796 See the RATIONALE for the **append** command. Historical practice for cursor positioning after
15797 the change command when no text is input, is as described in IEEE Std. 1003.1-200x. However,
15798 one System V implementation is known to have been modified such that the cursor is positioned
15799 on the first address specified, and not on the line before the first address. IEEE Std. 1003.1-200x
15800 disallows this modification for consistency.

15801 Historically, the **change** command did not support buffer arguments, although some
15802 implementations allow the specification of an optional buffer. This behavior is neither required
15803 nor disallowed by IEEE Std. 1003.1-200x.

15804 **Change Directory**

15805 A common extension in *ex* implementations is to use the elements of a **cdpath** edit option as
15806 prefix directories for *path* arguments to **chdir** that are relative path names and that do not have
15807 ' .' or " ." as their first component. Elements in the **cdpath** edit option are colon-separated.
15808 The initial value of the **cdpath** edit option is the value of the shell *CDPATH* environment
15809 variable. This feature was not included in IEEE Std. 1003.1-200x because it does not exist in any
15810 of the implementations considered historical practice.

15811 **Copy**

15812 Historical implementations of *ex* permitted copies to lines inside of the specified range; for
15813 example, **:2,5copy3** was a valid command. IEEE Std. 1003.1-200x requires conformance to
15814 historical practice.

15815 **Delete**

15816 IEEE Std. 1003.1-200x requires support for the historical parsing of a **delete** command followed
15817 by flags, without any intervening <blank> characters. For example:

15818 **1dp** Deletes the first line and prints the line that was second.

15819 **1delep** As for **1dp**.

15820 **1d** Deletes the first line, saving it in buffer *p*.

15821 **1d p11** (Pee-one-ell.) Deletes the first line, saving it in buffer *p*, and listing the line that was
15822 second.

15823 **Edit**

15824 Historically, any *ex* command could be entered as a *+command* argument to the **edit** command,
15825 although some (for example, **insert** and **append**) were known to confuse historical
15826 implementations. For consistency and simplicity of specification, IEEE Std. 1003.1-200x requires
15827 that any command be supported as an argument to the **edit** command.

15828 Historically, the command argument was executed with the current line set to the last line of the
15829 file, regardless of whether the **edit** command was executed from visual mode or not.
15830 IEEE Std. 1003.1-200x requires conformance to historical practice.

15831 Historically, the *+command* specified to the **edit** and **next** commands was delimited by the first
15832 <blank> character, and there was no way to quote them. For consistency, IEEE Std. 1003.1-200x
15833 requires that the usual *ex* backslash quoting be provided.

15834 Historically, specifying the *+command* argument to the edit command required a file name to be
15835 specified as well; for example, **:edit +100** would always fail. For consistency and simplicity of
15836 specification, IEEE Std. 1003.1-200x does not permit this usage to fail for that reason.

15837 Historically, only the cursor position of the last file edited was remembered by the editor.
15838 IEEE Std. 1003.1-200x requires that this be supported; however, implementations are permitted
15839 to remember and restore the cursor position for any file previously edited.

15840 **File**

15841 Historical versions of the *ex* editor **file** command displayed a current line and number of lines in
15842 the edit buffer of 0 when the file was empty, while the *vi* **<control>-G** command displayed a
15843 current line and number of lines in the edit buffer of 1 in the same situation.
15844 IEEE Std. 1003.1-200x does not permit this discrepancy, instead requiring that a message be
15845 displayed indicating that the file is empty.

15846 **Global**

15847 The two-pass operation of the **global** and **v** commands is not intended to imply implementation,
15848 only the required result of the operation.

15849 The current line and column are set as specified for the individual *ex* commands. This
15850 requirement is cumulative; that is, the current line and column must track across all the
15851 commands executed by the **global** or **v** commands.

15852 **Insert**

15853 See the RATIONALE for the **append** command.

15854 Historically, **insert** could not be used with an address of zero; that is, not when the edit buffer
15855 was empty. IEEE Std. 1003.1-200x requires that this command behave consistently with the
15856 **append** command.

15857 **Join**

15858 The action of the **join** command in relation to the special characters is only defined for the
15859 POSIX locale because the correct amount of white space after a period varies; in Japanese none is
15860 required, in French only a single space, and so on.

15861 **List**

15862 The historical output of the **list** command was potentially ambiguous. The standard developers
15863 believed correcting this to be more important than adhering to historical practice, and
15864 IEEE Std. 1003.1-200x requires unambiguous output.

15865 **Map**

15866 Historically, command mode maps only applied to command names; for example, if the
15867 character 'x' was mapped to 'y', the command **fx** searched for the 'x' character, not the 'y'
15868 character. IEEE Std. 1003.1-200x requires this behavior. Historically, entering **<control>-V** as the
15869 first character of a *vi* command was an error. Several implementations have extended the
15870 semantics of *vi* such that **<control>-V** means that the subsequent command character is not
15871 mapped. This is permitted, but not required, by IEEE Std. 1003.1-200x. Regardless, using
15872 **<control>-V** to escape the second or later character in a sequence of characters that might match
15873 a **map** command, or any character in text input mode, is historical practice, and stops the entered
15874 keys from matching a map. IEEE Std. 1003.1-200x requires conformance to historical practice.

15875 Historical implementations permitted digits to be used as a **map** command *lhs*, but then ignored
15876 the map. IEEE Std. 1003.1-200x requires that the mapped digits not be ignored.

15877 The historical implementation of the **map** command did not permit **map** commands that were
15878 more than a single character in length if the first character was printable. This behavior is
15879 permitted, but not required, by IEEE Std. 1003.1-200x.

15880 Historically, mapped characters were remapped unless the **remap** edit option was not set, or the
15881 prefix of the mapped characters matched the mapping characters; for example, in the **map**:

```
15882 :map ab abcd
```

15883 the characters "ab" were used as is and were not remapped, but the characters "cd" were
15884 mapped if appropriate. This can cause infinite loops in the *vi* mapping mechanisms.
15885 IEEE Std. 1003.1-200x requires conformance to historical practice, and that such loops be
15886 interruptible.

15887 Text input maps had the same problems with expanding the *lhs* for the **ex map!** and **unmap!**
15888 command as did the **ex abbreviate** and **unabbreviate** commands. See the RATIONALE for the **ex**
15889 **abbreviate** command. IEEE Std. 1003.1-200x requires similar modification of some historical
15890 practice for the **map** and **unmap** commands, as described for the **abbreviate** and **unabbreviate**
15891 commands.

15892 Historically, **maps** that were subsets of other **maps** behaved differently depending on the order
15893 in which they were defined. For example:

```
15894 :map! ab      short  
15895 :map! abc     long
```

15896 would always translate the characters "ab" to "short", regardless of how fast the characters
15897 "abc" were entered. If the entry order was reversed:

```
15898 :map! abc     long  
15899 :map! ab      short
```

15900 the characters "ab" would cause the editor to pause, waiting for the completing 'c' character,
15901 and the characters might never be mapped to "short". For consistency and simplicity of
15902 specification, IEEE Std. 1003.1-200x requires that the shortest match be used at all times.

15903 The length of time the editor spends waiting for the characters to complete the *lhs* is unspecified
15904 because the timing capabilities of systems are often inexact and variable, and it may depend on
15905 other factors such as the speed of the connection. The time should be long enough for the user to
15906 be able to complete the sequence, but not long enough for the user to have to wait. Some
15907 implementations of *vi* have added a **keytime** option, which permits users to set the number of
15908 0,1 seconds the editor waits for the completing characters. Because mapped terminal function
15909 and cursor keys tend to start with an <ESC> character, and <ESC> is the key ending *vi* text input
15910 mode, **maps** starting with <ESC> characters are generally exempted from this timeout period,
15911 or, at least timed out differently.

15912 **Mark**

15913 Historically, users were able to set the "previous context" marks explicitly. In addition, the **ex**
15914 commands " and " and the *vi* commands ", ", and " all referred to the same mark. In addition,
15915 the previous context marks were not set if the command, with which the address setting the
15916 mark was associated, failed. IEEE Std. 1003.1-200x requires conformance to historical practice.
15917 Historically, if marked lines were deleted, the mark was also deleted, but would reappear if the
15918 change was undone. IEEE Std. 1003.1-200x requires conformance to historical practice.

15919 The description of the special events that set the ' and ' marks matches historical practice. For
15920 example, historically the command `/a/,b/` did not set the ' and ' marks, but the command
15921 `/a/,b/delete` did.

15922 **Next**

15923 Historically, any `ex` command could be entered as a `+command` argument to the `next` command,
15924 although some (for example, `insert` and `append`) were known to confuse historical
15925 implementations. IEEE Std. 1003.1-200x requires that any command be permitted and that it
15926 behave as specified. The `next` command can accept more than one file, so usage such as:

```
15927 next `ls [abc] `
```

15928 is valid; it need not be valid for the `edit` or `read` commands, for example, because they expect
15929 only one file name.

15930 Historically, the `next` command behaved differently from the `:rewind` command in that it
15931 ignored the force flag if the `autowrite` flag was set. For consistency, IEEE Std. 1003.1-200x does
15932 not permit this behavior.

15933 Historically, the `next` command positioned the cursor as if the file had never been edited before,
15934 regardless. IEEE Std. 1003.1-200x does not permit this behavior, for consistency with the `edit`
15935 command.

15936 Implementations wanting to provide a counterpart to the `next` command that edited the
15937 previous file have used the command `prev[ious]`, which takes no `file` argument.
15938 IEEE Std. 1003.1-200x does not require this command.

15939 **Open**

15940 Historically, the `open` command would fail if the `open` edit option was not set.
15941 IEEE Std. 1003.1-200x does not mention the `open` edit option and does not require this behavior.
15942 Some historical implementations do not permit entering open mode from open or visual mode,
15943 only from `ex` mode. For consistency, IEEE Std. 1003.1-200x does not permit this behavior.

15944 Historically, entering open mode from the command line (that is, `vi +open`) resulted in
15945 anomalous behaviors; for example, the `ex` file and `set` commands, and the `vi` command
15946 `<control>-G` did not work. For consistency, IEEE Std. 1003.1-200x does not permit this behavior.

15947 Historically, the `open` command only permitted ' / ' characters to be used as the search pattern
15948 delimiter. For consistency, IEEE Std. 1003.1-200x requires that the search delimiters used by the
15949 `s`, `global`, and `v` commands be accepted as well.

15950 **Preserve**

15951 The `preserve` command does not historically cause the file to be considered unmodified for the
15952 purposes of future commands that may exit the editor. IEEE Std. 1003.1-200x requires
15953 conformance to historical practice.

15954 Historical documentation stated that mail was not sent to the user when `preserve` was executed;
15955 however, historical implementations did send mail in this case. IEEE Std. 1003.1-200x requires
15956 conformance to the historical implementations.

15957 **Print**

15958 The writing of NUL by the **print** command is not specified as a special case because the standard
 15959 developers did not want to require *ex* to support NUL characters. Historically, characters were
 15960 displayed using the ARPA standard mappings, which are as follows:

- 15961 1. Printable characters are left alone.
- 15962 2. Control characters less than \177 are represented as '^' followed by the character offset
 15963 from the '@' character in the ASCII map; for example, \007 is represented as '^G'.
- 15964 3. \177 is represented as '^' followed by '?'.

15965 The display of characters having their eighth bit set was less standard. Existing implementations
 15966 use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed bytes that had their
 15967 eighth bit set as the two characters "M-" followed by the seven-bit display as described above.)
 15968 The latter probably has the best claim to historical practice because it was used for the **-v** option
 15969 of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

15970 No specific display format is required by IEEE Std. 1003.1-200x.

15971 Explicit dependence on the ASCII character set has been avoided where possible, hence the use
 15972 of the phrase an "implementation-defined multi-character sequence" for the display of non-
 15973 printable characters in preference to the historical usage of, for instance, "^I" for the <tab>
 15974 character. Implementations are encouraged to conform to historical practice in the absence of
 15975 any strong reason to diverge.

15976 Historically, all *ex* commands beginning with the letter 'p' could be entered using capitalized
 15977 versions of the commands; for example, **P[rint]**, **Pre[serve]**, and **Pu[t]** were all valid command
 15978 names. IEEE Std. 1003.1-200x permits, but does not require, this historical practice because
 15979 capital forms of the commands are used by some implementations for other purposes.

15980 **Put**

15981 Historically, an *ex put* command, executed from open or visual mode, was the same as the open
 15982 or visual mode **P** command, if the buffer was named and was cut in character mode, and the
 15983 same as the **p** command if the buffer was named and cut in line mode. If the unnamed buffer
 15984 was the source of the text, the entire line from which the text was taken was usually **put**, and the
 15985 buffer was handled as if in line mode, but it was possible to get extremely anomalous behavior.
 15986 In addition, using the **Q** command to switch into *ex* mode, and then doing a **put** often resulted in
 15987 errors as well, such as appending text that was unrelated to the (supposed) contents of the
 15988 buffer. For consistency and simplicity of specification, IEEE Std. 1003.1-200x does not permit
 15989 these behaviors. All *ex put* commands are required to operate in line mode, and the contents of
 15990 the buffers are not altered by changing the mode of the editor.

15991 **Read**

15992 Historically, an *ex read* command executed from open or visual mode, executed in an empty file,
 15993 left an empty line as the first line of the file. For consistency and simplicity of specification,
 15994 IEEE Std. 1003.1-200x does not permit this behavior. Historically, a **read** in open or visual mode
 15995 from a program left the cursor at the last line read in, not the first. For consistency,
 15996 IEEE Std. 1003.1-200x does not permit this behavior.

15997 Historical implementations of *ex* were unable to undo **read** commands that read from the output
 15998 of a program. For consistency, IEEE Std. 1003.1-200x does not permit this behavior.

15999 Historically, the *ex* and *vi* message after a successful **read** or **write** command specified
 16000 "characters", not "bytes". IEEE Std. 1003.1-200x requires that the number of bytes be displayed,

16001 not the number of characters, because it may be difficult in multi-byte implementations to
 16002 determine the number of characters read. Implementations are encouraged to clarify the
 16003 message displayed to the user.

16004 Historically, reads were not permitted on files other than type regular, except that FIFO files
 16005 could be read (probably only because they did not exist when *ex* and *vi* were originally written).
 16006 Because the historical *ex* evaluated **read!** and **read !** equivalently, there can be no optional way
 16007 to force the read. IEEE Std. 1003.1-200x permits, but does not require, this behavior.

16008 **Recover**

16009 Some historical implementations of the editor permitted users to recover the edit buffer contents
 16010 from a previous edit session, and then exit without saving those contents (or explicitly
 16011 discarding them). The intent of IEEE Std. 1003.1-200x in requiring that the edit buffer be treated
 16012 as already modified is to prevent this user error.

16013 **Rewind**

16014 Historical implementations supported the **rewind** command when the user was editing the first
 16015 file in the list; that is, the file that the **rewind** command would edit. IEEE Std. 1003.1-200x
 16016 requires conformance to historical practice.

16017 **Substitute**

16018 Historically, *ex* accepted an **r** option to the **s** command. The effect of the **r** option was to use the
 16019 last regular expression used in any command as the pattern, the same as the **~** command. The **r**
 16020 option is not required by IEEE Std. 1003.1-200x. Historically, the **c** and **g** options were toggled;
 16021 for example, the command **:s/abc/def/** was the same as **s/abc/def/ccccgggg**. For simplicity of
 16022 specification, IEEE Std. 1003.1-200x does not permit this behavior.

16023 The tilde command is often used to replace the last search RE. For example, in the sequence:

```
16024 s/red/blue/  
16025 /green  
16026 ~
```

16027 the **~** command is equivalent to:

```
16028 s/green/blue/
```

16029 Historically, *ex* accepted all of the following forms:

```
16030 s/abc/def/  
16031 s/abc/def  
16032 s/abc/  
16033 s/abc
```

16034 IEEE Std. 1003.1-200x requires conformance to this historical practice.

16035 The **s** command presumes that the **'^'** character only occupies a single column in the display.
 16036 Much of the *ex* and *vi* specification presumes that the **<space>** character only occupies a single
 16037 column in the display. There are no known character sets for which this is not true.

16038 Historically, the final column position for the substitute commands was based on previous
 16039 column movements; a search for a pattern followed by a substitution would leave the column
 16040 position unchanged, while a **0** command followed by a substitution would change the column
 16041 position to the first non-**<blank>**. For consistency and simplicity of specification,
 16042 IEEE Std. 1003.1-200x requires that the final column position always be set to the first non-
 16043 **<blank>**.

16044

Set16045
16046

Historical implementations redisplayed all of the options for each occurrence of the **all** keyword. IEEE Std. 1003.1-200x permits, but does not require, this behavior.

16047

Tag16048
16049
16050
16051
16052
16053

No requirement is made as to where *ex* and *vi* shall look for the file referenced by the tag entry. Historical practice has been to look for the path found in the **tags** file, based on the current directory. A useful extension found in some implementations is to look based on the directory containing the tags file that held the entry, as well. No requirement is made as to which reference for the tag in the tags file is used. This is deliberate, in order to permit extensions such as multiple entries in a tags file for a tag.

16054
16055
16056
16057

Because users often specify many different tags files, some of which need not be relevant or exist at any particular time, IEEE Std. 1003.1-200x requires that error messages about problem tags files be displayed only if the requested tag is not found, and then, only once for each time that the **tag** edit option is changed.

16058
16059
16060
16061
16062
16063

The requirement that the current edit buffer be unmodified is only necessary if the file indicated by the tag entry is not the same as the current file (as defined by the current path name). Historically, the file would be reloaded if the file name had changed, as well as if the file name was different from the current path name. For consistency and simplicity of specification, IEEE Std. 1003.1-200x does not permit this behavior, requiring that the name be the only factor in the decision.

16064
16065
16066
16067

Historically, *vi* only searched for tags in the current file from the current cursor to the end of the file, and therefore, if the **wrapsan** option was not set, tags occurring before the current cursor were not found. IEEE Std. 1003.1-200x considers this a bug, and implementations are required to search for the first occurrence in the file, regardless.

16068

Undo16069
16070
16071

The **undo** description deliberately uses the word “modified”. The **undo** command is not intended to undo commands that replace the contents of the edit buffer, such as **edit**, **next**, **tag**, or **recover**.

16072
16073
16074
16075
16076

Cursor positioning after the **undo** command was inconsistent in the historical *vi*, sometimes attempting to restore the original cursor position (**global**, **undo**, and **v** commands), and sometimes, in the presence of maps, placing the cursor on the last line added or changed instead of the first. IEEE Std. 1003.1-200x requires a simplified behavior for consistency and simplicity of specification.

16077

Version16078
16079
16080

The **version** command cannot be exactly specified since there is no widely-accepted definition of what the version information should contain. Implementations are encouraged to do something reasonably intelligent.

16081 Write

16082 Historically, the *ex* and *vi* message after a successful **read** or **write** command specified
16083 “characters”, not “bytes”. IEEE Std. 1003.1-200x requires that the number of bytes be displayed,
16084 not the number of characters because it may be difficult in multi-byte implementations to
16085 determine the number of characters written. Implementations are encouraged to clarify the
16086 message displayed to the user.

16087 Implementation-defined tests are permitted so that implementations can make additional
16088 checks; for example, for locks or file modification times.

16089 Historically, attempting to append to a nonexistent file caused an error. It has been left
16090 unspecified in IEEE Std. 1003.1-200x to permit implementations to let the **write** succeed, so that
16091 the append semantics are similar to those of the historical *cs*.

16092 Historical *vi* permitted empty edit buffers to be written. However, since the way *vi* got around
16093 dealing with “empty” files was to always have a line in the edit buffer, no matter what, it wrote
16094 them as files of a single, empty line. IEEE Std. 1003.1-200x does not permit this behavior.

16095 Historically, *ex* restored standard output and standard error to their values as of when *ex* was
16096 invoked, before writes to programs were performed. This could disturb the terminal
16097 configuration as well as be a security issue for some terminals. IEEE Std. 1003.1-200x does not
16098 permit this, requiring that the program output be captured and displayed as if by the *ex* **print**
16099 command.

16100 Adjust Window

16101 Historically, the line count was set to the value of the **scroll** option if the type character was
16102 end-of-file. This feature was broken on most historical implementations long ago, however, and
16103 is not documented anywhere. For this reason, IEEE Std. 1003.1-200x is resolutely silent.

16104 Historically, the **z** command was <blank> character-sensitive and **z +** and **z -** did different
16105 things than **z+** and **z-** because the type could not be distinguished from a flag. (The commands
16106 **z .** and **z =** were historically invalid.) IEEE Std. 1003.1-200x requires conformance to this
16107 historical practice.

16108 Historically, the **z** command was further <blank> character-sensitive in that the *count* could not
16109 be <blank> character-delimited; for example, the commands **z= 5** and **z- 5** were also invalid.
16110 Because the *count* is not ambiguous with respect to either the type character or the flags, this is
16111 not permitted by IEEE Std. 1003.1-200x.

16112 Escape

16113 Historically, *ex* filter commands only read the standard output of the commands, letting
16114 standard error appear on the terminal as usual. The *vi* utility, however, read both standard
16115 output and standard error. IEEE Std. 1003.1-200x requires the latter behavior for both *ex* and *vi*,
16116 for consistency.

16117 Shift Left and Shift Right

16118 Historically, it was possible to add shift characters to increase the effect of the command; for
16119 example, <<< outdented (or >>> indented) the lines 3 levels of indentation instead of the default
16120 1. IEEE Std. 1003.1-200x requires conformance to historical practice.

16121 <control>-D

16122 Historically, the <control>-D command erased the prompt, providing the user with an unbroken
 16123 presentation of lines from the edit buffer. This is not required by IEEE Std. 1003.1-200x;
 16124 implementations are encouraged to provide it if possible. Historically, the <control>-D
 16125 command took, and then ignored, a *count*. IEEE Std. 1003.1-200x does not permit this behavior.

16126 **Write Line Number**

16127 Historically, the *ex =* command, when executed in *ex* mode in an empty edit buffer, reported 0,
 16128 and from open or visual mode, reported 1. For consistency and simplicity of specification,
 16129 IEEE Std. 1003.1-200x does not permit this behavior.

16130 **Execute**

16131 Historically, *ex* did not correctly handle the inclusion of text input commands (that is, **append**,
 16132 **insert**, and **change**) in executed buffers. IEEE Std. 1003.1-200x does not permit this exclusion for
 16133 consistency.

16134 Historically, the logical contents of the buffer being executed did not change if the buffer itself
 16135 were modified by the commands being executed; that is, buffer execution did not support self-
 16136 modifying code. IEEE Std. 1003.1-200x requires conformance to historical practice.

16137 Historically, the @ command took a range of lines, and the @ buffer was executed once per line,
 16138 with the current line (' . ') set to each specified line. IEEE Std. 1003.1-200x requires conformance
 16139 to historical practice.

16140 Some historical implementations did not notice if errors occurred during buffer execution. This,
 16141 coupled with the ability to specify a range of lines for the *ex @* command, makes it trivial to
 16142 cause them to drop core. IEEE Std. 1003.1-200x requires that implementations stop buffer
 16143 execution if any error occurs, if the specified line doesn't exist, or if the contents of the edit buffer
 16144 itself are replaced (for example, the buffer executes the *ex :edit* command).

16145 **Regular Expressions in ex**

16146 Historical practice is that the characters in the replacement part of the last *s* command—that is,
 16147 those matched by entering a '~' in the regular expression—were not further expanded by the
 16148 regular expression engine. So, if the characters contained the string "a . , " they would match
 16149 'a' followed by " . , " and not 'a' followed by any character. IEEE Std. 1003.1-200x requires
 16150 conformance to historical practice.

16151 **Edit Options in ex**

16152 The following paragraphs describe the historical behavior of some edit options that were not, for
 16153 whatever reason, included in IEEE Std. 1003.1-200x. Implementations are strongly encouraged
 16154 to only use these names if the functionality described here is fully supported.

16155 **extended** The **extended** edit option has been used in some implementations of *vi* to provide
 16156 extended regular expressions instead of basic regular expressions. This option was
 16157 omitted from IEEE Std. 1003.1-200x because it is not widespread historical practice.

16158 **flash** The **flash** edit option historically caused the screen to flash instead of beeping on
 16159 error. This option was omitted from IEEE Std. 1003.1-200x because it is not found
 16160 in some historical implementations.

16161 **hardtabs** The **hardtabs** edit option historically defined the number of columns between
 16162 hardware tab settings. This option was omitted from IEEE Std. 1003.1-200x
 16163 because it was believed to no longer be generally useful.

16164	modeline	The modeline (sometimes named modelines) edit option historically caused <i>ex</i> or <i>vi</i> to read the five first and last lines of the file for editor commands. This option is a security problem, and vendors are strongly encouraged to delete it from historical implementations.
16165		
16166		
16167		
16168	open	The open edit option historically disallowed the <i>ex</i> open and visual commands. This edit option was omitted because these commands are required by IEEE Std. 1003.1-200x.
16169		
16170		
16171	optimize	The optimize edit option historically expedited text throughput by setting the terminal to not do automatic carriage returns when printing more than one logical line of output. This option was omitted from IEEE Std. 1003.1-200x because it was intended for terminals without addressable cursors, which are rarely, if ever, still used.
16172		
16173		
16174		
16175		
16176	ruler	The ruler edit option has been used in some implementations of <i>vi</i> to present a current row/column ruler for the user. This option was omitted from IEEE Std. 1003.1-200x because it is not widespread historical practice.
16177		
16178		
16179	sourceany	The sourceany edit option historically caused <i>ex</i> or <i>vi</i> to source start-up files that were owned by users other than the user running the editor. This option is a security problem, and vendors are strongly encouraged to remove it from their implementations.
16180		
16181		
16182		
16183	timeout	The timeout edit option historically enabled the (now standard) feature of only waiting for a short period before returning keys that could be part of a macro. This feature was omitted from IEEE Std. 1003.1-200x because its behavior is now standard, it is not widely useful, and it was rarely documented.
16184		
16185		
16186		
16187	verbose	The verbose edit option has been used in some implementations of <i>vi</i> to cause <i>vi</i> to output error messages for common errors; for example, attempting to move the cursor past the beginning or end of the line instead of only alerting the screen. (The historical <i>vi</i> only alerted the terminal and presented no message for such errors. The historical editor option terse did not select when to present error messages, it only made existing error messages more or less verbose.) This option was omitted from IEEE Std. 1003.1-200x because it is not widespread historical practice; however, implementors are encouraged to use it if they wish to provide error messages for naive users.
16188		
16189		
16190		
16191		
16192		
16193		
16194		
16195		
16196	wraplen	The wraplen edit option has been used in some implementations of <i>vi</i> to specify an automatic margin measured from the left margin instead of from the right margin. This is useful when multiple screen sizes are being used to edit a single file. This option was omitted from IEEE Std. 1003.1-200x because it is not widespread historical practice; however, implementors are encouraged to use it if they add this functionality.
16197		
16198		
16199		
16200		
16201		
16202	autoindent, ai	
16203		Historically, the command Oa did not do any autoindentation, regardless of the current indentation of line 1. IEEE Std. 1003.1-200x requires that any indentation present in line 1 be used.
16204		
16205		

16206 **autoprint, ap**

16207 Historically, the **autoprint** edit option was not completely consistent or based solely on
 16208 modifications to the edit buffer. Exceptions were the **read** command (when reading from a file,
 16209 but not from a filter), the **append, change, insert, global,** and **v** commands, all of which were not
 16210 affected by **autoprint**, and the **tag** command, which was affected by **autoprint**.
 16211 IEEE Std. 1003.1-200x requires conformance to historical practice.

16212 Historically, the **autoprint** option only applied to the last of multiple commands entered using
 16213 vertical-bar delimiters; for example, **delete** <newline> was affected by **autoprint**, but
 16214 **delete|version** <newline> was not. IEEE Std. 1003.1-200x requires conformance to historical
 16215 practice.

16216 **autowrite, aw**

16217 Appending the '!' character to the **ex next** command to avoid performing an automatic write
 16218 was not supported in historical implementations. IEEE Std. 1003.1-200x requires that the
 16219 behavior match the other **ex** commands for consistency.

16220 **ignorecase, ic**

16221 Historical implementations of case-insensitive matching (the **ignorecase** edit option) lead to
 16222 counterintuitive situations when uppercase characters were used in range expressions.
 16223 Historically, the process was as follows:

- 16224 1. Take a line of text from the edit buffer.
- 16225 2. Convert uppercase to lowercase in text line.
- 16226 3. Convert uppercase to lowercase in regular expressions, except in character class
 16227 specifications.
- 16228 4. Match regular expressions against text.

16229 This would mean that, with **ignorecase** in effect, the text:

16230 The cat sat on the mat

16231 would be matched by

16232 /^the/

16233 but not by:

16234 /^[A-Z]he/

16235 For consistency with other commands implementing regular expressions, IEEE Std. 1003.1-200x
 16236 does not permit this behavior.

16237 **paragraphs, para**

16238 Earlier versions of IEEE Std. 1003.1-200x made the default **paragraphs** and **sections** edit options
 16239 implementation-defined, arguing they were historically oriented to the UNIX system *troff* text
 16240 formatter, and a “portable user” could use the { }, [[,]], (, and) commands in open or visual
 16241 mode and have the cursor stop in unexpected places. IEEE Std. 1003.1-200x specifies their values
 16242 in the POSIX locale because the unusual grouping (they only work when grouped into two
 16243 characters at a time) means that they cannot be used for general purpose movement, regardless.

- 16244 **readonly**
- 16245 Implementations are encouraged to provide the best possible information to the user as to the
16246 read-only status of the file, with the exception that they should not consider the current special
16247 privileges of the process. This provides users a safety net because they must force the overwrite
16248 of read-only files, even when running with additional privileges.
- 16249 The **readonly** edit option specification largely conforms to historical practice. The only
16250 difference is that historical implementations did not notice that the user had set the **readonly**
16251 edit option in cases where the file was already marked read-only for some reason, and would
16252 therefore reinitialize the **readonly** edit option the next time the contents of the edit buffer were
16253 replaced. This behavior is disallowed by IEEE Std. 1003.1-200x.
- 16254 **report**
- 16255 The requirement that lines copied to a buffer interact differently than deleted lines is historical
16256 practice. For example, if the **report** edit option is set to 3, deleting 3 lines will cause a report to be
16257 written, but 4 lines must be copied before a report is written.
- 16258 The requirement that the **ex global**, **v**, **open**, **undo**, and **visual** commands present reports based
16259 on the total number of lines added or deleted during the command execution, and that
16260 commands executed by the **global** and **v** commands not present reports, is historical practice.
16261 IEEE Std. 1003.1-200x extends historical practice by requiring that buffer execution be treated
16262 similarly. The reasons for this are two-fold. Historically, only the report by the last command
16263 executed from the buffer would be seen by the user, as each new report would overwrite the
16264 last. In addition, the standard developers believed that buffer execution had more in common
16265 with **global** and **v** commands than it did with other **ex** commands, and should behave similarly,
16266 for consistency and simplicity of specification.
- 16267 **showmatch, sm**
- 16268 The length of time the cursor spends on the matching character is unspecified because the
16269 timing capabilities of systems are often inexact and variable. The time should be long enough for
16270 the user to notice, but not long enough for the user to become annoyed. Some implementations
16271 of **vi** have added a **matchtime** option that permits users to set the number of 0,1 second intervals
16272 the cursor pauses on the matching character.
- 16273 **showmode**
- 16274 The **showmode** option has been used in some historical implementations of **ex** and **vi** to display
16275 the current editing mode when in open or visual mode. The editing modes have generally
16276 included “command” and “input”, and sometimes other modes such as “replace” and
16277 “change”. The string was usually displayed on the bottom line of the screen at the far right-hand
16278 corner. In addition, a preceding ‘*’ character often denoted if the contents of the edit buffer had
16279 been modified. The latter display has sometimes been part of the **showmode** option, and
16280 sometimes based on another option. This option was not available in the 4 BSD historical
16281 implementation of **vi**, but was viewed as generally useful, particularly to novice users, and is
16282 required by IEEE Std. 1003.1-200x.
- 16283 The **smd** shorthand for the **showmode** option was not present in all historical implementations
16284 of the editor. IEEE Std. 1003.1-200x requires it, for consistency.
- 16285 Not all historical implementations of the editor displayed a mode string for command mode,
16286 differentiating command mode from text input mode by the absence of a mode string.
16287 IEEE Std. 1003.1-200x permits this behavior for consistency with historical practice, but
16288 implementations are encouraged to provide a display string for both modes.

16289 **slowopen**

16290 Historically the **slowopen** option was automatically set if the terminal baud rate was less than
 16291 1 200 baud, or if the baud rate was 1 200 baud and the **redraw** option was not set. The **slowopen**
 16292 option had two effects. First, when inserting characters in the middle of a line, characters after
 16293 the cursor would not be pushed ahead, but would appear to be overwritten. Second, when
 16294 creating a new line of text, lines after the current line would not be scrolled down, but would
 16295 appear to be overwritten. In both cases, ending text input mode would cause the screen to be
 16296 refreshed to match the actual contents of the edit buffer. Finally, terminals that were sufficiently
 16297 intelligent caused the editor to ignore the **slowopen** option. IEEE Std. 1003.1-200x permits most
 16298 historical behavior, extending historical practice to require **slowopen** behaviors if the edit option
 16299 is set by the user.

16300 **tags**

16301 The default path for tags files is left unspecified as implementations may have their own **tags**
 16302 implementations that do not correspond to the historical ones. The default **tags** option value
 16303 should probably at least include the file `./tags`.

16304 **term**

16305 Historical implementations of *ex* and *vi* ignored changes to the **term** edit option after the initial
 16306 terminal information was loaded. This is permitted by IEEE Std. 1003.1-200x; however,
 16307 implementations are encouraged to permit the user to modify their terminal type at any time.

16308 **terse**

16309 Historically, the **terse** edit option optionally provided a shorter, less descriptive error message,
 16310 for some error messages. This is permitted, but not required, by IEEE Std. 1003.1-200x.
 16311 Historically, most common visual mode errors (for example, trying to move the cursor past the
 16312 end of a line) did not result in an error message, but simply alerted the terminal.
 16313 Implementations wishing to provide messages for novice users are urged to do so based on the
 16314 **edit** option **verbose**, and not **terse**.

16315 **window**

16316 In historical implementations, the default for the **window** edit option was based on the baud
 16317 rate as follows:

16318 1. If the baud rate was less than 1 200, the **edit** option **w300** set the window value; for
 16319 example, the line:

```
16320 set w300=12
```

16321 would set the window option to 12 if the baud rate was less than 1 200.

16322 2. If the baud rate was equal to 1 200, the **edit** option **w1200** set the window value.

16323 3. If the baud rate was greater than 1 200, the **edit** option **w9600** set the window value.

16324 The **w300**, **w1200**, and **w9600** options do not appear in IEEE Std. 1003.1-200x because of their
 16325 dependence on specific baud rates.

16326 In historical implementations, the size of the window displayed by various commands was
 16327 related to, but not necessarily the same as, the **window** edit option. For example, the size of the
 16328 window was set by the *ex* command **visual 10**, but it did not change the value of the **window**
 16329 edit option. However, changing the value of the **window** edit option did change the number of
 16330 lines that were displayed when the screen was repainted. IEEE Std. 1003.1-200x does not permit

16331 this behavior in the interests of consistency and simplicity of specification, and requires that all
 16332 commands that change the number of lines that are displayed do it by setting the value of the
 16333 **window** edit option.

16334 **wrapmargin, wm**

16335 Historically, the **wrapmargin** option did not affect maps inserting characters that also had
 16336 associated *counts*; for example **:map K 5aABC DEF**. Unfortunately, there are widely used
 16337 maps that depend on this behavior. For consistency and simplicity of specification,
 16338 IEEE Std. 1003.1-200x does not permit this behavior.

16339 Historically, **wrapmargin** was calculated using the column display width of all characters on the
 16340 screen. For example, an implementation using "**^I**" to represent <tab> characters when the **list**
 16341 edit option was set, where '**^**' and '**I**' each took up a single column on the screen, would
 16342 calculate the **wrapmargin** based on a value of 2 for each <tab> character. The **number** edit
 16343 option similarly changed the effective length of the line as well. IEEE Std. 1003.1-200x requires
 16344 conformance to historical practice.

16345 **FUTURE DIRECTIONS**

16346 None.

16347 **SEE ALSO**

16348 *ed, sed, stty, vi*, the System Interfaces volume of IEEE Std. 1003.1-200x, *access()*

16349 **CHANGE HISTORY**

16350 First released in Issue 2.

16351 **Issue 4**

16352 Aligned with the ISO/IEC 9945-2:1993 standard.

16353 **Issue 5**

16354 The FUTURE DIRECTIONS section is added.

16355 **Issue 6**

16356 This utility is now marked as part of the User Portability Utilities option.

16357 The obsolescent SYNOPSIS is removed, removing the *+command* and *-* options.

16358 The following new requirements on POSIX implementations derive from alignment with the
 16359 Single UNIX Specification:

- 16360 • The **-l** option is added.
- 16361 • In the *map* command description, the sequence *#digit* is added.
- 16362 • The **directory**, **edcompatible**, **redraw**, **slowopen**, and **lisp** edit options are added.

16363 The *ex* utility is extensively changed for alignment with the IEEE P1003.2b draft standard. This
 16364 includes changes as a result of the IEEE PASC Interpretations 1003.2 #31, #38, #49, #50, #51, #52,
 16365 #55, #56, #57, #61, #62, #63, #64, #65, and #78.

16366 **NAME**

16367 expand — convert tabs to spaces

16368 **SYNOPSIS**16369 UP expand [-t *tablist*][*file* ...]

16370

16371 **DESCRIPTION**

16372 The *expand* utility shall write files or the standard input to the standard output with <tab>
16373 characters replaced with one or more <space> characters needed to pad to the next tab stop. Any
16374 <backspace> characters shall be copied to the output and cause the column position count for
16375 tab stop calculations to be decremented; the column position count shall not be decremented
16376 below zero.

16377 **OPTIONS**

16378 The *expand* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
16379 12.2, Utility Syntax Guidelines.

16380 The following option shall be supported:

16381 -t *tablist* Specify the tab stops. The application shall ensure that the argument *tablist*
16382 consists of either a single positive decimal integer or a list of tabstops. If a single
16383 number is given, tabs shall be set that number of column positions apart instead of
16384 the default 8.

16385 If a list of tabstops is given, the application shall ensure that it consists of a list of
16386 two or more positive decimal integers, separated by <blank> characters or comms,
16387 in ascending order. The tabs shall be set at those specific column positions. Each
16388 tab stop *N* shall be an integer value greater than zero, and the list is in strictly
16389 ascending order. This is taken to mean that, from the start of a line of output,
16390 tabbing to position *N* shall cause the next character output to be in the (*N*+1)th
16391 column position on that line.

16392 In the event of *expand* having to process a <tab> character at a position beyond the
16393 last of those specified in a multiple tab-stop list, the <tab> character shall be
16394 replaced by a single <space> character in the output.

16395 **OPERANDS**

16396 The following operand shall be supported:

16397 *file* The path name of a text file to be used as input.

16398 **STDIN**

16399 See the INPUT FILES section.

16400 **INPUT FILES**

16401 Input files shall be text files.

16402 **ENVIRONMENT VARIABLES**

16403 The following environment variables shall affect the execution of *expand*:

16404 *LANG* Provide a default value for the internationalization variables that are unset or null.
16405 If *LANG* is unset or null, the corresponding value from the implementation-
16406 defined default locale shall be used. If any of the internationalization variables
16407 contains an invalid setting, the utility shall behave as if none of the variables had
16408 been defined.

16409 *LC_ALL* If set to a non-empty string value, override the values of all the other
16410 internationalization variables.

- 16411 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 16412 characters (for example, single-byte as opposed to multi-byte characters in
 16413 arguments and input files), the processing of <tab> and <space> characters, and
 16414 for the determination of the width in column positions each character would
 16415 occupy on an output device.
- 16416 **LC_MESSAGES**
 16417 Determine the locale that should be used to affect the format and contents of
 16418 diagnostic messages written to standard error.
- 16419 **XS1** **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 16420 **ASYNCHRONOUS EVENTS**
 16421 Default.
- 16422 **STDOUT**
 16423 The standard output shall be equivalent to the input files with <tab> characters converted into
 16424 the appropriate number of <space> characters.
- 16425 **STDERR**
 16426 Used only for diagnostic messages.
- 16427 **OUTPUT FILES**
 16428 None.
- 16429 **EXTENDED DESCRIPTION**
 16430 None.
- 16431 **EXIT STATUS**
 16432 The following exit values shall be returned:
 16433 0 Successful completion
 16434 >0 An error occurred.
- 16435 **CONSEQUENCES OF ERRORS**
 16436 The *expand* utility shall terminate with an error message and non-zero exit status upon
 16437 encountering difficulties accessing one of the *file* operands.
- 16438 **APPLICATION USAGE**
 16439 None.
- 16440 **EXAMPLES**
 16441 None.
- 16442 **RATIONALE**
 16443 The *expand* utility is useful for preprocessing text files (before sorting, looking at specific
 16444 columns, and so on) that contain <tab>s.
 16445 See the Base Definitions volume of IEEE Std. 1003.1-200x, Section 3.106, Column Position.
 16446 The *tablist* option-argument consists of integers in ascending order. Utility Syntax Guideline 8
 16447 mandates that *expand* shall accept the integers (within the single argument) separated using
 16448 either commas or <blank>s.
- 16449 **FUTURE DIRECTIONS**
 16450 None.

16451 **SEE ALSO**16452 *tabs, unexpand*16453 **CHANGE HISTORY**

16454 First released in Issue 4.

16455 **Issue 6**

16456 This utility is now marked as part of the User Portability Utilities option.

16457 The APPLICATION USAGE section is added.

16458 The obsolescent SYNOPSIS is removed.

16459 The *LC_CTYPE* environment variable description is updated to align with the IEEE P1003.2b
16460 draft standard.

16461 The normative text is reworded to avoid use of the term “must” for application requirements.

16462 **NAME**16463 *expr* — evaluate arguments as an expression16464 **SYNOPSIS**16465 *expr operand*16466 **DESCRIPTION**16467 The *expr* utility shall evaluate an expression and write the result to standard output.16468 **OPTIONS**

16469 None.

16470 **OPERANDS**

16471 The single expression evaluated by *expr* shall be formed from the operands, as described in the
 16472 EXTENDED DESCRIPTION section. The application shall ensure that each of the expression
 16473 operator symbols:

16474 () | & = > >= < <= != + - * / % :

16475 and the symbols *integer* and *string* in the table are provided as separate arguments to *expr*.16476 **STDIN**

16477 Not used.

16478 **INPUT FILES**

16479 None.

16480 **ENVIRONMENT VARIABLES**16481 The following environment variables shall affect the execution of *expr*:

16482 *LANG* Provide a default value for the internationalization variables that are unset or null.
 16483 If *LANG* is unset or null, the corresponding value from the implementation-
 16484 defined default locale shall be used. If any of the internationalization variables
 16485 contains an invalid setting, the utility shall behave as if none of the variables had
 16486 been defined.

16487 *LC_ALL* If set to a non-empty string value, override the values of all the other
 16488 internationalization variables.

16489 *LC_COLLATE*

16490 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 16491 character collating elements within regular expressions and by the string
 16492 comparison operators.

16493 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 16494 characters (for example, single-byte as opposed to multi-byte characters in
 16495 arguments) and the behavior of character classes within regular expressions.

16496 *LC_MESSAGES*

16497 Determine the locale that should be used to affect the format and contents of
 16498 diagnostic messages written to standard error.

16499 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

16500 **ASYNCHRONOUS EVENTS**

16501 Default.

16502 **STDOUT**

16503 The *expr* utility shall evaluate the expression and write the result, followed by a <newline>
 16504 character, to standard output.

16505 **STDERR**

16506 Used only for diagnostic messages.

16507 **OUTPUT FILES**

16508 None.

16509 **EXTENDED DESCRIPTION**

16510 The formation of the expression to be evaluated is shown in the following table. The symbols
 16511 *expr*, *expr1*, and *expr2* represent expressions formed from *integer* and *string* symbols and the
 16512 expression operator symbols (all separate arguments) by recursive application of the constructs
 16513 described in the table. The expressions are listed in order of increasing precedence, with equal-
 16514 precedence operators grouped between horizontal lines. All of the operators shall be left-
 16515 associative.

Expression	Description
<i>expr1</i> <i>expr2</i>	Returns the evaluation of <i>expr1</i> if it is neither null nor zero; otherwise, returns the evaluation of <i>expr2</i> if it is not null; otherwise, zero.
<i>expr1</i> & <i>expr2</i>	Returns the evaluation of <i>expr1</i> if neither expression evaluates to null or zero; otherwise, returns zero.
<i>expr1</i> = <i>expr2</i> <i>expr1</i> > <i>expr2</i> <i>expr1</i> >= <i>expr2</i> <i>expr1</i> < <i>expr2</i> <i>expr1</i> <= <i>expr2</i> <i>expr1</i> != <i>expr2</i>	Returns the result of a decimal integer comparison if both arguments are integers; otherwise, returns the result of a string comparison using the locale-specific collation sequence. The result of each comparison is 1 if the specified relationship is true, or 0 if the relationship is false. Equal. Greater than. Greater than or equal. Less than. Less than or equal. Not equal.
<i>expr1</i> + <i>expr2</i> <i>expr1</i> - <i>expr2</i>	Addition of decimal integer-valued arguments. Subtraction of decimal integer-valued arguments.
<i>expr1</i> * <i>expr2</i> <i>expr1</i> / <i>expr2</i> <i>expr1</i> % <i>expr2</i>	Multiplication of decimal integer-valued arguments. Integer division of decimal integer-valued arguments, producing an integer result. Remainder of integer division of decimal integer-valued arguments.
<i>expr1</i> : <i>expr2</i>	Matching expression; see below.
(<i>expr</i>)	Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of {EXPR_NEST_MAX}.
<i>integer</i> <i>string</i>	An argument consisting only of an (optional) unary minus followed by digits. A string argument; see below.

16547 **Matching Expression**

16548 The '=' matching operator shall compare the string resulting from the evaluation of *expr1* with
 16549 the regular expression pattern resulting from the evaluation of *expr2*. Regular expression syntax
 16550 shall be that defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 9.3, Basic
 16551 Regular Expressions, except that all patterns are anchored to the beginning of the string (that is,
 16552 only sequences starting at the first character of a string are matched by the regular expression)
 16553 and, therefore, it is unspecified whether '^' is a special character in that context. Usually, the
 16554 matching operator shall return a string representing the number of characters matched ('0' on
 16555 failure). Alternatively, if the pattern contains at least one regular expression subexpression
 16556 "[\\(\\.\\.\\.\\)]", the string corresponding to "\\1" shall be returned.

16557 **String Operand**

16558 A string argument is an argument that cannot be identified as an *integer* argument or as one of
 16559 the expression operator symbols shown in the OPERANDS section.

16560 The use of string arguments **length**, **substr**, **index**, or **match** produces unspecified results.

16561 **EXIT STATUS**

16562 The following exit values shall be returned:

16563 0 The *expression* evaluates to neither null nor zero.

16564 1 The *expression* evaluates to null or zero.

16565 2 Invalid *expression*.

16566 >2 An error occurred.

16567 **CONSEQUENCES OF ERRORS**

16568 Default.

16569 **APPLICATION USAGE**

16570 After argument processing by the shell, *expr* is not required to be able to tell the difference
 16571 between an operator and an operand except by the value. If "\$a" is '=', the command:

```
16572 expr $a = '='
```

16573 looks like:

```
16574 expr = = =
```

16575 as the arguments are passed to *expr* (and they all may be taken as the '=' operator). The
 16576 following works reliably:

```
16577 expr X$a = X=
```

16578 Also note that this volume of IEEE Std. 1003.1-200x permits implementations to extend utilities.
 16579 The *expr* utility permits the integer arguments to be preceded with a unary minus. This means
 16580 that an integer argument could look like an option. Therefore, the portable application must
 16581 employ the "—" construct of Guideline 10 of the Base Definitions volume of
 16582 IEEE Std. 1003.1-200x, Section 12.2, Utility Syntax Guidelines to protect its operands if there is
 16583 any chance the first operand might be a negative integer (or any string with a leading minus).

16584 **EXAMPLES**

16585 The *expr* utility has a rather difficult syntax:

- 16586 • Many of the operators are also shell control operators or reserved words, so they have to be
 16587 escaped on the command line.

- 16588 • Each part of the expression is composed of separate arguments, so liberal usage of <blank>
 16589 characters is required. For example:

16590

16591

16592

16593

16594

Invalid	Valid
<code>expr 1+2</code>	<code>expr 1 + 2</code>
<code>expr "1 + 2"</code>	<code>expr 1 + 2</code>
<code>expr 1 + (2 * 3)</code>	<code>expr 1 + \(2 * 3 \)</code>

16595

16596

16597

16598

In many cases, the arithmetic and string features provided as part of the shell command language are easier to use than their equivalents in *expr*. Newly written scripts should avoid *expr* in favor of the new features within the shell; see Section 2.5 (on page 2241) and Section 2.6.4 (on page 2248).

16599

The following command:

16600

```
a=$(expr $a + 1)
```

16601

adds 1 to the variable *a*.

16602

The following command, for "*\$a*" equal to either */usr/abc/file* or just *file*:

16603

```
expr $a : '.*\/(.*\)' \| $a
```

16604

16605

returns the last segment of a path name (that is, *file*). Applications should avoid the character *'/'* used alone as an argument: *expr* may interpret it as the division operator.

16606

The following command:

16607

```
expr "///$a" : '.*\/(.*\)'
```

16608

16609

16610

16611

is a better representation of the previous example. The addition of the *"/"* characters eliminates any ambiguity about the division operator and simplifies the whole expression. Also note that path names may contain characters contained in the *IFS* variable and should be quoted to avoid having "*\$a*" expand into multiple arguments.

16612

The following command:

16613

```
expr "$VAR" : '.*'
```

16614

returns the number of characters in *VAR*.

16615 RATIONALE

16616

16617

In an early proposal, EREs were used in the matching expression syntax. This was changed to BREs to avoid breaking historical applications.

16618

16619

16620

The use of a leading circumflex in the BRE is unspecified because many historical implementations have treated it as a special character, despite their system documentation. For example:

16621

```
expr foo : ^foo      expr ^foo : ^foo
```

16622

16623

16624

return 3 and 0, respectively, on those systems; their documentation would imply the reverse. Thus, the anchoring condition is left unspecified to avoid breaking historical scripts relying on this undocumented feature.

16625 FUTURE DIRECTIONS

16626

None.

16627 **SEE ALSO**

16628 Section 2.6.4

16629 **CHANGE HISTORY**

16630 First released in Issue 2.

16631 **Issue 4**

16632 Aligned with the ISO/IEC 9945-2: 1993 standard.

16633 **Issue 5**

16634 FUTURE DIRECTIONS section added.

16635 **Issue 6**16636 The *expr* utility is aligned with the IEEE P1003.2b draft standard, to include resolution of IEEE |

16637 PASC Interpretation 1003.2 #104. |

16638 The normative text is reworded to avoid use of the term “must” for application requirements.

16639 NAME

16640 false — return false value

16641 SYNOPSIS

16642 false

16643 DESCRIPTION

16644 The *false* utility shall return with a non-zero exit code.

16645 OPTIONS

16646 None.

16647 OPERANDS

16648 None.

16649 STDIN

16650 Not used.

16651 INPUT FILES

16652 None.

16653 ENVIRONMENT VARIABLES

16654 None.

16655 ASYNCHRONOUS EVENTS

16656 Default.

16657 STDOUT

16658 Not used.

16659 STDERR

16660 None.

16661 OUTPUT FILES

16662 None.

16663 EXTENDED DESCRIPTION

16664 None.

16665 EXIT STATUS

16666 The *false* utility always shall exit with a value other than zero.

16667 CONSEQUENCES OF ERRORS

16668 Default.

16669 APPLICATION USAGE

16670 None.

16671 EXAMPLES

16672 None.

16673 RATIONALE

16674 None.

16675 FUTURE DIRECTIONS

16676 None.

16677 SEE ALSO

16678 *true*

16679 **CHANGE HISTORY**

16680 First released in Issue 2.

16681 **Issue 4**

16682 Aligned with the ISO/IEC 9945-2: 1993 standard.

16683 NAME

16684 fc — process the command history list

16685 SYNOPSIS

16686 UP `fc [-r][-e editor] [first[last]]`16687 `fc -l[-nr] [first[last]]`16688 `fc -s[old=new][first]`

16689

16690 DESCRIPTION

16691 The *fc* utility shall list, or shall edit and re-execute, commands previously entered to an
16692 interactive *sh*.

16693 The command history list shall reference commands by number. The first number in the list is
16694 selected arbitrarily. The relationship of a number to its command shall not change except when
16695 the user logs in and no other process is accessing the list, at which time the system may reset the
16696 numbering to start the oldest retained command at another number (usually 1). When the
16697 number reaches an implementation-defined upper limit, which shall be no smaller than the
16698 value in *HISTSIZE* or 32 767 (whichever is greater), the shell may wrap the numbers, starting the
16699 next command with a lower number (usually 1). However, despite this optional wrapping of
16700 numbers, *fc* shall maintain the time-ordering sequence of the commands. For example, if four
16701 commands in sequence are given the numbers 32 766, 32 767, 1 (wrapped), and 2 as they are
16702 executed, command 32 767 is considered the command previous to 1, even though its number is
16703 higher.

16704 When commands are edited (when the *-l* option is not specified), the resulting lines shall be
16705 entered at the end of the history list and then re-executed by *sh*. The *fc* command that caused the
16706 editing shall not be entered into the history list. If the editor returns a non-zero exit status, this
16707 shall suppress the entry into the history list and the command re-execution. Any command line
16708 variable assignments or redirection operators used with *fc* shall affect both the *fc* command itself
16709 as well as the command that results; for example:

16710 `fc -s -- -l 2>/dev/null`16711 reinvokes the previous command, suppressing standard error for both *fc* and the previous
16712 command.

16713 OPTIONS

16714 The *fc* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
16715 Utility Syntax Guidelines.

16716 The following options shall be supported:

16717 **-e editor** Use the editor named by *editor* to edit the commands. The *editor* string is a utility
16718 name, subject to search via the *PATH* variable (see the Base Definitions volume of
16719 IEEE Std. 1003.1-200x, Chapter 8, Environment Variables). The value in the *FCEDIT*
16720 variable shall be used as a default when *-e* is not specified. If *FCEDIT* is null or
16721 unset, *ed* shall be used as the editor.

16722 **-l** (The letter ell.) List the commands rather than invoking an editor on them. The
16723 commands shall be written in the sequence indicated by the *first* and *last* operands,
16724 as affected by *-r*, with each command preceded by the command number.

16725 **-n** Suppress command numbers when listing with *-l*.

16726 **-r** Reverse the order of the commands listed (with *-l*) or edited (with neither *-l* nor
16727 *-s*).

- 16728 -s Reexecute the command without invoking an editor.
- 16729 **OPERANDS**
- 16730 The following operands shall be supported:
- 16731 *first, last*
- 16732 Select the commands to list or edit. The number of previous commands that can be
16733 accessed shall be determined by the value of the *HISTSIZE* variable. The value of
16734 *first* or *last* or both shall be one of the following:
- 16735 [+]*number* A positive number representing a command number; command
16736 numbers can be displayed with the -l option.
- 16737 -*number* A negative decimal number representing the command that was
16738 executed *number* of commands previously. For example, -1 is the
16739 immediately previous command.
- 16740 *string* A string indicating the most recently entered command that begins
16741 with that string. If the *old=new* operand is not also specified with -s,
16742 the string form of the *first* operand cannot contain an embedded
16743 equal sign.
- 16744 When the synopsis form with -s is used:
- 16745
 - If *first* is omitted, the previous command shall be used.
- 16746 For the synopsis forms without -s:
- 16747
 - If *last* is omitted, *last* shall default to the previous command when -l is
16748 specified; otherwise, it shall default to *first*.
 - If *first* and *last* are both omitted, the previous 16 commands shall be listed or
16750 the previous single command shall be edited (based on the -l option).
 - If *first* and *last* are both present, all of the commands from *first* to *last* shall be
16751 edited (without -l) or listed (with -l). Editing multiple commands shall be
16752 accomplished by presenting to the editor all of the commands at one time, each
16753 command starting on a new line. If *first* represents a newer command than *last*,
16754 the commands shall be listed or edited in reverse sequence, equivalent to using
16755 -r. For example, the following commands on the first line are equivalent to the
16756 corresponding commands on the second:
16757

```
16758                            fc -r 10 20     fc     30 40
16759                            fc     20 10     fc -r 40 30
```

16760
 - When a range of commands is used, it shall not be an error to specify *first* or *last*
16761 values that are not in the history list; *fc* shall substitute the value representing
16762 the oldest or newest command in the list, as appropriate. For example, if there
16763 are only ten commands in the history list, numbered 1 to 10:

```
16764                            fc -l
16765                            fc 1 99
```

16766 shall list and edit, respectively, all ten commands.

16767 *old=new* Replace the first occurrence of string *old* in the commands to be re-executed by the
16768 string *new*.

16769 **STDIN**

16770 Not used.

16771 **INPUT FILES**

16772 None.

16773 **ENVIRONMENT VARIABLES**16774 The following environment variables shall affect the execution of *fc*:

16775 **FCEDIT** This variable, when expanded by the shell, shall determine the default value for
 16776 the *-e editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be
 16777 used as the editor.

16778 **HISTFILE** Determine a path name naming a command history file. If the *HISTFILE* variable is
 16779 not set, the shell may attempt to access or create a file *.sh_history* in the directory
 16780 referred to by the *HOME* environment variable. If the shell cannot obtain both read
 16781 and write access to, or create, the history file, it shall use an unspecified
 16782 mechanism that allows the history to operate properly. (References to history
 16783 "file" in this section shall be understood to mean this unspecified mechanism in
 16784 such cases.) An implementation may choose to access this variable only when
 16785 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt
 16786 to retrieve entries from, or add entries to, the file, as the result of commands issued
 16787 by the user, the file named by the *ENV* variable, or implementation-defined system
 16788 start-up files. In some historical shells, the history file is initialized just after the
 16789 *ENV* file has been processed. Therefore, it is implementation-defined whether
 16790 changes made to *HISTFILE* after the history file has been initialized are effective.
 16791 Implementations may choose to disable the history list mechanism for users with
 16792 appropriate privileges who do not set *HISTFILE*; the specific circumstances under
 16793 which this occurs are implementation-defined. If more than one instance of the
 16794 shell is using the same history file, it is unspecified how updates to the history file
 16795 from those shells interact. As entries are deleted from the history file, they shall be
 16796 deleted oldest first. It is unspecified when history file entries are physically
 16797 removed from the history file.

16798 **HISTSIZE** Determine a decimal number representing the limit to the number of previous
 16799 commands that are accessible. If this variable is unset, an unspecified default
 16800 greater than or equal to 128 shall be used. The maximum number of commands in
 16801 the history list is unspecified, but shall be at least 128. An implementation may
 16802 choose to access this variable only when initializing the history file, as described
 16803 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE*
 16804 after the history file has been initialized are effective.

16805 **LANG** Provide a default value for the internationalization variables that are unset or null.
 16806 If *LANG* is unset or null, the corresponding value from the implementation-
 16807 defined default locale shall be used. If any of the internationalization variables
 16808 contains an invalid setting, the utility shall behave as if none of the variables had
 16809 been defined.

16810 **LC_ALL** If set to a non-empty string value, override the values of all the other
 16811 internationalization variables.

16812 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 16813 characters (for example, single-byte as opposed to multi-byte characters in
 16814 arguments and input files).

16815 **LC_MESSAGES**

16816 Determine the locale that should be used to affect the format and contents of

- 16817 diagnostic messages written to standard error.
- 16818 XSI **NLSPATH** Determine the location of message catalogs for the processing of `LC_MESSAGES`.
- 16819 **ASYNCHRONOUS EVENTS**
- 16820 Default.
- 16821 **STDOUT**
- 16822 When the `-l` option is used to list commands, the format of each command in the list shall be as
- 16823 follows:
- 16824 `"%d\t%s\n", <line number>, <command>`
- 16825 If both the `-l` and `-n` options are specified, the format of each command shall be:
- 16826 `"\t%s\n", <command>`
- 16827 If the `<command>` consists of more than one line, the lines after the first shall be displayed as:
- 16828 `"\t%s\n", <continued-command>`
- 16829 **STDERR**
- 16830 Used only for diagnostic messages.
- 16831 **OUTPUT FILES**
- 16832 None.
- 16833 **EXTENDED DESCRIPTION**
- 16834 None.
- 16835 **EXIT STATUS**
- 16836 The following exit values shall be returned:
- 16837 0 Successful completion of the listing.
- 16838 >0 An error occurred.
- 16839 Otherwise, the exit status shall be that of the commands executed by `fc`.
- 16840 **CONSEQUENCES OF ERRORS**
- 16841 Default.
- 16842 **APPLICATION USAGE**
- 16843 Since editors sometimes use file descriptors as integral parts of their editing, redirecting their file
- 16844 descriptors as part of the `fc` command can produce unexpected results. For example, if `vi` is the
- 16845 `FCEDIT` editor, the command:
- 16846 `fc -s | more`
- 16847 does not work correctly on many systems.
- 16848 Users on windowing systems may want to have separate history files for each window by
- 16849 setting `HISTFILE` as follows:
- 16850 `HISTFILE=$HOME/.sh_hist$$`
- 16851 **EXAMPLES**
- 16852 None.
- 16853 **RATIONALE**
- 16854 This utility is based on the `fc` built-in of the KornShell.
- 16855 An early proposal specified the `-e` option as `[-e editor [old= new]]`, which is not historical
- 16856 practice. Historical practice in `fc` of either `[-e editor]` or `[-e - [old= new]]` is acceptable, but not

16857 both together. To clarify this, a new option `-s` was introduced replacing the `[-e -]`. This resolves
16858 the conflict and makes `fc` conform to the Utility Syntax Guidelines.

16859 **HISTFILE** Some implementations of the KornShell check for the superuser and do not create
16860 a history file unless *HISTFILE* is set. This is done primarily to avoid creating
16861 unlinked files in the root file system when logging in during single-user mode.
16862 *HISTFILE* must be set for the superuser to have history.

16863 **HISTSIZE** Needed to limit the size of history files. It is the intent of the standard developers
16864 that when two shells share the same history file, commands that are entered in one
16865 shell shall be accessible by the other shell. Because of the difficulties of
16866 synchronization over a network, the exact nature of the interaction is unspecified.

16867 The initialization process for the history file can be dependent on the system start-up files, in
16868 that they may contain commands that effectively preempt the settings the user has for *HISTFILE*
16869 and *HISTSIZE*. For example, function definition commands are recorded in the history file. If the
16870 system administrator includes function definitions in some system start-up file called before the
16871 *ENV* file, the history file is initialized before the user can influence its characteristics. In some
16872 historical shells, the history file is initialized just after the *ENV* file has been processed. Because
16873 of these situations, the text requires the initialization process to be implementation-defined.

16874 Consideration was given to omitting the `fc` utility in favor of the command line editing feature in
16875 *sh*. For example, in *vi* editing mode, typing "`<ESC> v`" is equivalent to:

```
16876 EDITOR=vi fc
```

16877 However, the `fc` utility allows the user the flexibility to edit multiple commands simultaneously
16878 (such as `fc 10 20`) and to use editors other than those supported by *sh* for command line editing.

16879 In the KornShell, the alias `r` ("re-do") is preset to `fc -e -` (equivalent to the POSIX `fc -s`). This is
16880 probably an easier command name to remember than `fc` ("fix command"), but it does not meet
16881 the Utility Syntax Guidelines. Renaming `fc` to *hist* or *redo* was considered, but since this
16882 description closely matches historical KornShell practice already, such a renaming was seen as
16883 gratuitous. Users are free to create aliases whenever odd historical names such as *fc*, *awk*, *cat*,
16884 *grep*, or *yacc* are standardized by POSIX.

16885 Command numbers have no ordering effects; they are like serial numbers. The `-r` option and
16886 `-number` operand address the sequence of command execution, regardless of serial numbers. So,
16887 for example, if the command number wrapped back to 1 at some arbitrary point, there would be
16888 no ambiguity associated with traversing the wrap point. For example, if the command history
16889 were:

```
16890 32766: echo 1
16891 32767: echo 2
16892 1: echo 3
```

16893 the number `-2` refers to command 32 767 because it is the second previous command, regardless
16894 of serial number.

16895 **FUTURE DIRECTIONS**

16896 None.

16897 **SEE ALSO**

16898 *sh*

16899 **CHANGE HISTORY**

16900 First released in Issue 4.

16901 **Issue 5**

16902 FUTURE DIRECTIONS section added.

16903 **Issue 6**

16904 This utility is now marked as part of the User Portability Utilities option.

16905 In the ENVIRONMENT VARIABLES section, the text “user’s home directory” is updated to
16906 “directory referred to by the *HOME* environment variable”.

16907 **NAME**

16908 fg — run jobs in the foreground

16909 **SYNOPSIS**16910 UP fg [*job_id*]

16911

16912 **DESCRIPTION**16913 If job control is enabled (see the description of *set -m*), the *fg* utility shall move a background job
16914 from the current environment (see Section 2.13 (on page 2273)) into the foreground.16915 Using *fg* to place a job into the foreground shall remove its process ID from the list of those
16916 “known in the current shell execution environment”; see Section 2.9.3.1 (on page 2259).16917 **OPTIONS**

16918 None.

16919 **OPERANDS**

16920 The following operand shall be supported:

16921 *job_id* Specify the job to be run as a foreground job. If no *job_id* operand is given, the
16922 *job_id* for the job that was most recently suspended, placed in the background or
16923 run as a background job, shall be used. The format of *job_id* is described in the Base
16924 Definitions volume of IEEE Std. 1003.1-200x, Section 3.205, Job Control Job ID.16925 **STDIN**

16926 Not used.

16927 **INPUT FILES**

16928 None.

16929 **ENVIRONMENT VARIABLES**16930 The following environment variables shall affect the execution of *fg*:16931 *LANG* Provide a default value for the internationalization variables that are unset or null.
16932 If *LANG* is unset or null, the corresponding value from the implementation-
16933 defined default locale shall be used. If any of the internationalization variables
16934 contains an invalid setting, the utility shall behave as if none of the variables had
16935 been defined.16936 *LC_ALL* If set to a non-empty string value, override the values of all the other
16937 internationalization variables.16938 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
16939 characters (for example, single-byte as opposed to multi-byte characters in
16940 arguments).16941 *LC_MESSAGES*16942 Determine the locale that should be used to affect the format and contents of
16943 diagnostic messages written to standard error.16944 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.16945 **ASYNCHRONOUS EVENTS**

16946 Default.

16947 **STDOUT**16948 The *fg* utility shall write the command line of the job to standard output in the following format:16949 "%s\n", <*command*>

16950 **STDERR**

16951 Used only for diagnostic messages.

16952 **OUTPUT FILES**

16953 None.

16954 **EXTENDED DESCRIPTION**

16955 None.

16956 **EXIT STATUS**

16957 The following exit values shall be returned:

16958 0 Successful completion.

16959 >0 An error occurred.

16960 **CONSEQUENCES OF ERRORS**

16961 If job control is disabled, the *fg* utility shall exit with an error and no job shall be placed in the foreground.

16963 **APPLICATION USAGE**

16964 The *fg* utility does not work as expected when it is operating in its own utility execution environment because that environment has no applicable jobs to manipulate. See the APPLICATION USAGE section for *bg* (on page 2422). For this reason, *fg* is generally implemented as a shell regular built-in.

16968 **EXAMPLES**

16969 None.

16970 **RATIONALE**

16971 The extensions to the shell specified in this volume of IEEE Std. 1003.1-200x have mostly been based on features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also based on the KornShell. The standard developers examined the characteristics of the C shell versions of these utilities and found that differences exist. Despite widespread use of the C shell, the KornShell versions were selected for this volume of IEEE Std. 1003.1-200x to maintain a degree of uniformity with the rest of the KornShell features selected (such as the very popular command line editing features).

16978 **FUTURE DIRECTIONS**

16979 None.

16980 **SEE ALSO**

16981 *bg*, *kill*, *jobs*, *wait*

16982 **CHANGE HISTORY**

16983 First released in Issue 4.

16984 **Issue 6**

16985 This utility is now marked as part of the User Portability Utilities option.

16986 The APPLICATION USAGE section is added.

16987 The JC marking is removed from the SYNOPSIS since job control is mandatory in this issue.

16988 NAME

16989 file — determine file type

16990 SYNOPSIS

16991 UP file [-dhi][-M file][-m file] file ...

16992

16993 DESCRIPTION

16994 The *file* utility shall perform a series of tests on each specified *file* in an attempt to classify it:16995 1. If the file is not a regular file, its file type shall be identified. The file types directory, FIFO,
16996 block special, and character special shall be identified as such. Other implementation-
16997 defined file types may also be identified.

16998 2. If the file is a regular file, and:

16999 a. The file is zero-length, it shall be identified as an empty file.

17000 b. The file is not zero-length, *file* shall examine an initial segment of the file and shall
17001 make a guess at identifying its contents or whether it is an executable binary file.
17002 (The answer is not guaranteed to be correct.)17003 If *file* does not exist, cannot be read, or its file status could not be determined, the output shall
17004 indicate that the file was processed, but that its type could not be determined.17005 If *file* is a symbolic link, by default the link shall be resolved and *file* shall test the type of file
17006 referenced by the symbolic link.

17007 OPTIONS

17008 The *file* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
17009 12.2, Utility Syntax Guidelines.

17010 The following options shall be supported by the implementation:

17011 **-d** Apply any default system tests to the file.17012 **-h** When a symbolic link is encountered, identify the file as a symbolic link. If **-h** is
17013 not specified and *file* is a symbolic link that refers to a nonexistent file, *file* shall
17014 identify the file as a symbolic link, as if **-h** had been specified.17015 **-i** If a file is a regular file, do not attempt to classify the type of the file further, but
17016 identify the file as specified in the STDOUT section, using a *<type>* string that
17017 contains the string "regular file".17018 **-M file** Specify the name of a file containing tests that shall be applied to a file in order to
17019 classify it (see the EXTENDED DESCRIPTION). No default system tests shall be
17020 applied.17021 **-m file** Specify the name of a file containing tests that shall be applied to a file in order to
17022 classify it (see the EXTENDED DESCRIPTION).17023 If multiple instances of the **-m**, **-d**, or **-M** options are specified, the concatenation of the tests
17024 specified, in the order specified, shall be the set of tests that are applied. If a **-M** option is
17025 specified, no tests other than those specified using the **-d**, **-M**, and **-m** options shall be applied
17026 to the file. If neither the **-d** nor **-M** options are specified, any default system tests shall be
17027 applied after any tests specified using the **-m** option.

17028 **OPERANDS**

17029 The following operand shall be supported:

17030 *file* A path name of a file to be tested.

17031 **STDIN**

17032 Not used.

17033 **INPUT FILES**

17034 The *file* can be any file type.

17035 **ENVIRONMENT VARIABLES**

17036 The following environment variables shall affect the execution of *file*:

17037 *LANG* Provide a default value for the internationalization variables that are unset or null.
 17038 If *LANG* is unset or null, the corresponding value from the implementation-
 17039 defined default locale shall be used. If any of the internationalization variables
 17040 contains an invalid setting, the utility shall behave as if none of the variables had
 17041 been defined.

17042 *LC_ALL* If set to a non-empty string value, override the values of all the other
 17043 internationalization variables.

17044 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 17045 characters (for example, single-byte as opposed to multi-byte characters in
 17046 arguments and input files).

17047 *LC_MESSAGES*

17048 Determine the locale that should be used to affect the format and contents of
 17049 diagnostic messages written to standard error and informative messages written to
 17050 standard output.

17051 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

17052 **ASYNCHRONOUS EVENTS**

17053 Default.

17054 **STDOUT**

17055 In the POSIX locale, the following format shall be used to identify each operand, *file* specified:

17056 "%s: %s\n", <*file*>, <*type*>

17057 The values for <*type*> are unspecified, except that in the POSIX locale, if *file* is identified as one
 17058 of the types listed in the following table, <*type*> shall contain (but is not limited to) the
 17059 corresponding string. Each space shown in the strings shall be exactly one <space> character.

17060

Table 4-8 File Utility Output Strings

17061

If file is a:	<type> shall contain the string:
Directory	directory
FIFO	fifo
Block special	block special
Character special	character special
Executable binary	executable
Empty regular file	empty
Symbolic link	symbolic link to
<i>ar</i> archive library (see <i>ar</i>)	archive
Extended <i>cpio</i> format (see <i>pax</i>)	cpio archive
Extended <i>tar</i> format (see ustar in <i>pax</i>)	tar archive
Shell script	commands text
C-language source	c program text
FORTRAN source	fortran program text

17062

17063

17064

17065

17066

17067

17068

17069

17070

17071

17072

17073

17074

17075

If *file* is identified as a symbolic link (see **-h**), the following alternative output format shall be used:

17076

17077

"%s: %s %s\n", <file>, <type>, <contents of link>"

17078

If the file named by the *file* operand does not exist or cannot be read, the string "cannot open" shall be included as part of the <type> field, but this shall not be considered an error that affects the exit status. If the type of the file named by the *file* operand cannot be determined, the string "data" shall be included as part of the <type> field, but this shall not be considered an error that affects the exit status.

17079

17080

17081

17082

17083 **STDERR**

17084

Used only for diagnostic messages.

17085 **OUTPUT FILES**

17086

None.

17087 **EXTENDED DESCRIPTION**

17088

A file specified as an option-argument to the **-m** or **-M** options shall contain one test per line, which shall be applied to the file. If the test succeeds, the message field of the line shall be printed and no further tests shall be applied, with the exception that tests on immediately following lines beginning with a single '**>**' character shall be applied.

17089

17090

17091

17092

Each line shall be composed of the following four <blank>-separated fields:

17093

offset

An unsigned number (optionally preceded by a single '**>**' character) specifying the *offset*, in bytes, of the value in the file that is to be compared against the *value* field of the line. If the file is shorter than the specified offset, the test shall fail.

17094

17095

17096

17097

17098

17099

17100

17101

If the *offset* begins with the character '**>**', the test contained in the line shall not be applied to the file unless the test on the last line for which the *offset* did not begin with a '**>**' was successful. By default, the *offset* shall be interpreted as an unsigned decimal number. With a leading 0x or 0X, the *offset* shall be interpreted as a hexadecimal number; otherwise, with a leading 0, the *offset* shall be interpreted as an octal number.

17102

type

The type of the value in the file to be tested. The type shall consist of the type specification characters **c**, **d**, **f**, **s**, and **u**, specifying character, signed decimal, floating point, string, and unsigned decimal, respectively.

17103

17104

17105 The *type* string shall be interpreted as the bytes from the file starting at the
 17106 specified *offset* and including the same number of bytes specified by the *value* field.
 17107 If insufficient bytes remain in the file past the *offset* to match the *value* field, the test
 17108 shall fail.

17109 The type specification characters **d**, **f**, and **u** can be followed by an optional
 17110 unsigned decimal integer that specifies the number of bytes represented by the
 17111 type. The type specification character **f** can be followed by an optional **F**, **D**, or **L**,
 17112 indicating that the value is of type **float**, **double**, or **long double**, respectively. The
 17113 type specification characters **d** and **u** can be followed by an optional **C**, **S**, **I**, or **L**,
 17114 indicating that the value is of type **char**, **short**, **int**, or **long**, respectively.

17115 The default number of bytes represented by the type specifiers **d**, **f**, and **u** shall
 17116 correspond to their respective C-language types as follows. If the system claims
 17117 conformance to the C-Language Development Utilities option, those specifiers
 17118 shall correspond to the default sizes used in the *c99* utility. Otherwise, the default
 17119 sizes shall be implementation-defined.

17120 For the type specifier characters **d** and **u**, the default number of bytes shall
 17121 correspond to the size of a basic integer type of the implementation. For these
 17122 specifier characters, the implementation shall support values of the optional
 17123 number of bytes to be converted corresponding to the number of bytes in the C-
 17124 language types **char**, **short**, **int**, or **long**. These numbers can also be specified by an
 17125 application as the characters **C**, **S**, **I**, and **L**, respectively. The byte order used when
 17126 interpreting numeric values is implementation-defined, but shall correspond to the
 17127 order in which a constant of the corresponding type is stored in memory on the
 17128 system.

17129 For the type specifier **f**, the default number of bytes shall correspond to the number
 17130 of bytes in the basic double precision floating-point data type of the underlying
 17131 implementation. The implementation shall support values of the optional number
 17132 of bytes to be converted corresponding to the number of bytes in the C-language
 17133 types **float**, **double**, and **long double**. These numbers can also be specified by an
 17134 application as the characters **F**, **D**, and **L**, respectively.

17135 All type specifiers, except for **s**, can be followed by a mask specifier of the form
 17136 *&number*. The mask value shall be AND'ed with the value before the comparison
 17137 with the value from the file is made. By default, the mask shall be interpreted as an
 17138 unsigned decimal number. With a leading 0x or 0X, the mask shall be interpreted
 17139 as an unsigned hexadecimal number; otherwise, with a leading 0, the mask shall be
 17140 interpreted as an unsigned octal number.

17141 The strings **byte**, **short**, **long**, and **string** shall also be supported as type fields,
 17142 being interpreted as **dC**, **dS**, **dL**, and **s**, respectively.

17143 *value* The *value* to be compared with the value from the file.

17144 Any value that contains a character that is not a digit, other than a leading sign
 17145 ('+' or '-') or a leading 0x or 0X, shall be interpreted as a string. The test shall
 17146 succeed only when a string value exactly matches the bytes from the file.

17147 If the *value* is a string, it can contain the following sequences:

17148 *character* The backslash-escape sequences as specified in the Base
 17149 Definitions volume of IEEE Std. 1003.1-200x, Table 5-1, Escape
 17150 Sequences and Associated Actions ('\\', '\a', '\b', '\f',
 17151 '\n', '\r', '\t', '\v'). The results of using any other

17152 character, other than an octal digit, following the backslash are
 17153 unspecified.

17154 `\octal` Octal sequences that can be used to represent characters with
 17155 specific coded values. An octal sequence shall consist of a
 17156 backslash followed by the longest sequence of one, two, or three
 17157 octal-digit characters (01234567). If the size of a byte on the
 17158 system is greater than 9 bits, the valid escape sequence used to
 17159 represent a byte is implementation-defined.

17160 By default, any value that is not a string shall be interpreted as a signed decimal
 17161 number. Any such value, with a leading 0x or 0X, shall be interpreted as an
 17162 unsigned hexadecimal number; otherwise, with a leading zero, the value shall be
 17163 interpreted as an unsigned octal number.

17164 If the value is not a string, it can be preceded by a character indicating the
 17165 comparison to be performed. Permissible characters and the comparisons they
 17166 specify are as follows:

17167 = The test shall succeed if the value from the file equals the *value* field.
 17168 < The test shall succeed if the value from the file is less than the *value* field.
 17169 > The test shall succeed if the value from the file is greater than the *value* field.
 17170 & The test shall succeed if all of the bits in the *value* field are set in the value
 17171 from the file.

17172 ^ The test shall succeed if at least one of the bits in the *value* field is not set in the
 17173 value from the file.

17174 x The test shall succeed if there is any value in the file.

17175 *message* The *message* to be printed if the test succeeds. The *message* shall be interpreted
 17176 using the notation for the *printf* formatting specification; see *printf*. If the *value*
 17177 field was a string, then the value from the file shall be the argument for the *printf*
 17178 formatting specification; otherwise, the value from the file shall be the argument.

17179 EXIT STATUS

17180 The following exit values shall be returned:

17181 0 Successful completion.

17182 >0 An error occurred.

17183 CONSEQUENCES OF ERRORS

17184 Default.

17185 APPLICATION USAGE

17186 The *file* utility can only be required to guess at many of the file types because only exhaustive
 17187 testing can determine some types with certainty. For example, binary data on some systems
 17188 might match the initial segment of an executable or a *tar* archive.

17189 Note that the table indicates that the output contains the stated string. Systems may add text
 17190 before or after the string. For executables, as an example, the machine architecture and various
 17191 facts about how the file was link-edited may be included.

17192 **EXAMPLES**

17193 Determine whether an argument is a binary executable file:

```
17194 file "$1" | grep -Fq executable &&
17195 printf "%s is executable.\n" "$1"
```

17196 **RATIONALE**

17197 The `-f` option was omitted because the same effect can (and should) be obtained using the *xargs*
17198 utility.

17199 Historical versions of the *file* utility attempt to identify the following types of files: symbolic link,
17200 directory, character special, block special, socket, *tar* archive, *cpio* archive, *SCCS* archive, archive
17201 library, empty, *compress* output, *pack* output, binary data, C source, FORTRAN source, assembler
17202 source, *nroff*/*troff*/*eqn*/*tbl* source *troff* output, shell script, C shell script, English text, ASCII text,
17203 various executables, APL workspace, compiled terminfo entries, and *CURSES* screen images.
17204 Only those types that are reasonably well specified in POSIX or are directly related to POSIX
17205 utilities are listed in the table.

17206 Implementations that support symbolic links are encouraged to use the string "symbolic
17207 link" to identify them.

17208 Historical systems have used a "magic file" named `/etc/magic` to help identify file types. Because
17209 it is generally useful for users and scripts to be able to identify special file types, the `-m` flag and
17210 a portable format for user-created magic files has been specified. No requirement is made that an
17211 implementation of *file* use this method of identifying files, only that users be permitted to add
17212 their own classifying tests.

17213 In addition, three options have been added to historical practice. The `-d` flag has been added to
17214 permit users to cause their tests to follow any default system tests. The `-i` flag has been added to
17215 permit users to test portably for regular files in shell scripts. The `-M` flag has been added to
17216 permit users to ignore any default system tests.

17217 The historical `-c` option was omitted as not particularly useful to users or portable shell scripts.
17218 In addition, a reasonable implementation of the *file* utility would report any errors found each
17219 time the magic file is read.

17220 The historical format of the magic file was the same as that specified by the Rationale in the
17221 previous version of IEEE Std. 1003.1-200x for the *offset*, *value*, and *message* fields; however, it
17222 used less precise type fields than the format specified by the current normative text. The new
17223 type field values are a superset of the historical ones.

17224 The following is an example magic file:

```
17225 0 short      070707          cpio archive
17226 0 short      0143561        Byte-swapped cpio archive
17227 0 string     070707          ASCII cpio archive
17228 0 long       0177555        Very old archive
17229 0 short      0177545        Old archive
17230 0 short      017437         Old packed data
17231 0 string     \037\036       Packed data
17232 0 string     \377\037       Compacted data
17233 0 string     \037\235       Compressed data
17234 >2 byte&0x80 >0          Block compressed
17235 >2 byte&0x1f x            %d bits
17236 0 string     \032\001       Compiled Terminfo Entry
17237 0 short      0433           Curses screen image
17238 0 short      0434           Curses screen image
```

17239	0	string	<ar>	System V Release 1 archive
17240	0	string	!<arch>\n__.SYMDEF	Archive random library
17241	0	string	!<arch>	Archive
17242	0	string	ARF_BEGARF	PHIGS clear text archive
17243	0	long	0x137A2950	Scalable OpenFont binary
17244	0	long	0x137A2951	Encrypted scalable OpenFont binary
17245				The use of a basic integer data type is intended to allow the implementation to choose a word
17246				size commonly used by applications on that architecture.
17247				FUTURE DIRECTIONS
17248				None.
17249				SEE ALSO
17250				<i>ls</i>
17251				CHANGE HISTORY
17252				First released in Issue 4.
17253				Issue 6
17254				This utility is now marked as part of the User Portability Utilities option.
17255				Options and an EXTENDED DESCRIPTION are added as specified in the IEEE P1003.2b draft
17256				standard.

17257 **NAME**

17258 find — find files

17259 **SYNOPSIS**17260 find [-H | -L] *path* ... [*operand_expression* ...]17261 **DESCRIPTION**

17262 The *find* utility shall recursively descend the directory hierarchy from each file specified by *path*,
 17263 evaluating a Boolean expression composed of the primaries described in the OPERANDS section
 17264 for each file encountered.

17265 The *find* utility shall be able to descend to arbitrary depths in a file hierarchy and shall not fail
 17266 due to path length limitations (unless a *path* operand specified by the application exceeds
 17267 {PATH_MAX} requirements).

17268 The *find* utility shall detect infinite loops; that is, entering a previously visited directory that is an
 17269 ancestor of the last file encountered. When it detects an infinite loop, *find* shall write a
 17270 diagnostic message to standard error and shall either recover its position in the hierarchy or
 17271 terminate.

17272 **OPTIONS**

17273 The *find* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 17274 12.2, Utility Syntax Guidelines.

17275 The following options shall be supported by the implementation:

17276 **-H** Cause the file information and file type evaluated for each symbolic link
 17277 encountered on the command line to be those of the file referenced by the link, and
 17278 not the link itself. If the referenced file does not exist, the file information and type
 17279 shall be for the link itself. File information for all symbolic links not on the
 17280 command line shall be that of the link itself.

17281 **-L** Cause the file information and file type evaluated for each symbolic link to be
 17282 those of the file referenced by the link, and not the link itself.

17283 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered
 17284 an error. The last option specified shall determine the behavior of the utility.

17285 **OPERANDS**

17286 The following operands shall be supported:

17287 The *path* operand is a path name of a starting point in the directory hierarchy.

17288 The first argument that starts with a '-', or is a '!' or a '(', and all subsequent arguments
 17289 shall be interpreted as an *expression* made up of the following primaries and operators. In the
 17290 descriptions, wherever *n* is used as a primary argument, it shall be interpreted as a decimal
 17291 integer optionally preceded by a plus ('+') or minus ('-') sign, as follows:

17292 **+n** More than *n*.

17293 **n** Exactly *n*.

17294 **-n** Less than *n*.

17295 The following primaries shall be supported:

17296 **-name** *pattern*

17297 The primary shall evaluate as true if the basename of the file name being examined
 17298 matches *pattern* using the pattern matching notation described in Section 2.14 (on
 17299 page 2274).

- 17300 **-nouser** The primary shall evaluate as true if the file belongs to a user ID for which the
17301 *getpwuid()* function defined in the System Interfaces volume of
17302 IEEE Std. 1003.1-200x (or equivalent) returns NULL.
- 17303 **-nogroup** The primary shall evaluate as true if the file belongs to a group ID for which the
17304 *getgrgid()* function defined in the System Interfaces volume of
17305 IEEE Std. 1003.1-200x (or equivalent) returns NULL.
- 17306 **-xdev** The primary always shall evaluate as true; it shall cause *find* not to continue
17307 descending past directories that have a different device ID (*st_dev*, see the *stat()*
17308 function defined in the System Interfaces volume of IEEE Std. 1003.1-200x). If any
17309 **-xdev** primary is specified, it shall apply to the entire expression even if the **-xdev**
17310 primary would not normally be evaluated.
- 17311 **-prune** The primary always shall evaluate as true; it shall cause *find* not to descend the
17312 current path name if it is a directory. If the **-depth** primary is specified, the **-prune**
17313 primary shall have no effect.
- 17314 **-perm [-]mode**
17315 The *mode* argument is used to represent file mode bits. It shall be identical in
17316 format to the *symbolic_mode* operand described in *chmod* (on page 2450), and shall
17317 be interpreted as follows. To start, a template shall be assumed with all file mode
17318 bits cleared. An *op* symbol of '+' shall set the appropriate mode bits in the
17319 template; '-' shall clear the appropriate bits; '=' shall set the appropriate mode
17320 bits, without regard to the contents of process' file mode creation mask. The *op*
17321 symbol of '-' cannot be the first character of *mode*; this avoids ambiguity with the
17322 optional leading hyphen. Since the initial mode is all bits off, there are not any
17323 symbolic modes that need to use '-' as the first character.
- 17324 If the hyphen is omitted, the primary shall evaluate as true when the file
17325 permission bits exactly match the value of the resulting template.
- 17326 Otherwise, if *mode* is prefixed by a hyphen, the primary shall evaluate as true if at
17327 least all the bits in the resulting template are set in the file permission bits.
- 17328 **-perm [-]onum**
17329 If the hyphen is omitted, the primary shall evaluate as true when the file
17330 permission bits exactly match the value of the octal number *onum* and only the bits
17331 corresponding to the octal mask 07777 shall be compared. (See the description of
17332 the octal *mode* in *chmod* (on page 2450).) Otherwise, if *onum* is prefixed by a
17333 hyphen, the primary shall evaluate as true if at least all of the bits specified in *onum*
17334 that are also set in the octal mask 07777 are set.
- 17335 **-type c** The primary shall evaluate as true if the type of the file is *c*, where *c* is 'b', 'c',
17336 'd', 'l', 'p', 'f', or 's' for block special file, character special file, directory,
17337 symbolic link, FIFO, regular file, or socket, respectively.
- 17338 **-links n** The primary shall evaluate as true if the file has *n* links.
- 17339 **-user uname** The primary shall evaluate as true if the file belongs to the user *uname*. If *uname* is
17340 a decimal integer and the *getpwnam()* (or equivalent) function does not return a
17341 valid user name, *uname* shall be interpreted as a user ID.
- 17342 **-group gname**
17343 The primary shall evaluate as true if the file belongs to the group *gname*. If *gname*
17344 is a decimal integer and the *getgrnam()* (or equivalent) function does not return a
17345 valid group name, *gname* shall be interpreted as a group ID.

- 17346 **-size** *n*[*c*] The primary shall evaluate as true if the file size in bytes, divided by 512 and
17347 rounded up to the next integer, is *n*. If *n* is followed by the character '*c*', the size
17348 shall be in bytes.
- 17349 **-atime** *n* The primary shall evaluate as true if the file access time subtracted from the
17350 initialization time, divided by 86 400 (with any remainder discarded), is *n*.
- 17351 **-ctime** *n* The primary shall evaluate as true if the time of last change of file status
17352 information subtracted from the initialization time, divided by 86 400 (with any
17353 remainder discarded), is *n*.
- 17354 **-mtime** *n* The primary shall evaluate as true if the file modification time subtracted from the
17355 initialization time, divided by 86 400 (with any remainder discarded), is *n*.
- 17356 **-exec** *utility_name* [*argument* . . .];
17357 The primary shall evaluate as true if the invoked utility *utility_name* returns a zero
17358 value as exit status. The end of the primary expression shall be punctuated by a
17359 semicolon. A *utility_name* or *argument* containing only the two characters "{}"
17360 shall be replaced by the current path name. If a *utility_name* or argument string
17361 contains the two characters "{}", but not just the two characters "{}", it is
17362 implementation-defined whether *find* replaces those two characters with the
17363 current path name or uses the string without change. The current directory for the
17364 invocation of *utility_name* shall be the same as the current directory when the *find*
17365 utility was started. If the *utility_name* names any of the special built-in utilities in
17366 Section 2.15 (on page 2276), the results are undefined.
- 17367 **-ok** *utility_name* [*argument* . . .];
17368 The **-ok** primary shall be equivalent to **-exec**, except that *find* shall request
17369 affirmation of the invocation of *utility_name* using the current file as an argument
17370 by writing to standard error as described in the STDERR section. If the response on
17371 standard input is affirmative, the utility shall be invoked. Otherwise, the command
17372 shall not be invoked and the value of the **-ok** operand shall be false.
- 17373 **-print** The primary always shall evaluate as true; it shall cause the current path name to
17374 be written to standard output.
- 17375 **-newer file** The primary shall evaluate as true if the modification time of the current file is
17376 more recent than the modification time of the file named by the path name *file*.
- 17377 **-depth** The primary shall always evaluate as true; it shall cause descent of the directory
17378 hierarchy to be done so that all entries in a directory are acted on before the
17379 directory itself. If a **-depth** primary is not specified, all entries in a directory shall
17380 be acted on after the directory itself. If any **-depth** primary is specified, it shall
17381 apply to the entire expression even if the **-depth** primary would not normally be
17382 evaluated.
- 17383 The primaries can be combined using the following operators (in order of decreasing
17384 precedence):
- 17385 (*expression*) True if *expression* is true.
- 17386 !*expression* Negation of a primary; the unary NOT operator.
- 17387 *expression* [**-a**] *expression*
17388 Conjunction of primaries; the AND operator is implied by the juxtaposition of two
17389 primaries or made explicit by the optional **-a** operator. The second expression
17390 shall not be evaluated if the first expression is false.

- 17391 *expression -o expression*
 17392 Alternation of primaries; the OR operator. The second expression shall not be
 17393 evaluated if the first expression is true.
- 17394 If no *expression* is present, **-print** shall be used as the expression. Otherwise, if the given
 17395 expression does not contain any of the primaries **-exec**, **-ok**, or **-print**, the given expression shall
 17396 be effectively replaced by:
- 17397 (*given_expression*) -print
- 17398 The **-user**, **-group**, and **-newer** primaries each shall evaluate their respective arguments only
 17399 once.
- 17400 **STDIN**
- 17401 If the **-ok** primary is used, the response shall be read from the standard input. An entire line
 17402 shall be read as the response. Otherwise, the standard input shall not be used.
- 17403 **INPUT FILES**
- 17404 None.
- 17405 **ENVIRONMENT VARIABLES**
- 17406 The following environment variables shall affect the execution of *find*:
- 17407 **LANG** Provide a default value for the internationalization variables that are unset or null.
 17408 If *LANG* is unset or null, the corresponding value from the implementation-
 17409 defined default locale shall be used. If any of the internationalization variables
 17410 contains an invalid setting, the utility shall behave as if none of the variables had
 17411 been defined.
- 17412 **LC_ALL** If set to a non-empty string value, override the values of all the other
 17413 internationalization variables.
- 17414 **LC_COLLATE**
- 17415 Determine the locale for the behavior of ranges, equivalence classes and multi-
 17416 character collating elements used in the pattern matching notation for the **-n**
 17417 option and in the extended regular expression defined for the **yesexpr** locale
 17418 keyword in the *LC_MESSAGES* category.
- 17419 **LC_CTYPE** This variable determines the locale for the interpretation of sequences of bytes of
 17420 text data as characters (for example, single-byte as opposed to multi-byte
 17421 characters in arguments), the behavior of character classes within the pattern
 17422 matching notation used for the **-n** option, and the behavior of character classes
 17423 within regular expressions used in the extended regular expression defined for the
 17424 **yesexpr** locale keyword in the *LC_MESSAGES* category.
- 17425 **LC_MESSAGES**
- 17426 Determine the locale for the processing of affirmative responses that should be
 17427 used to affect the format and contents of diagnostic messages written to standard
 17428 error.
- 17429 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 17430 **PATH** Determine the location of the *utility_name* for the **-exec** and **-ok** primaries, as
 17431 described in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 8,
 17432 Environment Variables.

17433 **ASYNCHRONOUS EVENTS**

17434 Default.

17435 **STDOUT**

17436 The **-print** primary shall cause the current path names to be written to standard output. The
 17437 format shall be:

17438 "%s\n", <path>

17439 **STDERR**

17440 The **-ok** primary shall write a prompt to standard error containing at least the *utility_name* to be
 17441 invoked and the current path name. In the POSIX locale, the last non-<blank> character in the
 17442 prompt shall be '?'. The exact format used is unspecified.

17443 Otherwise, the standard error shall be used only for diagnostic messages.

17444 **OUTPUT FILES**

17445 None.

17446 **EXTENDED DESCRIPTION**

17447 None.

17448 **EXIT STATUS**

17449 The following exit values shall be returned:

17450 0 All *path* operands were traversed successfully.

17451 >0 An error occurred.

17452 **CONSEQUENCES OF ERRORS**

17453 Default.

17454 **APPLICATION USAGE**

17455 When used in operands, pattern matching notation, semicolons, opening parentheses, and
 17456 closing parentheses are special to the shell and must be quoted (see Section 2.2 (on page 2236)).

17457 The bit that is traditionally used for sticky (historically 01000) is specified in the **-perm** primary
 17458 using the octal number argument form. Since this bit is not defined by this volume of
 17459 IEEE Std. 1003.1-200x, applications must not assume that it actually refers to the traditional
 17460 sticky bit.

17461 **EXAMPLES**

17462 1. The following commands are equivalent:

17463 `find .`17464 `find . -print`

17465 They both write out the entire directory hierarchy from the current directory.

17466 2. The following command:

17467 `find / \(-name tmp -o -name '*.xx' \) -atime +7 -exec rm {} \;`

17468 removes all files named **tmp** or ending in **.xx** that have not been accessed for seven or more
 17469 24-hour periods.

17470 3. The following command:

17471 `find . -perm -o+w,+s`

17472 prints (**-print** is assumed) the names of all files in or below the current directory, with all
 17473 of the file permission bits **S_ISUID**, **S_ISGID**, and **S_IWOTH** set.

- 17474 4. The following command:
- ```
17475 find . -name SCCS -prune -o -print
```
- 17476 recursively prints path names of all files in the current directory and below, but skips  
17477 directories named SCCS and files in them.
- 17478 5. The following command:
- ```
17479 find . -print -name SCCS -prune
```
- 17480 behaves as in the previous example, but prints the names of the SCCS directories.
- 17481 6. The following command is roughly equivalent to the **-nt** extension to *test*:
- ```
17482 if [-n "$(find file1 -prune -newer file2)"]; then
17483 printf %s\n "file1 is newer than file2"
17484 fi
```
- 17485 7. The descriptions of **-atime**, **-ctime**, and **-mtime** use the terminology *n* “86 400 second  
17486 periods (days)”. For example, a file accessed at 23:59 is selected by:
- ```
17487 find . -atime -1 -print
```
- 17488 at 00:01 the next day (less than 24 hours later, not more than one day ago); the midnight
17489 boundary between days has no effect on the 24-hour calculation.

17490 RATIONALE

17491 The **-a** operator was retained as an optional operator for compatibility with historical shell
17492 scripts, even though it is redundant with expression concatenation.

17493 The descriptions of the ‘-’ modifier on the *mode* and *onum* arguments to the **-perm** primary
17494 agree with historical practice on BSD and System V implementations. System V and BSD
17495 documentation both describe it in terms of checking additional bits; in fact, it uses the same bits,
17496 but checks for having at least all of the matching bits set instead of having exactly the matching
17497 bits set.

17498 The exact format of the interactive prompts is unspecified. Only the general nature of the
17499 contents of prompts are specified because:

- 17500 • Implementations may desire more descriptive prompts than those used on historical
17501 implementations.
- 17502 • Since the historical prompt strings do not terminate with <newline>s, there is no portable
17503 way for another program to interact with the prompts of this utility via pipes.

17504 Therefore, an application using this prompting option relies on the system to provide the most
17505 suitable dialog directly with the user, based on the general guidelines specified.

17506 The **-name file** operand was changed to use the shell pattern matching notation so that *find* is
17507 consistent with other utilities using pattern matching.

17508 The **-size** operand refers to the size of a file, rather than the number of blocks it may occupy in
17509 the file system. The intent is that the *st_size* field defined in the System Interfaces volume of
17510 IEEE Std. 1003.1-200x should be used, not the *st_blocks* found in historical implementations.
17511 There are at least two reasons for this:

- 17512 1. In both System V and BSD, *find* only uses *st_size* in size calculations for the operands
17513 specified by this volume of IEEE Std. 1003.1-200x. (BSD uses *st_blocks* only when
17514 processing the **-ls** primary.)

17515 2. Users usually think of file size in terms of bytes, which is also the unit used by the *ls* utility
 17516 for the output from the `-l` option. (In both System V and BSD, *ls* uses *st_size* for the `-l`
 17517 option size field and uses *st_blocks* for the *ls -s* calculations. This volume of
 17518 IEEE Std. 1003.1-200x does not specify *ls -s*.)

17519 The descriptions of `-atime`, `-ctime`, and `-mtime` were changed from the SVID description of *n*
 17520 “days” to “24-hour periods”. The description is also different in terms of the exact timeframe for
 17521 the *n* case (*versus* the *+n* or *-n*), but it matches all known historical implementations. It refers to
 17522 one 86 400 second period in the past, not any time from the beginning of that period to the
 17523 current time. For example, `-atime 3` is true if the file was accessed any time in the period from 72
 17524 hours to 48 hours ago.

17525 Historical implementations do not modify “{ }” when it appears as a substring of an `-exec` or
 17526 `-ok utility_name` or argument string. There have been numerous user requests for this extension,
 17527 so this volume of IEEE Std. 1003.1-200x allows the desired behavior. At least one recent
 17528 implementation does support this feature, but encountered several problems in managing
 17529 memory allocation and dealing with multiple occurrences of “{ }” in a string while it was being
 17530 developed, so it is not yet required behavior.

17531 Assuming the presence of `-print` was added to correct a historical pitfall that plagues novice
 17532 users, it is entirely upward-compatible from the historical System V *find* utility. In its simplest
 17533 form (*find directory*), it could be confused with the historical BSD fast *find*. The BSD developers
 17534 agreed that adding `-print` as a default expression was the correct decision and have added the
 17535 fast *find* functionality within a new utility called *locate*.

17536 Historically, the `-L` option was implemented using the primary `-follow`. The `-H` and `-L` options
 17537 were added for two reasons. First, they offer a finer granularity of control and consistency with
 17538 other programs that walk file hierarchies. Second, the `-follow` primary always evaluated to true.
 17539 As they were historically really global variables that took effect before the traversal began, some
 17540 valid expressions had unexpected results. An example is the expression `-print -o -follow`.
 17541 Because `-print` always evaluates to true, the standard order of evaluation implies that `-follow`
 17542 would never be evaluated. This was never the case. Historical practice for the `-follow` primary,
 17543 however, is not consistent. Some implementations always follow symbolic links on the
 17544 command line whether `-follow` is specified or not. Others follow symbolic links on the
 17545 command line only if `-follow` is specified. Both behaviors are provided by the `-H` and `-L`
 17546 options, but scripts using the current `-follow` primary would be broken if the `-follow` option is
 17547 specified to work either way.

17548 Since the `-L` option resolves all symbolic links and the `-type l` primary is true for symbolic links
 17549 that still exist after symbolic links have been resolved, the command:

```
17550 find -L . -type l
```

17551 prints a list of symbolic links reachable from the current directory that do not resolve to
 17552 accessible files.

17553 FUTURE DIRECTIONS

17554 None.

17555 SEE ALSO

17556 *chmod*, *pax*, *sh*, *test*, the System Interfaces volume of IEEE Std. 1003.1-200x, *stat()*

17557 CHANGE HISTORY

17558 First released in Issue 2.

17559 **Issue 4**

17560 Aligned with the ISO/IEC 9945-2: 1993 standard.

17561 **Issue 5**

17562 FUTURE DIRECTIONS section added.

17563 **Issue 6**

17564 The following new requirements on POSIX implementations derive from alignment with the
17565 Single UNIX Specification:

17566 • The `-perm [-]onum` primary is supported.

17567 The `find` utility is aligned with the IEEE P1003.2b draft standard, to include processing of
17568 symbolic links and changes to the description of the `atime`, `ctime`, and `mtime` operands.

17569 **NAME**

17570 fold — filter for folding lines

17571 **SYNOPSIS**17572 fold [-bs][-w *width*][*file...*]17573 **DESCRIPTION**

17574 The *fold* utility is a filter that shall fold lines from its input files, breaking the lines to have a
 17575 maximum of *width* column positions (or bytes, if the **-b** option is specified). Lines shall be
 17576 broken by the insertion of a <newline> character such that each output line (referred to later in
 17577 this section as a *segment*) is the maximum width possible that does not exceed the specified
 17578 number of column positions (or bytes). A line shall not be broken in the middle of a character.
 17579 The behavior is undefined if *width* is less than the number of columns any single character in the
 17580 input would occupy.

17581 If the <carriage-return>, <backspace>, or <tab> characters are encountered in the input, and the
 17582 **-b** option is not specified, they shall be treated specially:

17583 <backspace> The current count of line width shall be decremented by one, although the count
 17584 never shall become negative. The *fold* utility shall not insert a <newline> character
 17585 immediately before or after any <backspace> character.

17586 <carriage-return>
 17587 The current count of line width shall be set to zero. The *fold* utility shall not insert a
 17588 <newline> character immediately before or after any <carriage-return> character.

17589 <tab> Each <tab> character encountered shall advance the column position pointer to the
 17590 next tab stop. Tab stops shall be at each column position *n* such that *n* modulo 8
 17591 equals 1.

17592 **OPTIONS**

17593 The *fold* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 17594 12.2, Utility Syntax Guidelines.

17595 The following options shall be supported:

17596 **-b** Count *width* in bytes rather than column positions.

17597 **-s** If a segment of a line contains a <blank> character within the first *width* column
 17598 positions (or bytes), break the line after the last such <blank> character meeting the
 17599 width constraints. If there is no <blank> character meeting the requirements, the **-s**
 17600 option shall have no effect for that output segment of the input line.

17601 **-w *width*** Specify the maximum line length, in column positions (or bytes if **-b** is specified).
 17602 The results are unspecified if *width* is not a positive decimal number. The default
 17603 value shall be 80.

17604 **OPERANDS**

17605 The following operand shall be supported:

17606 *file* A path name of a text file to be folded. If no *file* operands are specified, the
 17607 standard input shall be used.

17608 **STDIN**

17609 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
 17610 section.

17611 **INPUT FILES**

17612 If the **-b** option is specified, the input files shall be text files except that the lines are not limited
17613 to {**LINE_MAX**} bytes in length. If the **-b** option is not specified, the input files shall be text files.

17614 **ENVIRONMENT VARIABLES**

17615 The following environment variables shall affect the execution of *fold*:

17616 **LANG** Provide a default value for the internationalization variables that are unset or null.
17617 If *LANG* is unset or null, the corresponding value from the implementation-
17618 defined default locale shall be used. If any of the internationalization variables
17619 contains an invalid setting, the utility shall behave as if none of the variables had
17620 been defined.

17621 **LC_ALL** If set to a non-empty string value, override the values of all the other
17622 internationalization variables.

17623 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
17624 characters (for example, single-byte as opposed to multi-byte characters in
17625 arguments and input files), and for the determination of the width in column
17626 positions each character would occupy on a constant-width font output device.

17627 **LC_MESSAGES**

17628 Determine the locale that should be used to affect the format and contents of
17629 diagnostic messages written to standard error.

17630 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

17631 **ASYNCHRONOUS EVENTS**

17632 Default.

17633 **STDOUT**

17634 The standard output shall be a file containing a sequence of characters whose order shall be
17635 preserved from the input files, possibly with inserted <newline> characters.

17636 **STDERR**

17637 Used only for diagnostic messages.

17638 **OUTPUT FILES**

17639 None.

17640 **EXTENDED DESCRIPTION**

17641 None.

17642 **EXIT STATUS**

17643 The following exit values shall be returned:

17644 0 All input files were processed successfully.

17645 >0 An error occurred.

17646 **CONSEQUENCES OF ERRORS**

17647 Default.

17648 **APPLICATION USAGE**

17649 The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths. The
17650 *cut* utility should be used when the number of lines (or records) needs to remain constant. The
17651 *fold* utility should be used when the contents of long lines need to be kept contiguous.

17652 The *fold* utility is frequently used to send text files to printers that truncate, rather than fold, lines
17653 wider than the printer is able to print (usually 80 or 132 column positions).

17654 **EXAMPLES**

17655 An example invocation that submits a file of possibly long lines to the printer (under the
17656 assumption that the user knows the line width of the printer to be assigned by *lp*):

```
17657 fold -w 132 bigfile | lp
```

17658 **RATIONALE**

17659 Although terminal input in canonical processing mode requires the erase character (frequently
17660 set to <backspace>) to erase the previous character (not byte or column position), terminal
17661 output is not buffered and is extremely difficult, if not impossible, to parse correctly; the
17662 interpretation depends entirely on the physical device that actually displays/prints/stores the
17663 output. In all known internationalized implementations, the utilities producing output for mixed
17664 column-width output assume that a <backspace> backs up one column position and outputs
17665 enough <backspace>s to return to the start of the character when <backspace> is used to
17666 provide local line motions to support underlining and emboldening operations. Since *fold*
17667 without the **-b** option is dealing with these same constraints, <backspace> is always treated as
17668 backing up one column position rather than backing up one character.

17669 Historical versions of the *fold* utility assumed 1 byte was one character and occupied one column
17670 position when written out. This is no longer always true. Since the most common usage of *fold* is
17671 believed to be folding long lines for output to limited-length output devices, this capability was
17672 preserved as the default case. The **-b** option was added so that applications could *fold* files with
17673 arbitrary length lines into text files that could then be processed by the standard utilities. Note
17674 that although the width for the **-b** option is in bytes, a line is never split in the middle of a
17675 character. (It is unspecified what happens if a width is specified that is too small to hold a single
17676 character found in the input followed by a <newline>.)

17677 The tab stops are hardcoded to be every eighth column to meet historical practice. No new
17678 method of specifying other tab stops was invented.

17679 **FUTURE DIRECTIONS**

17680 None.

17681 **SEE ALSO**

17682 *cut*

17683 **CHANGE HISTORY**

17684 First released in Issue 4.

17685 **Issue 6**

17686 The normative text is reworded to avoid use of the term “must” for application requirements.

17687 NAME

17688 fort77 — FORTRAN compiler (**FORTRAN**)

17689 SYNOPSIS

```
17690 FD fort77 [-c][-g][-L directory]... [-O optlevel][-o outfile][-s][-w]
17691 operand...
```

17692

17693 DESCRIPTION

17694 The *fort77* utility is the interface to the FORTRAN compilation system; it shall accept the full
 17695 FORTRAN-77 language defined by the ANSI X3.9-1978 standard. The system conceptually
 17696 consists of a compiler and link editor. The files referenced by *operands* are compiled and linked
 17697 to produce an executable file. It is unspecified whether the linking occurs entirely within the
 17698 operation of *fort77*; some systems may produce objects that are not fully resolved until the file is
 17699 executed.

17700 If the `-c` option is present, for all path name operands of the form *file.f*, the files:

17701 `$(basename pathname.f).o`

17702 shall be created or overwritten as the result of successful compilation. If the `-c` option is not
 17703 specified, it is unspecified whether such `.o` files are created or deleted for the *file.f* operands.

17704 If there are no options that prevent link editing (such as `-c`) and all operands compile and link
 17705 without error, the resulting executable file shall be written into the file named by the `-o` option
 17706 (if present) or to the file **a.out**. The executable file shall be created as specified in the System
 17707 Interfaces volume of IEEE Std. 1003.1-200x, except that the file permissions shall be set to:

17708 `S_IRWXO | S_IRWXG | S_IRWXU`

17709 and that the bits specified by the *umask* of the process shall be cleared.

17710 OPTIONS

17711 The *fort77* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 17712 12.2, Utility Syntax Guidelines, except that:

- 17713 • The `-l library` operands have the format of options, but their position within a list of
 17714 operands affects the order in which libraries are searched.
- 17715 • The order of specifying the multiple `-L` options is significant.
- 17716 • Portable applications shall specify each option separately; that is, grouping option letters (for
 17717 example, `-cg`) need not be recognized by all implementations.

17718 The following options shall be supported:

17719 `-c` Suppress the link-edit phase of the compilation, and do not remove any object files
 17720 that are produced.

17721 `-g` Produce symbolic information in the object or executable files; the nature of this
 17722 information is unspecified, and may be modified by implementation-defined
 17723 interactions with other options.

17724 `-s` Produce object or executable files, or both, from which symbolic and other
 17725 information not required for proper execution using the *exec* family of functions
 17726 defined in the System Interfaces volume of IEEE Std. 1003.1-200x has been
 17727 removed (stripped). If both `-g` and `-s` options are present, the action taken is
 17728 unspecified.

17729 `-o outfile` Use the path name *outfile*, instead of the default **a.out**, for the executable file
 17730 produced. If the `-o` option is present with `-c`, the result is unspecified.

17731 **-L *directory*** Change the algorithm of searching for the libraries named in **-I** operands to look in
 17732 the directory named by the *directory* path name before looking in the usual places.
 17733 Directories named in **-L** options shall be searched in the specified order. At least
 17734 ten instances of this option shall be supported in a single *fort77* command
 17735 invocation. If a directory specified by a **-L** option contains a file named **libf.a**, the
 17736 results are unspecified.

17737 **-O *optlevel*** Specify the level of code optimization. If the *optlevel* option-argument is the digit
 17738 '0', all special code optimizations shall be disabled. If it is the digit '1', the
 17739 nature of the optimization is unspecified. If the **-O** option is omitted, the nature of
 17740 the system's default optimization is unspecified. It is unspecified whether code
 17741 generated in the presence of the **-O 0** option is the same as that generated when
 17742 **-O** is omitted. Other *optlevel* values may be supported.

17743 **-w** Suppress warnings.

17744 Multiple instances of **-L** options can be specified.

17745 OPERANDS

17746 An *operand* is either in the form of a path name or the form **-I *library***. At least one operand of the
 17747 path name form shall be specified. The following operands shall be supported:

17748 ***file.f*** The path name of a FORTRAN source file to be compiled and optionally passed to
 17749 the link editor. The file name operand shall be of this form if the **-c** option is used.

17750 ***file.a*** A library of object files typically produced by *ar*, and passed directly to the link
 17751 editor. Implementations may recognize implementation-defined suffixes other
 17752 than **.a** as denoting object file libraries.

17753 ***file.o*** An object file produced by *fort77 -c* and passed directly to the link editor.
 17754 Implementations may recognize implementation-defined suffixes other than **.o** as
 17755 denoting object files.

17756 The processing of other files is implementation-defined.

17757 **-I *library*** (The letter ell.) Search the library named:

17758 *liblibrary.a*

17759 A library is searched when its name is encountered, so the placement of a **-I**
 17760 operand is significant. Several standard libraries can be specified in this manner, as
 17761 described in the EXTENDED DESCRIPTION section. Implementations may
 17762 recognize implementation-defined suffixes other than **.a** as denoting libraries.

17763 STDIN

17764 Not used.

17765 INPUT FILES

17766 The input file shall be one of the following: a text file containing FORTRAN source code; an
 17767 object file in the format produced by *fort77 -c*; or a library of object files, in the format produced
 17768 by archiving zero or more object files, using *ar*. Implementations may supply additional utilities
 17769 that produce files in these formats. Additional input files are implementation-defined.

17770 A **<tab>** character encountered within the first six characters on a line of source code shall cause
 17771 the compiler to interpret the following character as if it were the seventh character on the line
 17772 (that is, in column 7).

17773 **ENVIRONMENT VARIABLES**

17774 The following environment variables shall affect the execution of *fort77*:

17775 **LANG** Provide a default value for the internationalization variables that are unset or null.
 17776 If *LANG* is unset or null, the corresponding value from the implementation-
 17777 defined default locale shall be used. If any of the internationalization variables
 17778 contains an invalid setting, the utility shall behave as if none of the variables had
 17779 been defined.

17780 **LC_ALL** If set to a non-empty string value, override the values of all the other
 17781 internationalization variables.

17782 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 17783 characters (for example, single-byte as opposed to multi-byte characters in
 17784 arguments and input files).

17785 **LC_MESSAGES**
 17786 Determine the locale that should be used to affect the format and contents of
 17787 diagnostic messages written to standard error.

17788 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

17789 **TMPDIR** Determine the path name that should override the default directory for temporary
 17790 files, if any.

17791 **ASYNCHRONOUS EVENTS**

17792 Default.

17793 **STDOUT**

17794 Not used.

17795 **STDERR**

17796 Used only for diagnostic messages. If more than one *file* operand ending in *.f* (or possibly other
 17797 unspecified suffixes) is given, for each such file:

17798 "%s:\n", <*file*>

17799 may be written to allow identification of the diagnostic message with the appropriate input file.

17800 This utility may produce warning messages about certain conditions that do not warrant
 17801 returning an error (non-zero) exit value.

17802 **OUTPUT FILES**

17803 Object files, listing files and executable files shall be produced in unspecified formats.

17804 **EXTENDED DESCRIPTION**17805 **Standard Libraries**

17806 The *fort77* utility shall recognize the following **-l** operand for the standard library:

17807 **-l f** This library contains all library functions referenced in the ANSI X3.9-1978
 17808 standard. This operand shall not be required to be present to cause a search of this
 17809 library.

17810 In the absence of options that inhibit invocation of the link editor, such as **-c**, the *fort77* utility
 17811 shall cause the equivalent of a **-l f** operand to be passed to the link editor as the last **-l** operand,
 17812 causing it to be searched after all other object files and libraries are loaded.

17813 It is unspecified whether the library **libf.a** exists as a regular file. The implementation may
 17814 accept as **-l** operands names of objects that do not exist as regular files.

17815 **External Symbols**

17816 The FORTRAN compiler and link editor shall support the significance of external symbols up to
17817 a length of at least 31 bytes; case folding is permitted. The action taken upon encountering
17818 symbols exceeding the implementation-defined maximum symbol length is unspecified.

17819 The compiler and link editor shall support a minimum of 511 external symbols per source or
17820 object file, and a minimum of 4095 external symbols total. A diagnostic message is written to
17821 standard output if the implementation-defined limit is exceeded; other actions are unspecified.

17822 **EXIT STATUS**

17823 The following exit values shall be returned:

17824 0 Successful compilation or link edit.

17825 >0 An error occurred.

17826 **CONSEQUENCES OF ERRORS**

17827 When *fort77* encounters a compilation error, it shall write a diagnostic to standard error and
17828 continue to compile other source code operands. It shall return a non-zero exit status, but it is
17829 implementation-defined whether an object module is created. If the link edit is unsuccessful, a
17830 diagnostic message shall be written to standard error, and *fort77* shall exit with a non-zero
17831 status.

17832 **APPLICATION USAGE**

17833 None.

17834 **EXAMPLES**

17835 The following usage example compiles **xyz.f** and creates the executable file **foo**:

17836 `fort77 -o foo xyz.f`

17837 The following example compiles **xyz.f** and creates the object file **xyz.o**:

17838 `fort77 -c xyz.f`

17839 The following example compiles **xyz.f** and creates the executable file **a.out**:

17840 `fort77 xyz.f`

17841 The following example compiles **xyz.f**, links it with **b.o**, and creates the executable **a.out**:

17842 `fort77 xyz.f b.o`

17843 **RATIONALE**

17844 The name of this utility was chosen as *fort77* to parallel the renaming of the C compiler. The
17845 name *f77* was not chosen to avoid problems with historical implementations. The
17846 ANSI X3.9-1978 standard was selected as a normative reference because the ISO/IEC version of
17847 FORTRAN-77 has been superseded by the ISO/IEC 1539: 1990 standard (Fortran-90).

17848 The file inclusion and symbol definition **#define** mechanisms used by the *c99* utility were not
17849 included in this volume of IEEE Std. 1003.1-200x—even though they are commonly
17850 implemented—since there is no requirement that the FORTRAN compiler use the C
17851 preprocessor.

17852 The **-onetrip** option was not included in this volume of IEEE Std. 1003.1-200x, even though
17853 many historical compilers support it, because it is derived from FORTRAN-66; it is an
17854 anachronism that should not be perpetuated.

17855 Some implementations produce compilation listings. This aspect of FORTRAN has been left
17856 unspecified because there was controversy concerning the various methods proposed for
17857 implementing it: a **-V** option overlapped with historical vendor practice and a naming

17858 convention of creating files with `.l` suffixes collided with historical *lex* file naming practice.

17859 There is no `-I` option in this version of this volume of IEEE Std. 1003.1-200x to specify a directory
17860 for file inclusion. An `INCLUDE` directive has been a part of the Fortran-90 discussions, but an
17861 interface supporting that standard is not in the current scope.

17862 It is noted that many FORTRAN compilers produce an object module even when compilation
17863 errors occur; during a subsequent compilation, the compiler may patch the object module rather
17864 than recompiling all the code. Consequently, it is left to the implementor whether or not an
17865 object file is created.

17866 A reference to MIL-STD-1753 was removed from an early proposal in response to a request from
17867 the POSIX FORTRAN-binding standard developers. It was not the intention of the standard
17868 developers to require certification of the FORTRAN compiler, and IEEE Std. 1003.9-1992 does
17869 not specify the military standard or any special preprocessing requirements. Furthermore, use of
17870 that document would have been inappropriate for an international standard.

17871 The specification of optimization has been subject to changes through early proposals. At one
17872 time, `-O` and `-N` were Booleans: optimize and do not optimize (with an unspecified default).
17873 Some historical practice lead this to be changed to:

17874 `-O 0` No optimization.

17875 `-O 1` Some level of optimization.

17876 `-O n` Other, unspecified levels of optimization.

17877 It is not always clear whether “good code generation” is the same thing as optimization. Simple
17878 optimizations of local actions do not usually affect the semantics of a program. The `-O 0` option
17879 has been included to accommodate the very particular nature of scientific calculations in a
17880 highly optimized environment; compilers make errors. Some degree of optimization is expected,
17881 even if it is not documented here, and the ability to shut it off completely could be important
17882 when porting an application. An implementation may treat `-O 0` as “do less than normal” if it
17883 wishes, but this is only meaningful if any of the operations it performs can affect the semantics
17884 of a program. It is highly dependent on the implementation whether doing less than normal is
17885 logical. It is not the intent of the `-O 0` option to ask for inefficient code generation, but rather to
17886 assure that any semantically visible optimization is suppressed.

17887 The specification of standard library access is consistent with the C compiler specification.
17888 Implementations are not required to have `/usr/lib/libf.a`, as many historical implementations do,
17889 but if not they are required to recognize `f` as a token.

17890 External symbol size limits are in normative text; portable applications need to know these
17891 limits. However, the minimum maximum symbol length should be taken as a constraint on a
17892 portable application, not on an implementation, and consequently the action taken for a symbol
17893 exceeding the limit is unspecified. The minimum size for the external symbol table was added
17894 for similar reasons.

17895 The CONSEQUENCES OF ERRORS section clearly specifies the behavior of the compiler when
17896 compilation or link-edit errors occur. The behavior of several historical implementations was
17897 examined, and the choice was made to be silent on the status of the executable, or `a.out`, file in
17898 the face of compiler or linker errors. If a linker writes the executable file, then links it on disk
17899 with `lseek()`s and `write()`s, the partially linked executable file can be left on disk and its execute
17900 bits turned off if the link edit fails. However, if the linker links the image in memory before
17901 writing the file to disk, it need not touch the executable file (if it already exists) because the link
17902 edit fails. Since both approaches are historical practice, a portable application shall rely on the
17903 exit status of *fort77*, rather than on the existence or mode of the executable file.

- 17904 The `-g` and `-s` options are not specified as mutually-exclusive. Historically these two options
17905 have been mutually-exclusive, but because both are so loosely specified, it seemed appropriate
17906 to leave their interaction unspecified.
- 17907 The requirement that portable applications specify compiler options separately is to reserve the
17908 multi-character option name space for vendor-specific compiler options, which are known to
17909 exist in many historical implementations. Implementations are not required to recognize, for
17910 example, `-gc` as if it were `-g -c`; nor are they forbidden from doing so. The SYNOPSIS shows all
17911 of the options separately to highlight this requirement on applications.
- 17912 Echoing file names to standard error is considered a diagnostic message because it would
17913 otherwise be difficult to associate an error message with the erring file. They are described with
17914 “may” to allow implementations to use other methods of identifying files and to parallel the
17915 description in *c99*.
- 17916 **FUTURE DIRECTIONS**
- 17917 A compilation system based on the ISO/IEC 1539:1990 standard (Fortran-90) may be considered
17918 for a future issue; it may have a different utility name from *fort77*.
- 17919 **SEE ALSO**
- 17920 *ar, asa, c99, umask*
- 17921 **CHANGE HISTORY**
- 17922 First released in Issue 4.
- 17923 **Issue 6**
- 17924 This utility is now marked as part of the FORTRAN Development Utilities option.
- 17925 The normative text is reworded to avoid use of the term “must” for application requirements.

17926 **NAME**

17927 fuser — list process IDs of all processes that have one or more files open

17928 **SYNOPSIS**

17929 xSI fuser [-cfu] file ...

17930

17931 **DESCRIPTION**

17932 The *fuser* utility shall write to standard output the process IDs of processes running on the local
17933 system that have one or more named files open. For block special devices, all processes using
17934 any file on that device are listed.

17935 The *fuser* utility shall write to standard error additional information about the named files
17936 indicating how the file is being used.

17937 Any output for processes running on remote systems that have a named file open is unspecified.

17938 A user may need appropriate privilege to invoke the *fuser* utility.

17939 **OPTIONS**

17940 The *fuser* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
17941 12.2, Utility Syntax Guidelines.

17942 The following options shall be supported:

17943 **-c** The file is treated as a mount point and the utility shall report on any files open in
17944 the file system.

17945 **-f** The report shall be only for the named files.

17946 **-u** The user name, in parentheses, associated with each process ID written to standard
17947 output shall be written to standard error.

17948 **OPERANDS**

17949 The following operand shall be supported:

17950 *file* A path name on which the file or file system is to be reported.

17951 **STDIN**

17952 Not used.

17953 **INPUT FILES**

17954 The user database.

17955 **ENVIRONMENT VARIABLES**

17956 The following environment variables shall affect the execution of *fuser*:

17957 **LANG** Provide a default value for the internationalization variables that are unset or null.
17958 If *LANG* is unset or null, the corresponding value from the implementation-
17959 defined default locale shall be used. If any of the internationalization variables
17960 contain an invalid setting, the utility behaves as if none of the variables had been
17961 set.

17962 **LC_ALL** If set to a non-empty string value, override the values of all the other
17963 internationalization variables.

17964 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
17965 characters (for example, single-byte as opposed to multi-byte characters in
17966 arguments).

17967 **LC_MESSAGES**

17968 Determine the locale that should be used to affect the format and contents of

17969 diagnostic messages written to standard error.

17970 **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

17971 **ASYNCHRONOUS EVENTS**

17972 Default.

17973 **STDOUT**

17974 The *fuser* utility shall write the process ID for each process using each file given as an operand to standard output in the following format:

17975

17976 "%d", <*process_id*>

17977 **STDERR**

17978 The *fuser* utility shall write diagnostic messages to standard error.

17979 The *fuser* utility also shall write the following to standard error:

17980

- The path name of each named file is written followed immediately by a colon.
- For each process ID written to standard output, the character 'c' shall be written to standard error if the process is using the file as its current directory and the character 'r' shall be written to standard error if the process is using the file as its root directory. Implementations may write other alphabetic characters to indicate other uses of files.
- When the **-u** option is specified, characters indicating the use of the file shall be followed immediately by the user name, in parentheses, corresponding to the process' real user ID. If the user name cannot be resolved from the process' real user ID, the process' real user ID shall be written instead of the user name.

17989 When standard output and standard error are directed to the same file, the output shall be interspersed so that the file name appears at the start of each line, followed by the process ID and characters indicating the use of the file. Then, if the **-u** option is specified, the user name or user ID for each process using that file shall be written.

17990

17991

17992

17993 A <newline> character shall be written to standard error after the last output described above

17994 for each *file* operand.

17995 **OUTPUT FILES**

17996 None.

17997 **EXTENDED DESCRIPTION**

17998 None.

17999 **EXIT STATUS**

18000 The following exit values shall be returned:

18001 0 Successful completion.

18002 >0 An error occurred.

18003 **CONSEQUENCES OF ERRORS**

18004 Default.

18005 APPLICATION USAGE

18006 None.

18007 EXAMPLES

18008 The command:

18009 `fuser -fu .`

18010 writes to standard output the process IDs of processes that are using the current directory and

18011 writes to standard error an indication of how those processes are using the directory and the

18012 user names associated with the processes that are using the current directory.

18013 RATIONALE

18014 None.

18015 FUTURE DIRECTIONS

18016 None.

18017 SEE ALSO

18018 None.

18019 CHANGE HISTORY

18020 First released in Issue 5.

18021 **NAME**

18022 gencat — generate a formatted message catalog

18023 **SYNOPSIS**18024 XSI `gencat catfile msgfile...`

18025

18026 **DESCRIPTION**

18027 The *gencat* utility shall merge the message text source files *msgfile* into a formatted message
 18028 catalog *catfile*. The file *catfile* shall be created if it does not already exist. If *catfile* does exist, its
 18029 messages shall be included in the new *catfile*. If set and message numbers collide, the new
 18030 message text defined in *msgfile* shall replace the old message text currently contained in *catfile*.

18031 **OPTIONS**

18032 None.

18033 **OPERANDS**

18034 The following operands shall be supported:

18035 *catfile* A path name of the formatted message catalog. If '-' is specified, standard output
 18036 shall be used. The format of the message catalog produced is unspecified.

18037 *msgfile* A path name of a message text source file. If '-' is specified for an instance of
 18038 *msgfile*, standard input shall be used. The format of message text source files is
 18039 defined in the EXTENDED DESCRIPTION section.

18040 **STDIN**18041 The standard input shall not be used unless a *msgfile* operand is specified as '-'.18042 **INPUT FILES**

18043 The input files shall be text files.

18044 **ENVIRONMENT VARIABLES**18045 The following environment variables shall affect the execution of *gencat*:

18046 *LANG* Provide a default value for the internationalization variables that are unset or null.
 18047 If *LANG* is unset or null, the corresponding value from the implementation-
 18048 defined default locale shall be used. If any of the internationalization variables
 18049 contains an invalid setting, the utility shall behave as if none of the variables had
 18050 been defined.

18051 *LC_ALL* If set to a non-empty string value, override the values of all the other
 18052 internationalization variables.

18053 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 18054 characters (for example, single-byte as opposed to multi-byte characters in
 18055 arguments and input files).

18056 *LC_MESSAGES*

18057 Determine the locale that should be used to affect the format and contents of
 18058 diagnostic messages written to standard error.

18059 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

18060 **ASYNCHRONOUS EVENTS**

18061 Default.

18062 **STDOUT**

18063 The standard output shall not be used unless the *catfile* operand is specified as '- '.

18064 **STDERR**

18065 Used only for diagnostic messages.

18066 **OUTPUT FILES**

18067 None.

18068 **EXTENDED DESCRIPTION**

18069 The application shall ensure that the format of a message text source file is defined as follows.

18070 Note that the fields of a message text source line are separated by a single <blank> character.

18071 Any other <blank> characters are considered as being part of the subsequent field.

18072 **\$set *n* comment**

18073 This line specifies the set identifier of the following messages until the next **\$set** or
 18074 end-of-file appears. The *n* denotes the set identifier, which is defined as a number
 18075 in the range [1, {NL_SETMAX}] (see the <limits.h> header defined in the System
 18076 Interfaces volume of IEEE Std. 1003.1-200x). The application shall ensure that set
 18077 identifiers are presented in ascending order within a single source file, but need
 18078 not be contiguous. Any string following the set identifier shall be treated as a
 18079 comment. If no **\$set** directive is specified in a message text source file, all messages
 18080 shall be located in an implementation-defined default message set NL_SETD (see
 18081 the <nl_types.h> header defined in the System Interfaces volume of
 18082 IEEE Std. 1003.1-200x).

18083 **\$delset *n* comment**

18084 This line deletes message set *n* from an existing message catalog. The *n* denotes the
 18085 set number [1, {NL_SETMAX}]. Any string following the set number shall be
 18086 treated as a comment.

18087 **\$ comment** A line beginning with '\$ ' followed by a <blank> character shall be treated as a
 18088 comment.

18089 ***m* message-text**

18090 The *m* denotes the message identifier, which is defined as a number in the range [1,
 18091 {NL_MSGMAX}] (see the <limits.h> header defined in the System Interfaces
 18092 volume of IEEE Std. 1003.1-200x). The *message-text* shall be stored in the message
 18093 catalog with the set identifier specified by the last **\$set** directive, and with message
 18094 identifier *m*. If the *message-text* is empty, and a <blank> character field separator is
 18095 present, an empty string shall be stored in the message catalog. If a message source
 18096 line has a message number, but neither a field separator nor *message-text*, the
 18097 existing message with that number (if any) shall be deleted from the catalog. The
 18098 application shall ensure that message identifiers are in ascending order within a
 18099 single set, but need not be contiguous. The application shall ensure that the length
 18100 of *message-text* is in the range [0, {NL_TEXTMAX}] (see the <limits.h> header
 18101 defined in the System Interfaces volume of IEEE Std. 1003.1-200x).

18102 **\$quote *n*** This line specifies an optional quote character *c*, which can be used to surround
 18103 *message-text* so that trailing spaces or null (empty) messages are visible in a
 18104 message source line. By default, or if an empty **\$quote** directive is supplied, no
 18105 quoting of *message-text* shall be recognized.

18106 Empty lines in a message text source file shall be ignored. The effects of lines starting with any
 18107 character other than those defined above are implementation-defined.

18108 Text strings can contain the special characters and escape sequences defined in the following
18109 table:

18110

18111

18112

18113

18114

18115

18116

18117

18118

18119

Description	Symbol	Sequence
<newline>	NL(LF)	\n
Horizontal tab	HT	\t
<vertical-tab>	VT	\v
<backspace>	BS	\b
<carriage-return>	CR	\r
<form-feed>	FF	\f
Backslash	\	\\
Bit pattern	ddd	\ddd

18120

18121

18122

The escape sequence "\ddd" consists of backslash followed by one, two, or three octal digits, which shall be taken to specify the value of the desired character. If the character following a backslash is not one of those specified, the backslash shall be ignored.

18123

18124

Backslash ('\') followed by a <newline> character is also used to continue a string on the following line. Thus, the following two lines describe a single message string:

18125

18126

```
1 This line continues \
to the next line
```

18127

which is equivalent to:

18128

```
1 This line continues to the next line
```

18129 EXIT STATUS

18130

The following exit values shall be returned:

18131

0 Successful completion.

18132

>0 An error occurred.

18133 CONSEQUENCES OF ERRORS

18134

Default.

18135 APPLICATION USAGE

18136

18137

18138

Message catalogs produced by *gencat* are binary encoded, meaning that their portability cannot be guaranteed between different types of machine. Thus, just as C programs need to be recompiled for each type of machine, so message catalogs must be recreated via *gencat*.

18139 EXAMPLES

18140

None.

18141 RATIONALE

18142

None.

18143 FUTURE DIRECTIONS

18144

None.

18145 SEE ALSO

18146

iconv, the System Interfaces volume of IEEE Std. 1003.1-200x, <**limits.h**>

18147 CHANGE HISTORY

18148

First released in Issue 3.

18149 **Issue 4**

18150 Format reorganized.

18151 Internationalized environment variable support mandated.

18152 **Issue 6**

18153 The normative text is reworded to avoid use of the term “must” for application requirements.

18154 NAME

18155 `get` — get a version of an SCCS file (**DEVELOPMENT**)

18156 SYNOPSIS

```
18157 xSI get [-begkmlPst][-c cutoff][-i list][-r SID][-x list] file...
```

18158

18159 DESCRIPTION

18160 The `get` utility shall generate a text file from each named SCCS *file* according to the specifications
18161 given by its options.

18162 The generated text is normally written into a file called the **g-file** whose name is derived from
18163 the SCCS file name by simply removing the leading "s . ".

18164 OPTIONS

18165 The `get` utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
18166 12.2, Utility Syntax Guidelines.

18167 The following options shall be supported:

18168 **-r *SID*** Indicate the SCCS Identification String (*SID*) of the version (*delta*) of an SCCS file
18169 to be retrieved. The table shows, for the most useful cases, what version of an
18170 SCCS file is retrieved (as well as the *SID* of the version to be eventually created by
18171 *delta* if the **-e** option is also used), as a function of the *SID* specified.

18172 **-c *cutoff*** Indicate the *cutoff* date-time, in the form:

```
18173 YY[MM[DD[HH[MM[SS] ] ] ] ]
```

18174 For the *YY* component, values in the range [69-99] shall refer to years in the
18175 twentieth century (1969 to 1999 inclusive); values in the range [00-68] shall refer to
18176 years in the twenty-first century (2000 to 2068 inclusive).

18177 No changes (*deltas*) to the SCCS file that were created after the specified *cutoff*
18178 date-time are included in the generated text file. Units omitted from the date-time
18179 default to their maximum possible values; for example, **-c 7502** is equivalent to **-c**
18180 **750228235959**.

18181 Any number of non-numeric characters may separate the various 2-digit pieces of
18182 the *cutoff* date-time. This feature allows the user to specify a *cutoff* date in the form:
18183 **-c "77/2/2 9:22:25"**.

18184 **-e** Indicate that the `get` is for the purpose of editing or making a change (*delta*) to the
18185 SCCS file via a subsequent use of *delta*. The **-e** option used in a `get` for a particular
18186 version (*SID*) of the SCCS file shall prevent further `get` commands from editing on
18187 the same *SID* until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file.
18188 Concurrent use of `get -e` for different *SIDs* is always allowed.

18189 If the **g-file** generated by `get` with a **-e** option is accidentally ruined in the process
18190 of editing, it may be regenerated by re-executing the `get` command with the **-k**
18191 option in place of the **-e** option.

18192 SCCS file protection specified via the ceiling, floor, and authorized user list stored
18193 in the SCCS file shall be enforced when the **-e** option is used.

18194 **-b** Use with the **-e** option to indicate that the new *delta* should have an *SID* in a new
18195 branch as shown in the table below. This option shall be ignored if the **b** flag is not
18196 present in the file or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that
18197 has no successors on the SCCS file tree.)

- 18198 **Note:** A branch delta may always be created from a non-leaf delta.
- 18199 **-i list** Indicate a *list* of deltas to be included (forced to be applied) in the creation of the
18200 generated file. The *list* has the following syntax:
- 18201 <list> ::= <range> | <list> , <range>
18202 <range> ::= SID | SID - SID
- 18203 SID, the SCCS Identification of a delta, may be in any form shown in the "SID
18204 Specified" column of the table in the EXTENDED DESCRIPTION section. Partial
18205 SIDs are interpreted as shown in the "SID Retrieved" column of the table.
- 18206 **-x list** Indicate a *list* of deltas to be excluded (forced not to be applied) in the creation of
18207 the generated file. See the **-i** option for the *list* format.
- 18208 **-k** Suppress replacement of identification keywords (see below) in the retrieved text
18209 by their value. The **-k** option is implied by the **-e** option.
- 18210 **-l** Write a delta summary into an **l-file**.
- 18211 **-L** Write a delta summary to standard output. All informative output that normally is
18212 written to standard output shall be written to standard error instead, unless the **-s**
18213 option is used, in which case it shall be suppressed.
- 18214 **-p** Write the text retrieved from the SCCS file to the standard output. No **g-file** shall
18215 be created. All informative output that normally goes to the standard output shall
18216 go to standard error instead, unless the **-s** option is used, in which case it
18217 disappears.
- 18218 **-s** Suppress all informative output normally written to standard output. However,
18219 fatal error messages (which shall always be written to the standard error) remain
18220 unaffected.
- 18221 **-m** Precede each text line retrieved from the SCCS file by the SID of the delta that
18222 inserted the text line in the SCCS file. The format is:
- 18223 "%s\t%s", <SID>, <text line>
- 18224 **-n** Precede each generated text line with the %M% identification keyword value (see
18225 below). The format is:
- 18226 "%s\t%s", <%M% value>, <text line>
- 18227 When both the **-m** and **-n** options are used, the <text line> shall be replaced by the
18228 **-m** option-generated format.
- 18229 **-g** Suppress the actual retrieval of text from the SCCS file. It is primarily used to
18230 generate an **l-file**, or to verify the existence of a particular SID.
- 18231 **-t** Use to access the most recently created (top) delta in a given release (for example,
18232 **-r 1**), or release and level (for example, **-r 1.2**).

18233 **OPERANDS**

18234 The following operands shall be supported:

- 18235 **file** A path name of an existing SCCS file or a directory. If *file* is a directory, the *get*
18236 utility shall behave as though each file in the directory were specified as a named
18237 file, except that non-SCCS files (last component of the path name does not begin
18238 with **s.**) and unreadable files shall be silently ignored.
- 18239 If a single instance *file* is specified as **'-'**, the standard input is read; each line of
18240 the standard input is taken to be the name of an SCCS file to be processed. Non-

- 18241 SCCS files and unreadable files shall be silently ignored.
- 18242 **STDIN**
- 18243 The standard input shall be a text file used only if the *file* operand is specified as '-'. Each line
18244 of the text file shall be interpreted as an SCCS path name.
- 18245 **INPUT FILES**
- 18246 The SCCS files are files of an unspecified format.
- 18247 **ENVIRONMENT VARIABLES**
- 18248 The following environment variables shall affect the execution of *get*:
- 18249 *LANG* Provide a default value for the internationalization variables that are unset or null.
18250 If *LANG* is unset or null, the corresponding value from the implementation-
18251 defined default locale shall be used. If any of the internationalization variables
18252 contains an invalid setting, the utility shall behave as if none of the variables had
18253 been defined.
- 18254 *LC_ALL* If set to a non-empty string value, override the values of all the other
18255 internationalization variables.
- 18256 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
18257 characters (for example, single-byte as opposed to multi-byte characters in
18258 arguments and input files).
- 18259 *LC_MESSAGES*
- 18260 Determine the locale that should be used to affect the format and contents of
18261 diagnostic messages written to standard error, and informative messages written
18262 to standard output (or standard error, if the **-p** option is used).
- 18263 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 18264 **ASYNCHRONOUS EVENTS**
- 18265 Default.
- 18266 **STDOUT**
- 18267 For each file processed, *get* shall write to standard output the SID being accessed and the number
18268 of lines retrieved from the SCCS file, in the following format:
- 18269 "*%s\n%d lines\n*", *<SID>*, *<number of lines>*
- 18270 If the **-e** option is used, the SID of the delta to be made shall appear after the SID accessed and
18271 before the number of lines generated, in the POSIX locale:
- 18272 "*%s\nnew delta %s\n%d\n*", *<SID accessed>*, *<SID to be made>*,
18273 *<number of lines>*
- 18274 If there is more than one named file or if a directory or standard input is named, each path name
18275 shall be written before each of the lines shown in one of the preceding formats:
- 18276 "*\n%s:\n*", *<pathname>*
- 18277 If the **-L** option is used, a delta summary shall be written following the format specified below
18278 for **l-files**.
- 18279 If the **-i** option is used, included deltas are listed following the notation, in the POSIX locale:
- 18280 "*Included:\n*"
- 18281 If the **-x** option is used, excluded deltas are listed following the notation, in the POSIX locale:

18282 "Excluded:\n"

18283 If the **-p** or **-L** options are specified, the standard output consists of the text retrieved from the
18284 SCCS file.

18285 **STDERR**

18286 The standard error shall be used only for diagnostic messages, except if the **-p** or **-L** options are
18287 specified, it includes all informative messages normally sent to standard output.

18288 **OUTPUT FILES**

18289 Several auxiliary files may be created by *get*. These files are known generically as the **g-file**, **l-**
18290 **file**, **p-file**, and **z-file**. The letter before the hyphen is called the *tag*. An auxiliary file name is
18291 formed from the SCCS file name: the application shall ensure that the last component of all
18292 SCCS file names is of the form *s.module-name*; the auxiliary files are named by replacing the
18293 leading *s* with the tag. The **g-file** is an exception to this scheme: the **g-file** is named by removing
18294 the *s*. prefix. For example, for *s.xyz.c*, the auxiliary file names would be *xyz.c*, *l.xyz.c*, *p.xyz.c*,
18295 and *z.xyz.c*, respectively.

18296 The **g-file**, which contains the generated text, is created in the current directory (unless the **-p**
18297 option is used). A **g-file** is created in all cases, whether or not any lines of text were generated by
18298 the *get*. It is owned by the real user. If the **-k** option is used or implied, it is writable by the
18299 owner only (read-only for everyone else); otherwise, it is read-only. Only the real user need have
18300 write permission in the current directory.

18301 The **l-file** contains a table showing which deltas were applied in generating the retrieved text.
18302 The **l-file** is created in the current directory if the **-l** option is used; it is read-only and it is
18303 owned by the real user. Only the real user need have write permission in the current directory.

18304 Lines in the **l-file** have the following format:

```
18305 "%c%c%cΔ%s\t%sΔ%s\n", <code1>, <code2>, <code3>,  
18306 <SID>, <date-time>, <login>
```

18307 where the entries are:

18308 **<code1>** A <space> character if the delta was applied; ' * ' otherwise.

18309 **<code2>** A <space> character if the delta was applied or was not applied and ignored; ' * '
18310 if the delta was not applied and was not ignored.

18311 **<code3>** A character indicating a special reason why the delta was or was not applied:

18312 **I** Included.

18313 **X** Excluded.

18314 **C** Cut off (by a **-c** option).

18315 **<date-time>** Date and time (using the *date* utility's %y/%m/%d %T format) of creation.

18316 **<login>** Login name of person who created *delta*.

18317 The comments and MR data shall follow on subsequent lines, indented one <tab> character. A
18318 blank line terminates each entry.

18319 The **p-file** is used to pass information resulting from a *get* with a **-e** option along to *delta*. Its
18320 contents are also used to prevent a subsequent execution of *get* with a **-e** option for the same SID
18321 until *delta* is executed or the joint edit flag, **j**, is set in the SCCS file. The **p-file** shall be created in
18322 the directory containing the SCCS file and the application shall ensure that the effective user has
18323 write permission in that directory. It is writable by owner only, and it is owned by the effective
18324 user. Each line in the **p-file** has the following format:

18325 "%sΔ%sΔ%sΔ%s%s\n", <g-file SID>,
 18326 <SID of new delta>, <login-name of real user>,
 18327 <date-time>, <i-value>, <x-value>

18328 where <i-value> uses the format " " if no **-i** option was specified, and uses the format:

18329 "Δ-i%s", <-i option option-argument>

18330 if a **-i** option was specified and <x-value> uses the format " " if no **-x** option was specified, and
 18331 uses the format:

18332 "Δ-x%s", <-x option option-argument>

18333 if a **-x** option was specified. There can be an arbitrary number of lines in the **p-file** at any time;
 18334 no two lines can have the same new delta SID.

18335 The **z-file** serves as a lock-out mechanism against simultaneous updates. Its contents are the
 18336 binary process ID of the command (that is, *get*) that created it. The **z-file** is created in the
 18337 directory containing the SCCS file for the duration of *get*. The same protection restrictions as
 18338 those for the **p-file** apply for the **z-file**. The **z-file** shall be created read-only.

18339 EXTENDED DESCRIPTION

Determination of SCCS Identification String				
SID* Specified	-b Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
18343 none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
18344 none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
18345 R	no	R > mR	mR.mL	R.1***
18346 R	no	R = mR	mR.mL	mR.(mL+1)
18347 R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
18348 R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
18349 R	-	R < mR and R does not exist	hR.mL**	hR.mL.(mB+1).1
18351 R	-	Trunk successor in release > R and R exists	R.mL	R.mL.(mB+1).1
18353 R.L	no	No trunk successor	R.L	R.(L+1)
18354 R.L	yes	No trunk successor	R.L	R.L.(mB+1).1
18355 R.L	-	Trunk successor in release ≥ R	R.L	R.L.(mB+1).1
18357 R.L.B	no	No branch successor	R.L.B.mS	R.L.B.(mS+1)
18358 R.L.B	yes	No branch successor	R.L.B.mS	R.L.(mB+1).1
18359 R.L.B.S	no	No branch successor	R.L.B.S	R.L.B.(S+1)
18360 R.L.B.S	yes	No branch successor	R.L.B.S	R.L.(mB+1).1
18361 R.L.B.S	-	Branch successor	R.L.B.S	R.L.(mB+1).1

18362 * R, L, B, and S are the release, level, branch, and sequence components of the SID,
 18363 respectively; m means maximum. Thus, for example, R.mL means "the maximum level
 18364 number within release R"; R.L.(mB+1).1 means "the first sequence number on the new
 18365 branch (that is, maximum branch number plus one) of level L within release R". Note
 18366 that if the SID specified is of the form R.L, R.L.B, or R.L.B.S, each of the specified
 18367 components shall exist.

18368	**	hR is the highest existing release that is lower than the specified, nonexistent, release R.
18369	***	This is used to force creation of the first delta in a new release.
18370	†	The -b option is effective only if the b flag is present in the file. An entry of '–' means
18371		“irrelevant”.
18372	‡	This case applies if the d (default SID) flag is not present in the file. If the d flag is
18373		present in the file, then the SID obtained from the d flag is interpreted as if it had been
18374		specified on the command line. Thus, one of the other cases in this table applies.

18375 Identification Keywords

18376 Identifying information shall be inserted into the text retrieved from the SCCS file by replacing
 18377 identification keywords with their value wherever they occur. The following keywords may be
 18378 used in the text stored in an SCCS file:

18379	%M%	Module name: either the value of the m flag in the file, or if absent, the name of the
18380		SCCS file with the leading s. removed.
18381	%I%	SCCS identification (SID) (%R%.%L% or %R%.%L%.%B%.%S%) of the retrieved
18382		text.
18383	%R%	Release.
18384	%L%	Level.
18385	%B%	Branch.
18386	%S%	Sequence.
18387	%D%	Current date (YY/MM/DD).
18388	%H%	Current date (MM/DD/YY).
18389	%T%	Current time (HH:MM:SS).
18390	%E%	Date newest applied delta was created (YY/MM/DD).
18391	%G%	Date newest applied delta was created (MM/DD/YY).
18392	%U%	Time newest applied delta was created (HH:MM:SS).
18393	%Y%	Module type: value of the t flag in the SCCS file.
18394	%F%	SCCS file name.
18395	%P%	SCCS absolute path name.
18396	%Q%	The value of the q flag in the file.
18397	%C%	Current line number. This keyword is intended for identifying messages output by
18398		the program, such as “this should not have happened” type errors. It is not
18399		intended to be used on every line to provide sequence numbers.
18400	%Z%	The four-character string "@(#)" recognizable by <i>what</i> .
18401	%W%	A shorthand notation for constructing <i>what</i> strings:
18402		%W%=%Z%%M%<tab>%I%
18403	%A%	Another shorthand notation for constructing <i>what</i> strings:
18404		%A%=%Z%%Y%%M%%I%%Z%

18405 **EXIT STATUS**

18406 The following exit values shall be returned:

18407 0 Successful completion.

18408 >0 An error occurred.

18409 **CONSEQUENCES OF ERRORS**

18410 Default.

18411 **APPLICATION USAGE**

18412 None.

18413 **EXAMPLES**

18414 None.

18415 **RATIONALE**

18416 None.

18417 **FUTURE DIRECTIONS**

18418 The **-lp** option may be withdrawn in a future issue.

18419 **SEE ALSO**

18420 *admin, delta, prs, what*

18421 **CHANGE HISTORY**

18422 First released in Issue 2.

18423 **Issue 4**

18424 Format reorganized.

18425 Exceptions to Utility Syntax Guidelines conformance noted.

18426 Internationalized environment variable support mandated.

18427 **Issue 5**

18428 Correction to the first format string in STDOUT.

18429 The interpretation of the *YY* component of the **-c cutoff** argument is noted.

18430 **Issue 6**

18431 The obsolescent SYNOPSIS is removed, removing the **-lp** option.

18432 The Open Group corrigenda item U025/5 has been applied, correcting text in the OPTIONS section.

18434 The normative text is reworded to avoid use of the term “must” for application requirements. |

18435 The normative text is reworded to emphasise the term “shall” for implementation requirements. |

18436 The Open Group corrigenda item U048/1 has been applied. |

18437 **NAME**

18438 getconf — get configuration values

18439 **SYNOPSIS**18440 getconf [*-v specification*] *system_var*18441 getconf [*-v specification*] *path_var pathname*18442 **DESCRIPTION**18443 In the first synopsis form, the *getconf* utility shall write to the standard output the value of the
18444 variable specified by the *system_var* operand.18445 In the second synopsis form, the *getconf* utility shall write to the standard output the value of the
18446 variable specified by the *path_var* operand for the path specified by the *pathname* operand.18447 The value of each configuration variable shall be determined as if it were obtained by calling the
18448 function from which it is defined to be available by this volume of IEEE Std. 1003.1-200x or by
18449 the System Interfaces volume of IEEE Std. 1003.1-200x (see the OPERANDS section). The value
18450 shall reflect conditions in the current operating environment.18451 **OPTIONS**18452 The *getconf* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
18453 12.2, Utility Syntax Guidelines.

18454 The following option shall be supported:

18455 *-v specification*18456 Indicate a specific specification and version for which configuration variables shall
18457 be determined. If this option is not specified, the values returned correspond to an
18458 implementation default conforming compilation environment.

18459 If the command:

18460 getconf _POSIX_V6_ILP32_OFF32

18461 does not write "-1\n" or "undefined\n" to standard output, then commands of
18462 the form:

18463 getconf -v POSIX_V6_ILP32_OFF32 ...

18464 determine values for configuration variables corresponding to the
18465 POSIX_V6_ILP32_OFF32 compilation environment specified in *c99* (on page 2425),
18466 EXTENDED DESCRIPTION.

18467 If the command:

18468 getconf _POSIX_V6_ILP32_OFFBIG

18469 does not write "-1\n" or "undefined\n" to standard output, then commands of
18470 the form:

18471 getconf -v POSIX_V6_ILP32_OFFBIG ...

18472 determine values for configuration variables corresponding to the
18473 POSIX_V6_ILP32_OFFBIG compilation environment specified in *c99* (on page
18474 2425), EXTENDED DESCRIPTION.

18475 If the command:

18476 getconf _POSIX_V6_LP64_OFF64

18477 does not write "-1\n" or "undefined\n" to standard output, then commands of
18478 the form:

18479 `getconf -v POSIX_V6_LP64_OFF64 ...`

18480 determine values for configuration variables corresponding to the
18481 `POSIX_V6_LP64_OFF64` compilation environment specified in *c99* (on page 2425),
18482 EXTENDED DESCRIPTION.

18483 If the command:

18484 `getconf _POSIX_V6_LPBIG_OFFBIG`

18485 does not write "-1\n" or "undefined\n" to standard output, then commands of
18486 the form:

18487 `getconf -v POSIX_V6_LPBIG_OFFBIG ...`

18488 determine values for configuration variables corresponding to the
18489 `POSIX_V6_LPBIG_OFFBIG` compilation environment specified in *c99* (on page
18490 2425), EXTENDED DESCRIPTION.

18491 OPERANDS

18492 The following operands shall be supported:

18493 *path_var* A name of a configuration variable. All of the variables in the *pathconf()* function
18494 defined in the System Interfaces volume of IEEE Std. 1003.1-200x are supported
18495 and the implementation may add other local variables.

18496 *pathname* A path name for which the variable specified by *path_var* is to be determined.

18497 *system_var* A name of a configuration variable. All of the variables in the *confstr()* and
18498 *sysconf()* functions defined in the System Interfaces volume of
18499 IEEE Std. 1003.1-200x shall be supported and the implementation may add other
18500 local values.

18501 When the symbol listed in the first column of the following table is used as the
18502 *system_var* operand, *getconf* yields the same value as *confstr()* when called with the
18503 value in the second column:

<i>system_var</i>	<i>confstr()</i> Name Value
18506 PATH	_CS_PATH
18507 POSIX_V6_ILP32_OFF32_CFLAGS	_CS_POSIX_V6_ILP32_OFF32_CFLAGS
18508 POSIX_V6_ILP32_OFF32_LDFLAGS	_CS_POSIX_V6_ILP32_OFF32_LDFLAGS
18509 POSIX_V6_ILP32_OFF32_LIBS	_CS_POSIX_V6_ILP32_OFF32_LIBS
18510 POSIX_V6_ILP32_OFF32_LINTFLAGS	_CS_POSIX_V6_ILP32_OFF32_LINTFLAGS
18511 POSIX_V6_ILP32_OFFBIG_CFLAGS	_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS
18512 POSIX_V6_ILP32_OFFBIG_LDFLAGS	_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS
18513 POSIX_V6_ILP32_OFFBIG_LIBS	_CS_POSIX_V6_ILP32_OFFBIG_LIBS
18514 POSIX_V6_ILP32_OFFBIG_LINTFLAGS	_CS_POSIX_V6_ILP32_OFFBIG_LINTFLAGS
18515 POSIX_V6_LP64_OFF64_CFLAGS	_CS_POSIX_V6_LP64_OFF64_CFLAGS
18516 POSIX_V6_LP64_OFF64_LDFLAGS	_CS_POSIX_V6_LP64_OFF64_LDFLAGS
18517 POSIX_V6_LP64_OFF64_LIBS	_CS_POSIX_V6_LP64_OFF64_LIBS
18518 POSIX_V6_LP64_OFF64_LINTFLAGS	_CS_POSIX_V6_LP64_OFF64_LINTFLAGS
18519 POSIX_V6_LPBIG_OFFBIG_CFLAGS	_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS
18520 POSIX_V6_LPBIG_OFFBIG_LDFLAGS	_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS
18521 POSIX_V6_LPBIG_OFFBIG_LIBS	_CS_POSIX_V6_LPBIG_OFFBIG_LIBS

18522

18523

18524

18525 XSI

18526

18527

18528

18529

18530

18531

18532

18533

18534

18535

18536

18537

18538

18539

18540

<i>system_var</i>	<i>confstr() Name Value</i>
POSIX_V6_LPBIG_OFFBIG_LINTFLAGS	_CS_POSIX_V6_LPBIG_OFFBIG_LINTFLAGS
XBS5_ILP32_OFF32_CFLAGS (LEGACY)	_CS_XBS5_ILP32_OFF32_CFLAGS
XBS5_ILP32_OFF32_LDFLAGS (LEGACY)	_CS_XBS5_ILP32_OFF32_LDFLAGS
XBS5_ILP32_OFF32_LIBS (LEGACY)	_CS_XBS5_ILP32_OFF32_LIBS
XBS5_ILP32_OFF32_LINTFLAGS (LEGACY)	_CS_XBS5_ILP32_OFF32_LINTFLAGS
XBS5_ILP32_OFFBIG_CFLAGS (LEGACY)	_CS_XBS5_ILP32_OFFBIG_CFLAGS
XBS5_ILP32_OFFBIG_LDFLAGS (LEGACY)	_CS_XBS5_ILP32_OFFBIG_LDFLAGS
XBS5_ILP32_OFFBIG_LIBS (LEGACY)	_CS_XBS5_ILP32_OFFBIG_LIBS
XBS5_ILP32_OFFBIG_LINTFLAGS (LEGACY)	_CS_XBS5_ILP32_OFFBIG_LINTFLAGS
XBS5_LP64_OFF64_CFLAGS (LEGACY)	_CS_XBS5_LP64_OFF64_CFLAGS
XBS5_LP64_OFF64_LDFLAGS (LEGACY)	_CS_XBS5_LP64_OFF64_LDFLAGS
XBS5_LP64_OFF64_LIBS (LEGACY)	_CS_XBS5_LP64_OFF64_LIBS
XBS5_LP64_OFF64_LINTFLAGS (LEGACY)	_CS_XBS5_LP64_OFF64_LINTFLAGS
XBS5_LPBIG_OFFBIG_CFLAGS (LEGACY)	_CS_XBS5_LPBIG_OFFBIG_CFLAGS
XBS5_LPBIG_OFFBIG_LDFLAGS (LEGACY)	_CS_XBS5_LPBIG_OFFBIG_LDFLAGS
XBS5_LPBIG_OFFBIG_LIBS (LEGACY)	_CS_XBS5_LPBIG_OFFBIG_LIBS
XBS5_LPBIG_OFFBIG_LINTFLAGS (LEGACY)	_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS

18541 **STDIN**

18542 Not used.

18543 **INPUT FILES**

18544 None.

18545 **ENVIRONMENT VARIABLES**18546 The following environment variables shall affect the execution of *getconf*:

18547 *LANG* Provide a default value for the internationalization variables that are unset or null.
 18548 If *LANG* is unset or null, the corresponding value from the implementation-
 18549 defined default locale shall be used. If any of the internationalization variables
 18550 contains an invalid setting, the utility shall behave as if none of the variables had
 18551 been defined.

18552 *LC_ALL* If set to a non-empty string value, override the values of all the other
 18553 internationalization variables.

18554 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 18555 characters (for example, single-byte as opposed to multi-byte characters in
 18556 arguments).

18557 *LC_MESSAGES*
 18558 Determine the locale that should be used to affect the format and contents of
 18559 diagnostic messages written to standard error.

18560 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

18561 **ASYNCHRONOUS EVENTS**

18562 Default.

18563 **STDOUT**

18564 If the specified variable is defined on the system and its value is described to be available from
 18565 the *confstr()* function defined in the System Interfaces volume of IEEE Std. 1003.1-200x, its value
 18566 shall be written in the following format:

18567 "%s\n", <value>

18568 Otherwise, if the specified variable is defined on the system, its value shall be written in the
18569 following format:

18570 "%d\n", <value>

18571 If the specified variable is valid, but is undefined on the system, *getconf* shall write using the
18572 following format:

18573 "undefined\n"

18574 If the variable name is invalid or an error occurs, nothing shall be written to standard output.

18575 **STDERR**

18576 Used only for diagnostic messages.

18577 **OUTPUT FILES**

18578 None.

18579 **EXTENDED DESCRIPTION**

18580 None.

18581 **EXIT STATUS**

18582 The following exit values shall be returned:

18583 0 The specified variable is valid and information about its current state was written
18584 successfully.

18585 >0 An error occurred.

18586 **CONSEQUENCES OF ERRORS**

18587 Default.

18588 **APPLICATION USAGE**

18589 None.

18590 **EXAMPLES**

18591 The following example illustrates the value of {NGROUPS_MAX}:

18592 `getconf NGROUPS_MAX`

18593 The following example illustrates the value of {NAME_MAX} for a specific directory:

18594 `getconf NAME_MAX /usr`

18595 The following example shows how to deal more carefully with results that might be unspecified:

```
18596 if value=$(getconf PATH_MAX /usr); then
18597     if [ "$value" = "undefined" ]; then
18598         echo PATH_MAX in /usr is infinite.
18599     else
18600         echo PATH_MAX in /usr is $value.
18601     fi
18602 else
18603     echo Error in getconf.
18604 fi
```

18605 Note that:

18606 `sysconf(_SC_POSIX_C_BIND);`

18607 and:

18608 `system("getconf POSIX2_C_BIND");`

18609 in a C program could give different answers. The `sysconf()` call supplies a value that corresponds
18610 to the conditions when the program was either compiled or executed, depending on the
18611 implementation; the `system()` call to `getconf` always supplies a value corresponding to conditions
18612 when the program is executed.

18613 RATIONALE

18614 The original need for this utility, and for the `confstr()` function, was to provide a way of finding
18615 the configuration-defined default value for the `PATH` environment variable. Since `PATH` can be
18616 modified by the user to include directories that could contain utilities replacing the standard
18617 utilities, shell scripts need a way to determine the system-supplied `PATH` environment variable
18618 value that contains the correct search path for the standard utilities. It was later suggested that
18619 access to the other variables described in this volume of IEEE Std. 1003.1-200x could also be
18620 useful to applications.

18621 This functionality of `getconf` would not be adequately subsumed by another command such as:

18622 `grep var /etc/conf`

18623 because such a strategy would provide correct values for neither those variables that can vary at
18624 runtime, nor those that can vary depending on the path.

18625 Early proposal versions of `getconf` specified exit status 1 when the specified variable was valid,
18626 but not defined on the system. The output string "undefined" is now used to specify this case
18627 with exit code 0 because so many things depend on an exit code of zero when an invoked utility
18628 is successful.

18629 FUTURE DIRECTIONS

18630 None.

18631 SEE ALSO

18632 `c99`, the System Interfaces volume of IEEE Std. 1003.1-200x, `confstr()`, `pathconf()`, `sysconf()`

18633 CHANGE HISTORY

18634 First released in Issue 4.

18635 Issue 4, Version 2

18636 The following changes are made in the table of values for `system_var`:

- 18637 • Names beginning with `POSIX_` are changed to begin with `_POSIX_`.
- 18638 • Names beginning with `XOPEN_` are changed to begin with `_XOPEN_`.
- 18639 • `{MN_NMAX}` is changed to `{NL_MAX}`.
- 18640 • `{NL_SET_MAX}` is changed to `{NL_SETMAX}`.
- 18641 • `{NL_TEXT_MAX}` is changed to `{NL_TEXTMAX}`.
- 18642 • The `_XOPEN_CRYPT`, `_XOPEN_ENH_I18N`, and `_XOPEN_SHM` configuration variables are
18643 added to the list.

18644 Issue 5

18645 In the OPERANDS section:

- 18646 • `{NL_MAX}` is changed to `{NL_NMAX}`.
- 18647 • Entries beginning `NL_` are deleted from the list of standard configuration variables.
- 18648 • The list of variables previously marked `UX` is merged with the list marked `EX`.

- 18649 • Operands are added to support new Option Groups.
- 18650 • Operands are added so that *getconf* can determine supported programming environments.
- 18651 **Issue 6**
- 18652 The Open Group corrigenda item U029/4 has been applied, correcting the example command in
- 18653 the last paragraph of the OPTIONS section.
- 18654 The following new requirements on POSIX implementations derive from alignment with the
- 18655 Single UNIX Specification:
- 18656 • Operands are added to determine supported programming environments.
- 18657 This reference page is updated for alignment with the ISO/IEC 9899: 1999 standard. Specifically,
- 18658 new macros for *c99* programming environments are introduced.

18659 **NAME**

18660 getopts — parse utility options

18661 **SYNOPSIS**

18662 `getopts optstring name [arg...]`

18663 **DESCRIPTION**

18664 The *getopts* utility can be used to retrieve options and option-arguments from a list of
 18665 parameters. It shall support the Utility Syntax Guidelines 3 to 10, inclusive, described in the Base
 18666 Definitions volume of IEEE Std. 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

18667 Each time it is invoked, the *getopts* utility shall place the value of the next option in the shell
 18668 variable specified by the *name* operand and the index of the next argument to be processed in the
 18669 shell variable *OPTIND*. Whenever the shell is invoked, *OPTIND* shall be initialized to 1.

18670 When the option requires an option-argument, the *getopts* utility shall place it in the shell
 18671 variable *OPTARG*. If no option was found, or if the option that was found does not have an
 18672 option-argument, *OPTARG* shall be unset.

18673 If an option character not contained in the *optstring* operand is found where an option character
 18674 is expected, the shell variable specified by *name* shall be set to the question-mark ('?') character.
 18675 In this case, if the first character in *optstring* is a colon (':'), the shell variable *OPTARG* shall be
 18676 set to the option character found, but no output shall be written to standard error; otherwise, the
 18677 shell variable *OPTARG* shall be unset and a diagnostic message shall be written to standard
 18678 error. This condition shall be considered to be an error detected in the way arguments were
 18679 presented to the invoking application, but shall be not an error in *getopts* processing.

18680 If an option-argument is missing:

- 18681 • If the first character of *optstring* is a colon, the shell variable specified by *name* shall be set to
 18682 the colon character and the shell variable *OPTARG* shall be set to the option character found.
- 18683 • Otherwise, the shell variable specified by *name* shall be set to the question-mark character,
 18684 the shell variable *OPTARG* shall be unset, and a diagnostic message shall be written to
 18685 standard error. This condition shall be considered to be an error detected in the way
 18686 arguments were presented to the invoking application, but shall not be an error in *getopts*
 18687 processing; a diagnostic message shall be written as stated, but the exit status shall be zero.

18688 When the end of options is encountered, the *getopts* utility shall exit with a return value greater
 18689 than zero; the shell variable *OPTIND* shall be set to the index of the first non-option-argument,
 18690 where the first "—" argument is considered to be an option-argument if there are no other non-
 18691 option-arguments appearing before it, or the value "\$#+1" if there are no non-option-
 18692 arguments; the *name* variable shall be set to the question-mark character. Any of the following
 18693 shall identify the end of options: the special option "—", finding an argument that does not
 18694 begin with a '-', or encountering an error.

18695 The shell variables *OPTIND* and *OPTARG* shall be local to the caller of *getopts* and shall not be
 18696 exported by default.

18697 The shell variable specified by the *name* operand, *OPTIND* and *OPTARG* shall affect the current
 18698 shell execution environment; see Section 2.13 (on page 2273).

18699 If the application sets *OPTIND* to the value 1, a new set of parameters can be used: either the
 18700 current positional parameters or new *arg* values. Any other attempt to invoke *getopts* multiple
 18701 times in a single shell execution environment with parameters (positional parameters or *arg*
 18702 operands) that are not the same in all invocations, or with an *OPTIND* value modified to be a
 18703 value other than 1, produces unspecified results.

18704 **OPTIONS**

18705 None.

18706 **OPERANDS**

18707 The following operands shall be supported:

18708 *optstring* A string containing the option characters recognized by the utility invoking *getopts*.
 18709 If a character is followed by a colon, the option shall be expected to have an
 18710 argument, which should be supplied as a separate argument. Applications should
 18711 specify an option character and its option-argument as separate arguments, but
 18712 *getopts* shall interpret the characters following an option character requiring
 18713 arguments as an argument whether or not this is done. An explicit null option-
 18714 argument need not be recognized if it is not supplied as a separate argument when
 18715 *getopts* is invoked. (See also the *getopt()* function defined in the System Interfaces
 18716 volume of IEEE Std. 1003.1-200x.) The characters question-mark and colon shall
 18717 not be used as option characters by an application. The use of other option
 18718 characters that are not alphanumeric produces unspecified results. If the option-
 18719 argument is not supplied as a separate argument from the option character, the
 18720 value in *OPTARG* shall be stripped of the option character and the '-'. The first
 18721 character in *optstring* determines how *getopts* behaves if an option character is not
 18722 known or an option-argument is missing.

18723 *name* The name of a shell variable that shall be set by the *getopts* utility to the option
 18724 character that was found.

18725 The *getopts* utility by default shall parse positional parameters passed to the invoking shell
 18726 procedure. If *args* are given, they shall be parsed instead of the positional parameters.

18727 **STDIN**

18728 Not used.

18729 **INPUT FILES**

18730 None.

18731 **ENVIRONMENT VARIABLES**18732 The following environment variables shall affect the execution of *getopts*:

18733 *LANG* Provide a default value for the internationalization variables that are unset or null.
 18734 If *LANG* is unset or null, the corresponding value from the implementation-
 18735 defined default locale shall be used. If any of the internationalization variables
 18736 contains an invalid setting, the utility shall behave as if none of the variables had
 18737 been defined.

18738 *LC_ALL* If set to a non-empty string value, override the values of all the other
 18739 internationalization variables.

18740 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 18741 characters (for example, single-byte as opposed to multi-byte characters in
 18742 arguments and input files).

18743 *LC_MESSAGES*

18744 Determine the locale that should be used to affect the format and contents of
 18745 diagnostic messages written to standard error.

18746 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

18747 *OPTIND* This variable shall be used by the *getopts* utility as the index of the next argument
 18748 to be processed.

18749 **ASYNCHRONOUS EVENTS**

18750 Default.

18751 **STDOUT**

18752 Not used.

18753 **STDERR**

18754 Whenever an error is detected and the first character in the *optstring* operand is not a colon
 18755 (' : '), a diagnostic message shall be written to standard error with the following information in
 18756 an unspecified format:

18757 • The invoking program name shall be identified in the message. The invoking program name
 18758 shall be the value of the shell special parameter 0 (see Section 2.5.2 (on page 2241)) at the time
 18759 the *getopts* utility is invoked. A name equivalent to:

18760 basename "\$0"

18761 may be used.

18762 • If an option is found that was not specified in *optstring*, this error is identified and the invalid
 18763 option character shall be identified in the message.

18764 • If an option requiring an option-argument is found, but an option-argument is not found,
 18765 this error shall be identified and the invalid option character shall be identified in the
 18766 message.

18767 **OUTPUT FILES**

18768 None.

18769 **EXTENDED DESCRIPTION**

18770 None.

18771 **EXIT STATUS**

18772 The following exit values shall be returned:

18773 0 An option, specified or unspecified by *optstring*, was found.

18774 >0 The end of options was encountered or an error occurred.

18775 **CONSEQUENCES OF ERRORS**

18776 Default.

18777 **APPLICATION USAGE**

18778 Since *getopts* affects the current shell execution environment, it is generally provided as a shell
 18779 regular built-in. If it is called in a subshell or separate utility execution environment, such as one
 18780 of the following:

18781 (getopts abc value "\$@")

18782 nohup getopts ...

18783 find . -exec getopts ... \;

18784 it does not affect the shell variables in the caller's environment.

18785 Note that shell functions share *OPTIND* with the calling shell even though the positional
 18786 parameters are changed. If the calling shell and any of its functions uses *getopts* to parse
 18787 arguments, the results are unspecified.

18788 **EXAMPLES**

18789 The following example script parses and displays its arguments:

18790 aflag=

18791 bflag=


```

18792     while getopts ab: name
18793     do
18794         case $name in
18795             a)     aflag=1;;
18796             b)     bflag=1
18797                   bval="$OPTARG";;
18798             ?)     printf "Usage: %s: [-a] [-b value] args\n" $0
18799                   exit 2;;
18800         esac
18801     done
18802     if [ ! -z "$aflag" ]; then
18803         printf "Option -a specified\n"
18804     fi
18805     if [ ! -z "$bflag" ]; then
18806         printf 'Option -b "%s" specified\n' "$bval"
18807     fi
18808     shift $(( $OPTIND - 1 ))
18809     printf "Remaining arguments are: %s\n" "$*"

```

18810 RATIONALE

18811 The *getopts* utility was chosen in preference to the System V *getopt* utility because *getopts* handles
 18812 option-arguments containing <blank> characters.

18813 The *OPTARG* variable is not mentioned in the ENVIRONMENT VARIABLES section because it
 18814 does not affect the execution of *getopts*; it is one of the few “output-only” variables used by the
 18815 standard utilities.

18816 The colon is not allowed as an option character because that is not historical behavior, and it
 18817 violates the Utility Syntax Guidelines. The colon is now specified to behave as in the KornShell
 18818 version of the *getopts* utility; when used as the first character in the *optstring* operand, it disables
 18819 diagnostics concerning missing option-arguments and unexpected option characters. This
 18820 replaces the use of the *OPTERR* variable that was specified in an early proposal.

18821 The formats of the diagnostic messages produced by the *getopts* utility and the *getopt()* function
 18822 are not fully specified because implementations with superior (“friendlier”) formats objected to
 18823 the formats used by some historical implementations. The standard developers considered it
 18824 important that the information in the messages used be uniform between *getopts* and *getopt()*.
 18825 Exact duplication of the messages might not be possible, particularly if a utility is built on
 18826 another system that has a different *getopt()* function, but the messages must have specific
 18827 information included so that the program name, invalid option character, and type of error can
 18828 be distinguished by a user.

18829 Only a rare application program intercepts a *getopts* standard error message and wants to parse
 18830 it. Therefore, implementations are free to choose the most usable messages they can devise. The
 18831 following formats are used by many historical implementations:

18832 "%s: illegal option -- %c\n", <program name>, <option character>

18833 "%s: option requires an argument -- %c\n", <program name>, \
 18834 <option character>

18835 Historical shells with built-in versions of *getopt()* or *getopts* have used different formats,
 18836 frequently not even indicating the option character found in error.

18837 **FUTURE DIRECTIONS**

18838 None.

18839 **SEE ALSO**18840 The System Interfaces volume of IEEE Std. 1003.1-200x, *getopt()*18841 **CHANGE HISTORY**

18842 First released in Issue 4.

18843 **Issue 6**

18844 The normative text is reworded to avoid use of the term “must” for application requirements.

18845 **NAME**18846 `grep` — search a file for a pattern18847 **SYNOPSIS**18848 `grep [-E | -F][-c | -l | -q][-insvx] -e pattern_list...`
18849 `[-f pattern_file]...[file...]`18850 `grep [-E | -F][-c | -l | -q][-insvx][-e pattern_list]...`
18851 `-f pattern_file...[file...]`18852 `grep [-E | -F][-c | -l | -q][-insvx] pattern_list[file...]`18853 **DESCRIPTION**

18854 The *grep* utility shall search the input files, selecting lines matching one or more patterns; the
 18855 types of patterns are controlled by the options specified. The patterns are specified by the `-e`
 18856 option, `-f` option, or the *pattern_list* operand. The *pattern_list*'s value shall consist of one or more
 18857 patterns separated by <newline> characters; the *pattern_file*'s contents shall consist of one or
 18858 more patterns terminated by <newline> characters. By default, an input line shall be selected if
 18859 any pattern, treated as an entire basic regular expression (BRE) as described in the Base
 18860 Definitions volume of IEEE Std. 1003.1-200x, Section 9.3, Basic Regular Expressions, matches any
 18861 part of the line; a null BRE shall match every line. By default, each selected input line shall be
 18862 written to the standard output.

18863 Regular expression matching shall be based on text lines. Since a <newline> character separates
 18864 or terminates patterns (see the `-e` and `-f` options below), regular expressions cannot contain a
 18865 <newline> character. Similarly, since patterns are matched against individual lines of the input,
 18866 there is no way for a pattern to match a <newline> character found in the input.

18867 **OPTIONS**

18868 The *grep* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 18869 12.2, Utility Syntax Guidelines.

18870 The following options shall be supported:

18871 **-E** Match using extended regular expressions. Treat each pattern specified as an ERE,
 18872 as described in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 9.4,
 18873 Extended Regular Expressions. If any entire ERE pattern matches some part of an
 18874 input line, the line shall be matched. A null ERE shall match every line.

18875 **-F** Match using fixed strings. Treat each pattern specified as a string instead of a
 18876 regular expression. If an input line contains any of the patterns as a contiguous
 18877 sequence of bytes, the line shall be matched. A null string shall match every line.

18878 **-c** Write only a count of selected lines to standard output.

18879 **-e *pattern_list***

18880 Specify one or more patterns to be used during the search for input. The
 18881 application shall ensure that patterns in *pattern_list* are separated by a <newline>
 18882 character. A null pattern can be specified by two adjacent <newline> characters in
 18883 *pattern_list*. Unless the `-E` or `-F` option is also specified, each pattern shall be
 18884 treated as a BRE, as described in the Base Definitions volume of
 18885 IEEE Std. 1003.1-200x, Section 9.3, Basic Regular Expressions. Multiple `-e` and `-f`
 18886 options shall be accepted by the *grep* utility. All of the specified patterns shall be
 18887 used when matching lines, but the order of evaluation is unspecified.

18888 **-f *pattern_file***

18889 Read one or more patterns from the file named by the path name *pattern_file*.
 18890 Patterns in *pattern_file* shall be terminated by a <newline> character. A null pattern

18891 can be specified by an empty line in *pattern_file*. Unless the **-E** or **-F** option is also
 18892 specified, each pattern shall be treated as a BRE, as described in the Base
 18893 Definitions volume of IEEE Std. 1003.1-200x, Section 9.3, Basic Regular
 18894 Expressions.

18895 **-i** Perform pattern matching in searches without regard to case; see the Base
 18896 Definitions volume of IEEE Std. 1003.1-200x, Section 9.2, Regular Expression
 18897 General Requirements.

18898 **-l** (The letter ell.) Write only the names of files containing selected lines to standard
 18899 output. Path names shall be written once per file searched. If the standard input is
 18900 searched, a path name of "(standard input)" shall be written, in the POSIX
 18901 locale. In other locales, "standard input" may be replaced by something more
 18902 appropriate in those locales.

18903 **-n** Precede each output line by its relative line number in the file, each file starting at
 18904 line 1. The line number counter shall be reset for each file processed.

18905 **-q** Quiet. Do not write anything to the standard output, regardless of matching lines.
 18906 Exit with zero status if an input line is selected.

18907 **-s** Suppress the error messages ordinarily written for nonexistent or unreadable files.
 18908 Other error messages shall not be suppressed.

18909 **-v** Select lines not matching any of the specified patterns. If the **-v** option is not
 18910 specified, selected lines shall be those that match any of the specified patterns.

18911 **-x** Consider only input lines that use all characters in the line to match an entire fixed
 18912 string or regular expression to be matching lines.

18913 OPERANDS

18914 The following operands shall be supported:

18915 *pattern_list* Specify one or more patterns to be used during the search for input. This operand
 18916 shall be treated as if it were specified as **-e pattern_list**.

18917 *file* A path name of a file to be searched for the patterns. If no *file* operands are
 18918 specified, the standard input shall be used.

18919 STDIN

18920 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
 18921 section.

18922 INPUT FILES

18923 The input files shall be text files.

18924 ENVIRONMENT VARIABLES

18925 The following environment variables shall affect the execution of *grep*:

18926 *LANG* Provide a default value for the internationalization variables that are unset or null.
 18927 If *LANG* is unset or null, the corresponding value from the implementation-
 18928 defined default locale shall be used. If any of the internationalization variables
 18929 contains an invalid setting, the utility shall behave as if none of the variables had
 18930 been defined.

18931 *LC_ALL* If set to a non-empty string value, override the values of all the other
 18932 internationalization variables.

18933 *LC_COLLATE*

18934 Determine the locale for the behavior of ranges, equivalence classes and multi-

- 18935 character collating elements within regular expressions.
- 18936 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
18937 characters (for example, single-byte as opposed to multi-byte characters in
18938 arguments and input files) and the behavior of character classes within regular
18939 expressions.
- 18940 **LC_MESSAGES**
18941 Determine the locale that should be used to affect the format and contents of
18942 diagnostic messages written to standard error.
- 18943 **XSI NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 18944 **ASYNCHRONOUS EVENTS**
18945 Default.
- 18946 **STDOUT**
18947 If the **-l** option is in effect, and the **-q** option is not, the following shall be written for each file
18948 containing at least one selected input line:
18949 `"%s\n", <file>`
- 18950 Otherwise, if more than one *file* argument appears, and **-q** is not specified, the *grep* utility shall
18951 prefix each output line by:
18952 `"%s:", <file>`
- 18953 The remainder of each output line shall depend on the other options specified:
- 18954 • If the **-c** option is in effect, the remainder of each output line shall contain:
18955 `"%d\n", <count>`
 - 18956 • Otherwise, if **-c** is not in effect and the **-n** option is in effect, the following shall be written to
18957 standard output:
18958 `"%d:", <line number>`
 - 18959 • Finally, the following shall be written to standard output:
18960 `"%s", <selected-line contents>`
- 18961 **STDERR**
18962 Used only for diagnostic messages.
- 18963 **OUTPUT FILES**
18964 None.
- 18965 **EXTENDED DESCRIPTION**
18966 None.
- 18967 **EXIT STATUS**
18968 The following exit values shall be returned:
18969 **0** One or more lines were selected.
18970 **1** No lines were selected.
18971 **>1** An error occurred.

18972 CONSEQUENCES OF ERRORS

18973 If the `-q` option is specified, the exit status shall be zero if an input line is selected, even if an
 18974 error was detected. Otherwise, default actions shall be performed.

18975 APPLICATION USAGE

18976 Care should be taken when using characters in *pattern_list* that may also be meaningful to the
 18977 command interpreter. It is safest to enclose the entire *pattern_list* argument in single quotes:

18978 '...'

18979 The `-e pattern_list` option has the same effect as the *pattern_list* operand, but is useful when
 18980 *pattern_list* begins with the hyphen delimiter. It is also useful when it is more convenient to
 18981 provide multiple patterns as separate arguments.

18982 Multiple `-e` and `-f` options are accepted and *grep* uses all of the patterns it is given while
 18983 matching input text lines. (Note that the order of evaluation is not specified. If an
 18984 implementation finds a null string as a pattern, it is allowed to use that pattern first, matching
 18985 every line, and effectively ignore any other patterns.)

18986 The `-q` option provides a means of easily determining whether or not a pattern (or string) exists
 18987 in a group of files. When searching several files, it provides a performance improvement
 18988 (because it can quit as soon as it finds the first match) and requires less care by the user in
 18989 choosing the set of files to supply as arguments (because it exits zero if it finds a match even if
 18990 *grep* detected an access or read error on earlier *file* operands).

18991 EXAMPLES

18992 1. To find all uses of the word "Posix" (in any case) in file **text.mm** and write with line
 18993 numbers:

18994 `grep -i -n posix text.mm`

18995 2. To find all empty lines in the standard input:

18996 `grep ^$`

18997 or:

18998 `grep -v .`

18999 3. Both of the following commands print all lines containing strings "abc" or "def" or both:

19000 `grep -E 'abc`

19001 `def'`

19002 `grep -F 'abc`

19003 `def'`

19004 4. Both of the following commands print all lines matching exactly "abc" or "def":

19005 `grep -E '^abc$`

19006 `^def$'`

19007 `grep -F -x 'abc`

19008 `def'`

19009 RATIONALE

19010 This *grep* has been enhanced in an upward-compatible way to provide the exact functionality of
 19011 the historical *egrep* and *fgrep* commands as well. It was the clear intention of the standard
 19012 developers to consolidate the three *greps* into a single command.

- 19013 The old *egrep* and *fgrep* commands are likely to be supported for many years to come as
19014 implementation extensions, allowing historical applications to operate unmodified.
- 19015 Historical implementations usually silently ignored all but one of multiply-specified *-e* and *-f*
19016 options, but were not consistent as to which specification was actually used.
- 19017 The *-b* option was omitted from the `OPTIONS` section because block numbers are
19018 implementation-defined.
- 19019 The System V restriction on using *-* to mean standard input was omitted.
- 19020 A definition of action taken when given a null BRE or ERE is specified. This is an error condition
19021 in some historical implementations.
- 19022 The *-I* option previously indicated that its use was undefined when no files were explicitly
19023 named. This behavior was historical and placed an unnecessary restriction on future
19024 implementations. It has been removed.
- 19025 The historical BSD *grep -s* option practice is easily duplicated by redirecting standard output to
19026 `/dev/null`. The *-s* option required here is from System V.
- 19027 The *-x* option, historically available only with *fgrep*, is available here for all of the non-
19028 obsolescent versions.
- 19029 **FUTURE DIRECTIONS**
- 19030 None.
- 19031 **SEE ALSO**
- 19032 *sed*
- 19033 **CHANGE HISTORY**
- 19034 First released in Issue 2.
- 19035 **Issue 4**
- 19036 Aligned with the ISO/IEC 9945-2:1993 standard.
- 19037 **Issue 6**
- 19038 The Open Group corrigenda item U029/5 has been applied, correcting the SYNOPSIS.
- 19039 The normative text is reworded to avoid use of the term “must” for application requirements.

19040 **NAME**

19041 hash — remember or report utility locations

19042 **SYNOPSIS**19043 xSI hash [*utility...*]

19044 hash -r

19045

19046 **DESCRIPTION**

19047 The *hash* utility shall affect the way the current shell environment remembers the locations of
 19048 utilities found as described in Section 2.9.1.1 (on page 2257). Depending on the arguments
 19049 specified, it shall add utility locations to its list of remembered locations or it shall purge the
 19050 contents of the list. When no arguments are specified, it shall report on the contents of the list.

19051 Utilities provided as built-ins to the shell shall not be reported by *hash*.19052 **OPTIONS**

19053 The *hash* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 19054 12.2, Utility Syntax Guidelines.

19055 The following option shall be supported:

19056 -r Forget all previously remembered utility locations.

19057 **OPERANDS**

19058 The following operand shall be supported:

19059 *utility* The name of a utility to be searched for and added to the list of remembered
 19060 locations. If *utility* contains one or more slashes, the results are unspecified.

19061 **STDIN**

19062 Not used.

19063 **INPUT FILES**

19064 None.

19065 **ENVIRONMENT VARIABLES**19066 The following environment variables shall affect the execution of *hash*:

19067 *LANG* Provide a default value for the internationalization variables that are unset or null.
 19068 If *LANG* is unset or null, the corresponding value from the implementation-
 19069 defined default locale shall be used. If any of the internationalization variables
 19070 contains an invalid setting, the utility shall behave as if none of the variables had
 19071 been defined.

19072 *LC_ALL* If set to a non-empty string value, override the values of all the other
 19073 internationalization variables.

19074 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 19075 characters (for example, single-byte as opposed to multi-byte characters in
 19076 arguments).

19077 *LC_MESSAGES*

19078 Determine the locale that should be used to affect the format and contents of
 19079 diagnostic messages written to standard error.

19080 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

19081 *PATH* Determine the location of *utility*, as described in the Base Definitions volume of
 19082 IEEE Std. 1003.1-200x, Chapter 8, Environment Variables.

19083 **ASYNCHRONOUS EVENTS**

19084 Default.

19085 **STDOUT**

19086 The standard output of *hash* shall be used when no arguments are specified. Its format is
19087 unspecified, but includes the path name of each utility in the list of remembered locations for the
19088 current shell environment. This list shall consist of those utilities named in previous *hash*
19089 invocations that have been invoked, and may contain those invoked and found through the
19090 normal command search process.

19091 **STDERR**

19092 Used only for diagnostic messages.

19093 **OUTPUT FILES**

19094 None.

19095 **EXTENDED DESCRIPTION**

19096 None.

19097 **EXIT STATUS**

19098 The following exit values shall be returned:

19099 0 Successful completion.

19100 >0 An error occurred.

19101 **CONSEQUENCES OF ERRORS**

19102 Default.

19103 **APPLICATION USAGE**

19104 Since *hash* affects the current shell execution environment, it is always provided as a shell
19105 regular built-in. If it is called in a separate utility execution environment, such as one of the
19106 following:

19107 `nohup hash -r`19108 `find . -type f | xargs hash`

19109 it does not affect the command search process of the caller's environment.

19110 The *hash* utility may be implemented as an alias—for example, *alias -t -*, in which case utilities
19111 found through normal command search are not listed by the *hash* command.

19112 The effects of *hash -r* can also be achieved portably by resetting the value of *PATH*; in the
19113 simplest form, this can be:

19114 `PATH="$PATH"`

19115 The use of *hash* with *utility* names is unnecessary for most applications, but may provide a
19116 performance improvement on a few implementations; normally, the hashing process is included
19117 by default.

19118 **EXAMPLES**

19119 None.

19120 **RATIONALE**

19121 None.

19122 **FUTURE DIRECTIONS**

19123 None.

19124 **SEE ALSO**

19125 Section 2.9.1.1 (on page 2257)

19126 **CHANGE HISTORY**

19127 First released in Issue 2.

19128 **Issue 4**

19129 Relocated from the *sh* description to reflect its status as a regular built-in utility.

19130 **NAME**

19131 head — copy the first part of files

19132 **SYNOPSIS**19133 head [-n *number*][*file...*]19134 **DESCRIPTION**19135 The *head* utility shall copy its input files to the standard output, ending the output for each file at
19136 a designated point.19137 Copying shall end at the point in each input file indicated by the **-n *number*** option. The option-
19138 argument *number* shall be counted in units of lines.19139 **OPTIONS**19140 The *head* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
19141 12.2, Utility Syntax Guidelines.

19142 The following option shall be supported:

19143 **-n *number*** The first *number* lines of each input file shall be copied to standard output. The
19144 application shall ensure that the *number* option-argument is a positive decimal
19145 integer.19146 If no options are specified, *head* shall act as if **-n 10** had been specified.19147 **OPERANDS**

19148 The following operand shall be supported:

19149 *file* A path name of an input file. If no *file* operands are specified, the standard input
19150 shall be used.19151 **STDIN**19152 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
19153 section.19154 **INPUT FILES**

19155 Input files shall be text files, but the line length is not restricted to {LINE_MAX} bytes.

19156 **ENVIRONMENT VARIABLES**19157 The following environment variables shall affect the execution of *head*:19158 *LANG* Provide a default value for the internationalization variables that are unset or null.
19159 If *LANG* is unset or null, the corresponding value from the implementation-
19160 defined default locale shall be used. If any of the internationalization variables
19161 contains an invalid setting, the utility shall behave as if none of the variables had
19162 been defined.19163 *LC_ALL* If set to a non-empty string value, override the values of all the other
19164 internationalization variables.19165 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
19166 characters (for example, single-byte as opposed to multi-byte characters in
19167 arguments and input files).19168 *LC_MESSAGES*19169 Determine the locale that should be used to affect the format and contents of
19170 diagnostic messages written to standard error.19171 *XSI* *NLS_PATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

19172 **ASYNCHRONOUS EVENTS**

19173 Default.

19174 **STDOUT**

19175 The standard output shall contain designated portions of the input files.

19176 If multiple *file* operands are specified, *head* shall precede the output for each with the header:

19177 "\n==> %s <==\n", <pathname>

19178 except that the first header written shall not include the initial <newline> character.

19179 **STDERR**

19180 Used only for diagnostic messages.

19181 **OUTPUT FILES**

19182 None.

19183 **EXTENDED DESCRIPTION**

19184 None.

19185 **EXIT STATUS**

19186 The following exit values shall be returned:

19187 0 Successful completion.

19188 >0 An error occurred.

19189 **CONSEQUENCES OF ERRORS**

19190 Default.

19191 **APPLICATION USAGE**19192 The obsolescent *-number* form is withdrawn in this version. Applications should use the *-n*
19193 *number* option.19194 **EXAMPLES**

19195 To write the first ten lines of all files (except those with a leading period) in the directory:

19196 head *

19197 **RATIONALE**19198 Although it is possible to simulate *head* with *sed 10q* for a single file, the standard developers
19199 decided that the popularity of *head* on historical BSD systems warranted its inclusion alongside
19200 *tail*.19201 This standard version of *head* follows the Utility Syntax Guidelines. The *-n* option was added to
19202 this new interface so that *head* and *tail* would be more logically related.19203 There is no *-c* option (as there is in *tail*) because it is not historical practice and because other
19204 utilities in this volume of IEEE Std. 1003.1-200x provide similar functionality.19205 **FUTURE DIRECTIONS**

19206 None.

19207 **SEE ALSO**19208 *sed*, *tail*19209 **CHANGE HISTORY**

19210 First released in Issue 4.

19211 **Issue 6**

19212 The obsolescent **–number** form is withdrawn.

19213 The normative text is reworded to avoid use of the term “must” for application requirements.

19214 NAME

19215 iconv — codeset conversion

19216 SYNOPSIS

19217 iconv [-cs] -f *fromcode* -t *toctype* [*file* ...]

19218 iconv -l

19219 DESCRIPTION

19220 The *iconv* utility shall convert the encoding of characters in *file* from one codeset to another and
 19221 write the results to standard output.

19222 When the options indicate that charmap files are used to specify the codesets (see OPTIONS),
 19223 the codeset conversion shall be accomplished by performing a logical join on the symbolic
 19224 character names in the two charmaps. The implementation need not support the use of charmap
 19225 files for codeset conversion unless the POSIX2_LOCALEDEF symbol is defined on the system.

19226 OPTIONS

19227 The *iconv* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 19228 12.2, Utility Syntax Guidelines.

19229 The following options shall be supported:

19230 **-c** Omit any invalid characters from the output. When **-c** is not used, the results of
 19231 encountering invalid characters in the input stream (either those that are not valid
 19232 members of the *fromcode* or those that have no corresponding value in *toctype*) shall
 19233 be specified in the system documentation. The presence or absence of **-c** shall not
 19234 affect the exit status of *iconv*.

19235 **-f *fromcode*** Identify the codeset of the input file. If the option-argument contains a slash
 19236 character, *iconv* shall attempt to use it as the path name of a charmap file, as
 19237 defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 6.4,
 19238 Character Set Description File. If the path name does not represent a valid,
 19239 readable charmap file, the results are undefined. If the option-argument does not
 19240 contain a slash, it shall be considered the name of one of the codeset descriptions
 19241 provided by the system, in an unspecified format. The valid values of the option-
 19242 argument without a slash are implementation-defined. If this option is omitted, the
 19243 codeset of the current locale shall be used.

19244 **-l** Write all supported *fromcode* and *toctype* values to standard output in an unspecified
 19245 format.

19246 **-s** Suppress any messages written to standard error concerning invalid characters.
 19247 When **-s** is not used, the results of encountering invalid characters in the input
 19248 stream (either those that are not valid members of the *fromcode* or those that have
 19249 no corresponding value in *toctype*) shall be specified in the system documentation.
 19250 The presence or absence of **-s** shall not affect the exit status of *iconv*.

19251 **-t *toctype*** Identify the codeset to be used for the output file. The semantics are equivalent to
 19252 the **-f *fromcode*** option.

19253 If either **-f** or **-t** represents a charmap file, but the other does not (or is omitted), or both **-f** and
 19254 **-t** are omitted, the results are undefined.

19255 OPERANDS

19256 The following operand shall be supported:

19257 ***file*** A path name of an input file. If no *file* operands are specified, or if a *file* operand is
 19258 '-', the standard input shall be used.

19259 **STDIN**

19260 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'. |

19261 **INPUT FILES**

19262 The input file shall be a text file.

19263 **ENVIRONMENT VARIABLES**

19264 The following environment variables shall affect the execution of *iconv*:

19265 *LANG* Provide a default value for the internationalization variables that are unset or null.
 19266 If *LANG* is unset or null, the corresponding value from the implementation-
 19267 defined default locale shall be used. If any of the internationalization variables
 19268 contains an invalid setting, the utility shall behave as if none of the variables had
 19269 been defined.

19270 *LC_ALL* If set to a non-empty string value, override the values of all the other
 19271 internationalization variables.

19272 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 19273 characters (for example, single-byte as opposed to multi-byte characters in
 19274 arguments). During translation of the file, this variable is superseded by the use of
 19275 the *fromcode* option-argument.

19276 *LC_MESSAGES*

19277 Determine the locale that should be used to affect the format and contents of
 19278 diagnostic messages written to standard error.

19279 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

19280 **ASYNCHRONOUS EVENTS**

19281 Default.

19282 **STDOUT**

19283 When the *-I* option is used, the standard output shall contain all supported *fromcode* and *to*
 19284 *code* values, written in an unspecified format.

19285 When the *-I* option is not used, the standard output shall contain the sequence of characters
 19286 read from the input files, translated to the specified codeset. Nothing else shall be written to the
 19287 standard output.

19288 **STDERR**

19289 Used only for diagnostic messages.

19290 **OUTPUT FILES**

19291 None.

19292 **EXTENDED DESCRIPTION**

19293 None.

19294 **EXIT STATUS**

19295 The following exit values shall be returned:

19296 0 Successful completion.

19297 >0 An error occurred.

19298 **CONSEQUENCES OF ERRORS**

19299 Default.

19300 **APPLICATION USAGE**

19301 The user must ensure that both charmap files use the same symbolic names for characters the
19302 two codesets have in common.

19303 **EXAMPLES**

19304 The following example converts the contents of file **mail.x400** from the ISO/IEC 6937:1994
19305 standard codeset to the ISO/IEC 8859-1:1998 standard codeset, and stores the results in file
19306 **mail.local**:

```
19307           iconv -f IS6937 -t IS8859 mail.x400 > mail.local
```

19308 **RATIONALE**

19309 The *iconv* utility can be used portably only when the user provides two charmap files as option-
19310 arguments. This is because a single charmap provided by the user cannot reliably be joined with
19311 the names in a system-provided character set description. The valid values for *fromcode* and
19312 *tocode* are implementation-defined and do not have to have any relation to the charmap
19313 mechanisms. As an aid to interactive users, the **-I** option was adopted from the Plan 9 operating
19314 system. It writes information concerning these implementation-defined values. The format is
19315 unspecified because there are many possible useful formats that could be chosen, such as a
19316 matrix of valid combinations of *fromcode* and *tocode*. The **-I** option is not intended for shell script
19317 usage; portable applications will have to use charmaps.

19318 **FUTURE DIRECTIONS**

19319 None.

19320 **SEE ALSO**

19321 *gencat*

19322 **CHANGE HISTORY**

19323 First released in Issue 3.

19324 **Issue 4**

19325 Format reorganized.

19326 Utility Syntax Guidelines support mandated.

19327 Internationalized environment variable support mandated.

19328 **Issue 6**

19329 This utility has been rewritten to align with the IEEE P1003.2b draft standard. Specifically, the
19330 ability to use charmap files for conversion has been added.

19331 **NAME**19332 `id` — return user identity19333 **SYNOPSIS**19334 `id` [*user*]19335 `id -G[-n]` [*user*]19336 `id -g[-nr]` [*user*]19337 `id -u[-nr]` [*user*]19338 **DESCRIPTION**

19339 If no *user* operand is provided, the *id* utility shall write the user and group IDs and the
 19340 corresponding user and group names of the invoking process to standard output. If the effective
 19341 and real IDs do not match, both shall be written. If multiple groups are supported by the
 19342 underlying system (see the description of {NGROUPS_MAX} in the System Interfaces volume of
 19343 IEEE Std. 1003.1-200x), the supplementary group affiliations of the invoking process shall also be
 19344 written.

19345 If a *user* operand is provided and the process has the appropriate privileges, the user and group
 19346 IDs of the selected user shall be written. In this case, effective IDs shall be assumed to be
 19347 identical to real IDs. If the selected user has more than one allowable group membership listed
 19348 in the group database, these shall be written in the same manner as the supplementary groups
 19349 described in the preceding paragraph.

19350 **OPTIONS**

19351 The *id* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
 19352 Utility Syntax Guidelines.

19353 The following options shall be supported:

19354 **-G** Output all different group IDs (effective, real, and supplementary) only, using the
 19355 format "`%u\n`". If there is more than one distinct group affiliation, output each
 19356 such affiliation, using the format "`%u`", before the <newline> character is output.

19357 **-g** Output only the effective group ID, using the format "`%u\n`".

19358 **-n** Output the name in the format `%s` instead of the numeric ID using the format `%u`.

19359 **-r** Output the real ID instead of the effective ID.

19360 **-u** Output only the effective user ID, using the format "`%u\n`".

19361 **OPERANDS**

19362 The following operand shall be supported:

19363 *user* The login name for which information is to be written.

19364 **STDIN**

19365 Not used.

19366 **INPUT FILES**

19367 None.

19368 **ENVIRONMENT VARIABLES**

19369 The following environment variables shall affect the execution of *id*:

19370 *LANG* Provide a default value for the internationalization variables that are unset or null.
 19371 If *LANG* is unset or null, the corresponding value from the implementation-
 19372 defined default locale shall be used. If any of the internationalization variables
 19373 contains an invalid setting, the utility shall behave as if none of the variables had

- 19374 been defined.
- 19375 *LC_ALL* If set to a non-empty string value, override the values of all the other
19376 internationalization variables.
- 19377 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
19378 characters (for example, single-byte as opposed to multi-byte characters in
19379 arguments).
- 19380 *LC_MESSAGES*
19381 Determine the locale that should be used to affect the format and contents of
19382 diagnostic messages written to standard error and informative messages written to
19383 standard output.
- 19384 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 19385 **ASYNCHRONOUS EVENTS**
19386 Default.
- 19387 **STDOUT**
19388 The following formats shall be used when the *LC_MESSAGES* locale category specifies the
19389 POSIX locale. In other locales, the strings *uid*, *gid*, *eid*, *egid*, and *groups* may be replaced with
19390 more appropriate strings corresponding to the locale.
- 19391 "uid=%u(%s) gid=%u(%s)\n", <real user ID>, <user-name>,
19392 <real group ID>, <group-name>
- 19393 If the effective and real user IDs do not match, the following shall be inserted immediately
19394 before the '\n' character in the previous format:
- 19395 " eid=%u(%s) "
- 19396 with the following arguments added at the end of the argument list:
- 19397 "effective user ID", <effective user-name>
- 19398 If the effective and real group IDs do not match, the following shall be inserted directly before
19399 the '\n' character in the format string (and after any addition resulting from the effective and
19400 real user IDs not matching):
- 19401 " egid=%u(%s) "
- 19402 with the following arguments added at the end of the argument list:
- 19403 <effective group-ID>, <effective group name>
- 19404 If the process has supplementary group affiliations or the selected user is allowed to belong to
19405 multiple groups, the first shall be added directly before the <newline> character in the format
19406 string:
- 19407 " groups=%u(%s) "
- 19408 with the following arguments added at the end of the argument list:
- 19409 <supplementary group ID>, <supplementary group name>
- 19410 and the necessary number of the following added after that for any remaining supplementary
19411 group IDs:
- 19412 " ,%u(%s) "
- 19413 and the necessary number of the following arguments added at the end of the argument list:

19414 <supplementary group ID>, <supplementary group name>

19415 If any of the user ID, group ID, effective user ID, effective group ID, or supplementary/multiple
 19416 group IDs cannot be mapped by the system into printable user or group names, the
 19417 corresponding (%s) and name argument is omitted from the corresponding format string.

19418 When any of the options are specified, the output format shall be as described in the OPTIONS
 19419 section.

19420 **STDERR**

19421 Used only for diagnostic messages.

19422 **OUTPUT FILES**

19423 None.

19424 **EXTENDED DESCRIPTION**

19425 None.

19426 **EXIT STATUS**

19427 The following exit values shall be returned:

19428 0 Successful completion.

19429 >0 An error occurred.

19430 **CONSEQUENCES OF ERRORS**

19431 Default.

19432 **APPLICATION USAGE**

19433 Output produced by the **-G** option and by the default case could potentially produce very long
 19434 lines on systems that support large numbers of supplementary groups. (On systems with user
 19435 and group IDs that are 32-bit integers and with group names with a maximum of 8 bytes per
 19436 name, 93 supplementary groups plus distinct effective and real group and user IDs could
 19437 theoretically overflow the 2 048-byte {LINE_MAX} text file line limit on the default output case.
 19438 It would take about 186 supplementary groups to overflow the 2 048-byte barrier using *id -G*).
 19439 This is not expected to be a problem in practice, but in cases where it is a concern, applications
 19440 should consider using *fold -s* before postprocessing the output of *id*.

19441 **EXAMPLES**

19442 None.

19443 **RATIONALE**

19444 The functionality provided by the 4 BSD *groups* utility can be simulated using:

19445 `id -Gn [user]`

19446 The 4 BSD command *groups* was considered, but it was not included because it did not provide
 19447 the functionality of the *id* utility of the SVID. Also, it was thought that it would be easier to
 19448 modify *id* to provide the additional functionality necessary to systems with multiple groups
 19449 than to invent another command.

19450 The options **-u**, **-g**, **-n**, and **-r** were added to ease the use of *id* with shell commands
 19451 substitution. Without these options it is necessary to use some preprocessor such as *sed* to select
 19452 the desired piece of information. Since output such as that produced by:

19453 `id -u -n`

19454 is frequently wanted, it seemed desirable to add the options.

19455 **FUTURE DIRECTIONS**

19456 None.

19457 **SEE ALSO**19458 *fold, logname, who*, the System Interfaces volume of IEEE Std. 1003.1-200x, *getgid()*, *getgroups()*,
19459 *getuid()*19460 **CHANGE HISTORY**

19461 First released in Issue 2.

19462 **Issue 4**

19463 Aligned with the ISO/IEC 9945-2:1993 standard.

19464 **NAME**

19465 ipcrm — remove an XSI message queue, semaphore set, or shared memory segment identifier

19466 **SYNOPSIS**

```
19467 xsi ipcrm [ -q msgid | -Q msgkey | -s semid | -S semkey |
19468 -m shmid | -M shmkey ] ...
```

19469

19470 **DESCRIPTION**

19471 The *ipcrm* utility shall remove zero or more message queues, semaphore sets, or shared memory
19472 segments. The interprocess communication facilities to be removed are specified by the options.

19473 Only a user with appropriate privilege shall be allowed to remove an interprocess
19474 communication facility that was not created by or owned by the user invoking *ipcrm*.

19475 **OPTIONS**

19476 The *ipcrm* facility supports the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
19477 Utility Syntax Guidelines.

19478 The following options shall be supported:

19479 **-q msgid** Remove the message queue identifier *msgid* from the system and destroy the
19480 message queue and data structure associated with it.

19481 **-m shmid** Remove the shared memory identifier *shmid* from the system. The shared memory
19482 segment and data structure associated with it shall be destroyed after the last
19483 detach.

19484 **-s semid** Remove the semaphore identifier *semid* from the system and destroy the set of
19485 semaphores and data structure associated with it.

19486 **-Q msgkey** Remove the message queue identifier, created with key *msgkey*, from the system
19487 and destroy the message queue and data structure associated with it.

19488 **-M shmkey** Remove the shared memory identifier, created with key *shmkey*, from the system.
19489 The shared memory segment and data structure associated with it shall be
19490 destroyed after the last detach.

19491 **-S semkey** Remove the semaphore identifier, created with key *semkey*, from the system and
19492 destroy the set of semaphores and data structure associated with it.

19493 **OPERANDS**

19494 None.

19495 **STDIN**

19496 Not used.

19497 **INPUT FILES**

19498 None.

19499 **ENVIRONMENT VARIABLES**

19500 The following environment variables shall affect the execution of *ipcrm*:

19501 **LANG** Provide a default value for the internationalization variables that are unset or null.
19502 If *LANG* is unset or null, the corresponding value from the implementation-
19503 defined default locale shall be used. If any of the internationalization variables
19504 contain an invalid setting, the utility behaves as if none of the variables had been
19505 set.

19506 **LC_ALL** If set to a non-empty string value, override the values of all the other
19507 internationalization variables.

- 19508 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
19509 characters (for example, single-byte as opposed to multi-byte characters in
19510 arguments).
- 19511 *LC_MESSAGES*
19512 Determine the locale that should be used to affect the format and contents of
19513 diagnostic messages written to standard error.
- 19514 *NLSPATH*
19515 Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 19516 **ASYNCHRONOUS EVENTS**
- 19517 Default.
- 19518 **STDOUT**
- 19519 Not used.
- 19520 **STDERR**
- 19521 Used only for diagnostic messages.
- 19522 **OUTPUT FILES**
- 19523 None.
- 19524 **EXTENDED DESCRIPTION**
- 19525 None.
- 19526 **EXIT STATUS**
- 19527 The following exit values shall be returned:
- 19528 0 Successful completion.
- 19529 >0 An error occurred.
- 19530 **CONSEQUENCES OF ERRORS**
- 19531 Default.
- 19532 **APPLICATION USAGE**
- 19533 None.
- 19534 **EXAMPLES**
- 19535 None.
- 19536 **RATIONALE**
- 19537 None.
- 19538 **FUTURE DIRECTIONS**
- 19539 None.
- 19540 **SEE ALSO**
- 19541 *ipcs*, the System Interfaces volume of IEEE Std. 1003.1-200x, *msgctl()*, *semctl()*, *shmctl()*
- 19542 **CHANGE HISTORY**
- 19543 First released in Issue 5.

19544 **NAME**19545 `ipcs` — report XSI interprocess communication facilities status19546 **SYNOPSIS**19547 XSI `ipcs [-qms] [-a | -bcopt]`

19548

19549 **DESCRIPTION**19550 The *ipcs* utility shall write information about active interprocess communication facilities.

19551 Without options, information shall be written in short format for message queues, shared
 19552 memory segments, and semaphores sets that are currently active in the system. Otherwise, the
 19553 information that is displayed is controlled by the options specified.

19554 **OPTIONS**

19555 The *ipcs* facility supports the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
 19556 Utility Syntax Guidelines.

19557 The *ipcs* utility accepts the following options:19558 **-q** Write information about active message queues.19559 **-m** Write information about active shared memory segments.19560 **-s** Write information about active semaphores sets.

19561 If **-q**, **-m**, or **-s** are specified, only information about those facilities shall be written. If none of
 19562 these three are specified, information about all three shall be written subject to the following
 19563 options:

19564 **-a** Use all print options. (This is a shorthand notation for **-b**, **-c**, **-o**, **-p**, and **-t**.)

19565 **-b** Write information on maximum allowable size. (Maximum number of bytes in
 19566 messages on queue for message queues, size of segments for shared memory, and
 19567 number of semaphores in each set for semaphores.)

19568 **-c** Write creator's user name and group name; see below.

19569 **-o** Write information on outstanding usage. (Number of messages on queue and total
 19570 number of bytes in messages on queue for message queues, and number of
 19571 processes attached to shared memory segments.)

19572 **-p** Write process number information. (Process ID of last process to send a message
 19573 and process ID of last process to receive a message on message queues, process ID
 19574 of creating process, and process ID of last process to attach or detach on shared
 19575 memory segments.)

19576 **-t** Write time information. (Time of the last control operation that changed the access
 19577 permissions for all facilities, time of last *msgsnd()* and *msgrcv()* operations on
 19578 message queues, time of last *shmat()* and *shmdt()* operations on shared memory,
 19579 and time of last *semop()* operation on semaphores.)

19580 **OPERANDS**

19581 None.

19582 **STDIN**

19583 Not used.

19584 **INPUT FILES**

- 19585 • The group database
- 19586 • The user database

19587 **ENVIRONMENT VARIABLES**

19588 The following environment variables shall affect the execution of *ipcs*:

- 19589 *LANG* Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-defined default locale shall be used. If any of the internationalization variables contain an invalid setting, the utility behaves as if none of the variables had been set.
- 19590
- 19591
- 19592
- 19593
- 19594 *LC_ALL* If set to a non-empty string value, override the values of all the other internationalization variables.
- 19595
- 19596 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
- 19597
- 19598
- 19599 *LC_MESSAGES* Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
- 19600
- 19601
- 19602 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 19603 *TZ* Determine the timezone for the time strings written by *ipcs*.

19604 **ASYNCHRONOUS EVENTS**

19605 Default.

19606 **STDOUT**

19607 An introductory line shall be written with the format:

19608 "IPC status from %s as of %s\n", <source>, <date>

19609 where <source> indicates the source used to gather the statistics and <date> is the information that would be produced by the *date* command when invoked in the POSIX locale.

19610

19611 The *ipcs* utility then shall create up to three reports depending upon the *-q*, *-m*, and *-s* options. The first report shall indicate the status of message queues, the second report shall indicate the status of shared memory segments, and the third report shall indicate the status of semaphore sets.

19612

19613

19614

19615 If the corresponding facility is not installed or has not been used since the last reboot, then the report shall be written out in the format:

19616

19617 "%s facility not in system.\n", <facility>

19618 where <facility> is *Message Queue*, *Shared Memory*, or *Semaphore*, as appropriate. If the facility has been installed and has been used since the last reboot, column headings separated by one or more spaces and followed by a <newline> shall be written as indicated below followed by the facility name written out using the format:

19619

19620

19621

19622 "%s:\n", <facility>

19623 where <facility> is *Message Queues*, *Shared Memory*, or *Semaphores*, as appropriate. On the second and third reports the column headings need not be written if the last column headings written already provide column headings for all information in that report.

19624

19625

19626 The column headings provided in the first column below and the meaning of the information in
 19627 those columns shall be given in order below; the letters in parentheses indicate the options that
 19628 shall cause the corresponding column to appear; “all” means that the column shall always
 19629 appear. Each column is separated by one or more <space> characters. Note that these options
 19630 only determine what information is provided for each report; they do not determine which
 19631 reports are written.

19632	T	(all)	Type of facility:
19633		q	Message queue.
19634		m	Shared memory segment.
19635		s	Semaphore.
19636			This field is a single character written using the format %c.
19637	ID	(all)	The identifier for the facility entry. This field shall be written using the format
19638			%d.
19639	KEY	(all)	The key used as an argument to <i>msgget()</i> , <i>semget()</i> , or <i>shmget()</i> to create the
19640			facility entry.
19641		Note:	The key of a shared memory segment is changed to IPC_PRIVATE
19642			when the segment has been removed until all processes attached
19643			to the segment detach it.
19644			This field shall be written using the format 0x%x.
19645	MODE	(all)	The facility access modes and flags. The mode shall consist of 11 characters
19646			that are interpreted as follows.
19647			The first character shall be:
19648		S	If a process is waiting on a <i>msgsnd()</i> operation.
19649		–	If the above is not true.
19650			The second character shall be:
19651		R	If a process is waiting on a <i>msgrcv()</i> operation.
19652		C or –	If the associated shared memory segment is to be cleared when the
19653			first attach operation is executed.
19654		–	If none of the above is true.
19655			The next nine characters shall be interpreted as three sets of three bits each.
19656			The first set refers to the owner’s permissions; the next to permissions of
19657			others in the usergroup of the facility entry; and the last to all others. Within
19658			each set, the first character indicates permission to read, the second character
19659			indicates permission to write or alter the facility entry, and the last character is
19660			a minus sign (‘-’).
19661			The permissions shall be indicated as follows:
19662		r	If read permission is granted.
19663		w	If write permission is granted.
19664		a	If alter permission is granted.
19665		–	If the indicated permission is not granted.

19666			The first character following the permissions specifies if there is an alternate or additional access control method associated with the facility. If there is no alternate or additional access control method associated with the facility, a single <space> character shall be written; otherwise, another printable character is written.
19667			
19668			
19669			
19670			
19671	OWNER	(all)	The user name of the owner of the facility entry. If the user name of the owner is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the owner shall be written using the format %d.
19672			
19673			
19674			
19675	GROUP	(all)	The group name of the owner of the facility entry. If the group name of the owner is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the owner shall be written using the format %d.
19676			
19677			
19678			
19679			The following nine columns shall be only written out for message queues:
19680	CREATOR	(a,c)	The user name of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the creator shall be written using the format %d.
19681			
19682			
19683			
19684	CGROUP	(a,c)	The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the creator shall be written using the format %d.
19685			
19686			
19687			
19688	CBYTES	(a,o)	The number of bytes in messages currently outstanding on the associated message queue. This field shall be written using the format %d.
19689			
19690	QNUM	(a,o)	The number of messages currently outstanding on the associated message queue. This field shall be written using the format %d.
19691			
19692	QBYTES	(a,b)	The maximum number of bytes allowed in messages outstanding on the associated message queue. This field shall be written using the format %d.
19693			
19694	LSPID	(a,p)	The process ID of the last process to send a message to the associated queue. This field shall be written using the format:
19695			
19696			"%d", <pid>
19697			where <pid> is 0 if no message has been sent to the corresponding message queue; otherwise, <pid> shall be the process ID of the last process to send a message to the queue.
19698			
19699			
19700	LRPID	(a,p)	The process ID of the last process to receive a message from the associated queue. This field shall be written using the format:
19701			
19702			"%d", <pid>
19703			where <pid> is 0 if no message has been received from the corresponding message queue; otherwise, <pid> shall be the process ID of the last process to receive a message from the queue.
19704			
19705			
19706	STIME	(a,t)	The time the last message was sent to the associated queue. If a message has been sent to the corresponding message queue, the hour, minute, and second of the last time a message was sent to the queue shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be written.
19707			
19708			
19709			

19710	RTIME	(a,t)	The time the last message was received from the associated queue. If a message has been received from the corresponding message queue, the hour, minute, and second of the last time a message was received from the queue shall be written using the format <code>%d:%2.2d:%2.2d</code> . Otherwise, the format <code>" no-entry"</code> shall be written.
19711			
19712			
19713			
19714			
19715	The following eight columns shall be only written out for shared memory segments.		
19716	CREATOR	(a,c)	The user of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format <code>%s</code> . Otherwise, the user ID of the creator shall be written using the format <code>%d</code> .
19717			
19718			
19719			
19720	CGROUP	(a,c)	The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format <code>%s</code> . Otherwise, the group ID of the creator shall be written using the format <code>%d</code> .
19721			
19722			
19723			
19724	NATTCH	(a,o)	The number of processes attached to the associated shared memory segment. This field shall be written using the format <code>%d</code> .
19725			
19726	SEGSZ	(a,b)	The size of the associated shared memory segment. This field shall be written using the format <code>%d</code> .
19727			
19728	CPID	(a,p)	The process ID of the creator of the shared memory entry. This field shall be written using the format <code>%d</code> .
19729			
19730	LPID	(a,p)	The process ID of the last process to attach or detach the shared memory segment. This field shall be written using the format:
19731			
19732			<code>"%d", <pid></code>
19733			where <code><pid></code> is 0 if no process has attached the corresponding shared memory segment; otherwise, <code><pid></code> shall be the process ID of the last process to attach or detach the segment.
19734			
19735			
19736	ATIME	(a,t)	The time the last attach on the associated shared memory segment was completed. If the corresponding shared memory segment has ever been attached, the hour, minute, and second of the last time the segment was attached shall be written using the format <code>%d:%2.2d:%2.2d</code> . Otherwise, the format <code>" no-entry"</code> shall be written.
19737			
19738			
19739			
19740			
19741	DTIME	(a,t)	The time the last detach on the associated shared memory segment was completed. If the corresponding shared memory segment has ever been detached, the hour, minute, and second of the last time the segment was detached shall be written using the format <code>%d:%2.2d:%2.2d</code> . Otherwise, the format <code>" no-entry"</code> shall be written.
19742			
19743			
19744			
19745			
19746	The following four columns shall be only written out for semaphore sets:		
19747	CREATOR	(a,c)	The user of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format <code>%s</code> . Otherwise, the user ID of the creator shall be written using the format <code>%d</code> .
19748			
19749			
19750			
19751	CGROUP	(a,c)	The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format <code>%s</code> . Otherwise, the group ID of the creator shall be written using the format <code>%d</code> .
19752			
19753			
19754			

- 19755 NSEMS (a,b) The number of semaphores in the set associated with the semaphore entry.
19756 This field shall be written using the format %d.
- 19757 OTIME (a,t) The time the last semaphore operation on the set associated with the
19758 semaphore entry was completed. If a semaphore operation has ever been
19759 performed on the corresponding semaphore set, the hour, minute, and second
19760 of the last semaphore operation on the semaphore set shall be written using
19761 the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be
19762 written.
- 19763 The following column shall be written for all three reports when it is requested:
- 19764 CTIME (a,t) The time the associated entry was created or changed. The hour, minute, and
19765 second of the time when the associated entry was created shall be written
19766 using the format %d:%2.2d:%2.2d.
- 19767 **STDERR**
19768 Used only for diagnostic messages.
- 19769 **OUTPUT FILES**
19770 None.
- 19771 **EXTENDED DESCRIPTION**
19772 None.
- 19773 **EXIT STATUS**
19774 The following exit values shall be returned:
19775 0 Successful completion.
19776 >0 An error occurred.
- 19777 **CONSEQUENCES OF ERRORS**
19778 Default.
- 19779 **APPLICATION USAGE**
19780 Things can change while *ipcs* is running; the information it gives is guaranteed to be accurate
19781 only when it was retrieved.
- 19782 **EXAMPLES**
19783 None.
- 19784 **RATIONALE**
19785 None.
- 19786 **FUTURE DIRECTIONS**
19787 None.
- 19788 **SEE ALSO**
19789 The System Interfaces volume of IEEE Std. 1003.1-200x, *msgop()*, *msgrcv()*, *msgsnd()*, *semget()*,
19790 *semop()*, *shmat()*, *shmdt()*, *shmget()*, *shmop()*
- 19791 **CHANGE HISTORY**
19792 First released in Issue 5.
- 19793 **Issue 6**
19794 The Open Group corrigenda item U020/1 has been applied, correcting the SYNOPSIS.
19795 The Open Group corrigenda items U032/1 and U032/2 have been applied, clarifying the output
19796 format.

19797

The Open Group Base Resolution bwg98-004 is applied.

19798 **NAME**

19799 jobs — display status of jobs in the current session

19800 **SYNOPSIS**19801 UP jobs [-l | -p][*job_id*...]

19802

19803 **DESCRIPTION**19804 The *jobs* utility shall display the status of jobs that were started in the current shell environment;
19805 see Section 2.13 (on page 2273).19806 When *jobs* reports the termination status of a job, the shell shall remove its process ID from the
19807 list of those “known in the current shell execution environment”; see Section 2.9.3.1 (on page
19808 2259).19809 **OPTIONS**19810 The *jobs* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
19811 12.2, Utility Syntax Guidelines.

19812 The following options shall be supported:

19813 **-l** (The letter ell.) Provide more information about each job listed. This information
19814 shall include the job number, current job, process group ID, state, and the
19815 command that formed the job.19816 **-p** Display only the process IDs for the process group leaders of the selected jobs.19817 By default, the *jobs* utility shall display the status of all stopped jobs, running background jobs
19818 and all jobs whose status has changed and have not been reported by the shell.19819 **OPERANDS**

19820 The following operand shall be supported:

19821 *job_id* Specifies the jobs for which the status is to be displayed. If no *job_id* is given, the
19822 status information for all jobs shall be displayed. The format of *job_id* is described
19823 in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 3.205, Job Control
19824 Job ID.19825 **STDIN**

19826 Not used.

19827 **INPUT FILES**

19828 None.

19829 **ENVIRONMENT VARIABLES**19830 The following environment variables shall affect the execution of *jobs*:19831 **LANG** Provide a default value for the internationalization variables that are unset or null.
19832 If **LANG** is unset or null, the corresponding value from the implementation-
19833 defined default locale shall be used. If any of the internationalization variables
19834 contains an invalid setting, the utility shall behave as if none of the variables had
19835 been defined.19836 **LC_ALL** If set to a non-empty string value, override the values of all the other
19837 internationalization variables.19838 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
19839 characters (for example, single-byte as opposed to multi-byte characters in
19840 arguments).

- 19841 **LC_MESSAGES**
- 19842 Determine the locale that should be used to affect the format and contents of
- 19843 diagnostic messages written to standard error and informative messages written to
- 19844 standard output.
- 19845 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 19846 **ASYNCHRONOUS EVENTS**
- 19847 Default.
- 19848 **STDOUT**
- 19849 If the **-p** option is specified, the output shall consist of one line for each process ID:
- 19850 "%d\n", <process ID>
- 19851 Otherwise, if the **-l** option is not specified, the output shall be a series of lines of the form:
- 19852 "[%d] %c %s %s\n", <job-number>, <current>, <state>, <command>
- 19853 where the fields shall be as follows:
- 19854 <current> The character '+' identifies the job that would be used as a default for the *fg* or *bg*
- 19855 utilities; this job can also be specified using the *job_id* %+ or "%%". The character
- 19856 '-' identifies the job that would become the default if the current default job were to
- 19857 exit; this job can also be specified using the *job_id* %-. For other jobs, this field is
- 19858 a <space> character. At most one job can be identified with '+' and at most one
- 19859 job can be identified with '-'. If there is any suspended job, then the current job
- 19860 shall be a suspended job. If there are at least two suspended jobs, then the previous
- 19861 job also shall be a suspended job.
- 19862 <job-number> A number that can be used to identify the process group to the *wait*, *fg*, *bg*, and *kill*
- 19863 utilities. Using these utilities, the job can be identified by prefixing the job number
- 19864 with '% '.
- 19865 <state> One of the following strings (in the POSIX locale):
- 19866 **Running** Indicates that the job has not been suspended by a signal and has not
- 19867 exited.
- 19868 **Done** Indicates that the job completed and returned exit status zero.
- 19869 **Done(code)** Indicates that the job completed normally and that it exited with the
- 19870 specified non-zero exit status, *code*, expressed as a decimal number.
- 19871 **Stopped** Indicates that the job was suspended by the SIGTSTP signal.
- 19872 **Stopped (SIGTSTP)**
- 19873 Indicates that the job was suspended by the SIGTSTP signal.
- 19874 **Stopped (SIGSTOP)**
- 19875 Indicates that the job was suspended by the SIGSTOP signal.
- 19876 **Stopped (SIGTTIN)**
- 19877 Indicates that the job was suspended by the SIGTTIN signal.
- 19878 **Stopped (SIGTTOU)**
- 19879 Indicates that the job was suspended by the SIGTTOU signal.
- 19880 The implementation may substitute the string **Suspended** in place of **Stopped**. If
- 19881 the job was terminated by a signal, the format of <state> is unspecified, but it shall
- 19882 be visibly distinct from all of the other <state> formats shown here and shall
- 19883 indicate the name or description of the signal causing the termination.

- 19884 <*command*> The associated command that was given to the shell.
- 19885 If the **-I** option is specified, a field containing the process group ID shall be inserted before the
19886 <*state*> field. Also, more processes in a process group may be output on separate lines, using
19887 only the process ID and <*command*> fields.
- 19888 **STDERR**
- 19889 Used only for diagnostic messages.
- 19890 **OUTPUT FILES**
- 19891 None.
- 19892 **EXTENDED DESCRIPTION**
- 19893 None.
- 19894 **EXIT STATUS**
- 19895 The following exit values shall be returned:
- 19896 0 Successful completion.
- 19897 >0 An error occurred.
- 19898 **CONSEQUENCES OF ERRORS**
- 19899 Default.
- 19900 **APPLICATION USAGE**
- 19901 The **-p** option is the only portable way to find out the process group of a job because different
19902 implementations have different strategies for defining the process group of the job. Usage such
19903 as $\$(jobs -p)$ provides a way of referring to the process group of the job in an implementation-
19904 independent way.
- 19905 The *jobs* utility does not work as expected when it is operating in its own utility execution
19906 environment because that environment has no applicable jobs to manipulate. See the
19907 APPLICATION USAGE section for *bg* (on page 2422). For this reason, *jobs* is generally
19908 implemented as a shell regular built-in.
- 19909 **EXAMPLES**
- 19910 None.
- 19911 **RATIONALE**
- 19912 Both "%%" and "%+" are used to refer to the current job. Both forms are of equal validity—the
19913 "%%" mirroring "\$\$" and "%+" mirroring the output of *jobs*. Both forms reflect historical
19914 practice of the KornShell and the C shell with job control.
- 19915 The job control features provided by *bg*, *fg*, and *jobs* are based on the KornShell. The standard
19916 developers examined the characteristics of the C shell versions of these utilities and found that
19917 differences exist. Despite widespread use of the C shell, the KornShell versions were selected for
19918 this volume of IEEE Std. 1003.1-200x to maintain a degree of uniformity with the rest of the
19919 KornShell features selected (such as the very popular command line editing features).
- 19920 The *jobs* utility is not dependent on the job control option, as are the seemingly related *bg* and *fg*
19921 utilities because *jobs* is useful for examining background jobs, regardless of the condition of job
19922 control. When the user has invoked a *set +m* command and job control has been turned off, *jobs*
19923 can still be used to examine the background jobs associated with that current session. Similarly,
19924 *kill* can then be used to kill background jobs with *kill% <background job number>*.
- 19925 The output for terminated jobs is left unspecified to accommodate various historical systems.
19926 The following formats have been witnessed:

- 19927 1. **Killed**(*signal name*)
- 19928 2. *signal name*
- 19929 3. *signal name*(**coredump**)
- 19930 4. *signal description*– **core dumped**
- 19931 Most users should be able to understand these formats, although it means that applications have
19932 trouble parsing them.
- 19933 The calculation of job IDs was not described since this would suggest an implementation, which
19934 may impose unnecessary restrictions.
- 19935 In an early proposal, a **-n** option was included to “Display the status of jobs that have changed,
19936 exited, or stopped since the last status report”. It was removed because the shell always writes
19937 any changed status of jobs before each prompt.
- 19938 **FUTURE DIRECTIONS**
- 19939 None.
- 19940 **SEE ALSO**
- 19941 *bg, fg, kill, wait*
- 19942 **CHANGE HISTORY**
- 19943 First released in Issue 4.
- 19944 **Issue 6**
- 19945 This utility is now marked as part of the User Portability Utilities option.
- 19946 The JC shading is removed as job control is mandatory in this issue.

19947 **NAME**

19948 join — relational database operator

19949 **SYNOPSIS**

```
19950 join [-a file_number | -v file_number][-e string][-o list][-t char]
19951 [-1 field][-2 field] file1 file2
```

19952 **DESCRIPTION**

19953 The *join* utility shall perform an equality join on the files *file1* and *file2*. The joined files shall be
 19954 written to the standard output.

19955 The join field is a field in each file on which the files are compared. By default, *join* shall write
 19956 one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The
 19957 output line by default shall consist of the join field, then the remaining fields from *file1*, then the
 19958 remaining fields from *file2*. This format can be changed by using the **-o** option (see below). The
 19959 **-a** option can be used to add unmatched lines to the output. The **-v** option can be used to output
 19960 only unmatched lines.

19961 **Notes to Reviewers**19962 *This section with side shading will not appear in the final copy. - Ed.*

19963 D1, XCU, ERN 265 proposes to add the following text here: "If the same key appears more than
 19964 once in either file, all possible pairwise combinations are output, in unspecified order".

19965 By default, the files *file1* and *file2* should be ordered in the collating sequence of *sort -b* on the
 19966 fields on which they shall be joined, by default the first in each line. All selected output shall be
 19967 written in the same collating sequence.

19968 The default input field separators shall be <blank> characters. In this case, multiple separators
 19969 shall count as one field separator, and leading separators shall be ignored. The default output
 19970 field separator shall be a <space> character.

19971 The field separator and collating sequence can be changed by using the **-t** option (see below).

19972 If the input files are not in the appropriate collating sequence, the results are unspecified.

19973 **OPTIONS**

19974 The *join* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 19975 12.2, Utility Syntax Guidelines.

19976 The following options shall be supported:

19977 **-a file_number**

19978 Produce a line for each unpairable line in file *file_number*, where *file_number* is 1 or
 19979 2, in addition to the default output. If both **-a1** and **-a2** are specified, all unpairable
 19980 lines shall be output.

19981 **-e string** Replace empty output fields in the list selected by **-o** with the string *string*.

19982 **-o list** Construct the output line to comprise the fields specified in *list*, each element of
 19983 which shall have one of the following two forms:

- 19984 1. *file_number.field*, where *file_number* is a file number and *field* is a decimal
 19985 integer field number
- 19986 2. 0 (zero), representing the join field

19987 The elements of *list* shall be either comma-separated or <blank>-separated, as
 19988 specified in Guideline 8 of the Base Definitions volume of IEEE Std. 1003.1-200x,
 19989 Section 12.2, Utility Syntax Guidelines. The fields specified by *list* shall be written

- 19990 for all selected output lines. Fields selected by *list* that do not appear in the input
 19991 shall be treated as empty output fields. (See the `-e` option.) Only specifically
 19992 requested fields shall be written. The application shall ensure that *list* is a single
 19993 command line argument.
- 19994 `-t char` Use character *char* as a separator, for both input and output. Every appearance of
 19995 *char* in a line shall be significant. When this option is specified, the collating
 19996 sequence should be the same as *sort* without the `-b` option.
- 19997 `-v file_number`
 19998 Instead of the default output, produce a line only for each unpairable line in
 19999 *file_number*, where *file_number* is 1 or 2. If both `-v1` and `-v2` are specified, all
 20000 unpairable lines shall be output.
- 20001 `-1 field` Join on the *field*th field of file 1. Fields are decimal integers starting with 1.
 20002 `-2 field` Join on the *field*th field of file 2. Fields are decimal integers starting with 1.
- 20003 **OPERANDS**
 20004 The following operands shall be supported:
- 20005 *file1, file2*
 20006 A path name of a file to be joined. If either of the *file1* or *file2* operands is '-', the
 20007 standard input shall be used in its place.
- 20008 **STDIN**
 20009 The standard input shall be used only if the *file1* or *file2* operand is '-'. See the INPUT FILES
 20010 section.
- 20011 **INPUT FILES**
 20012 The input files shall be text files.
- 20013 **ENVIRONMENT VARIABLES**
 20014 The following environment variables shall affect the execution of *join*:
- 20015 *LANG* Provide a default value for the internationalization variables that are unset or null.
 20016 If *LANG* is unset or null, the corresponding value from the implementation-
 20017 defined default locale shall be used. If any of the internationalization variables
 20018 contains an invalid setting, the utility shall behave as if none of the variables had
 20019 been defined.
- 20020 *LC_ALL* If set to a non-empty string value, override the values of all the other
 20021 internationalization variables.
- 20022 *LC_COLLATE*
 20023 Determine the locale of the collating sequence *join* expects to have been used when
 20024 the input files were sorted.
- 20025 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 20026 characters (for example, single-byte as opposed to multi-byte characters in
 20027 arguments and input files).
- 20028 *LC_MESSAGES*
 20029 Determine the locale that should be used to affect the format and contents of
 20030 diagnostic messages written to standard error.
- 20031 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

20032 **ASYNCHRONOUS EVENTS**

20033 Default.

20034 **STDOUT**

20035 The *join* utility output shall be a concatenation of selected character fields. When the **-o** option
 20036 is not specified, the output shall be:

20037 "%s%s%s\n", <join field>, <other file1 fields>,
 20038 <other file2 fields>

20039 If the join field is not the first field in a file, the <other file fields> for that file shall be:

20040 <fields preceding join field>, <fields following join field>

20041 When the **-o** option is specified, the output format shall be:

20042 "%s\n", <concatenation of fields>

20043 where the concatenation of fields is described by the **-o** option, above.

20044 For either format, each field (except the last) shall be written with its trailing separator character.
 20045 If the separator is the default (<blank> characters), a single <space> character shall be written
 20046 after each field (except the last).

20047 **STDERR**

20048 Used only for diagnostic messages.

20049 **OUTPUT FILES**

20050 None.

20051 **EXTENDED DESCRIPTION**

20052 None.

20053 **EXIT STATUS**

20054 The following exit values shall be returned:

20055 0 All input files were output successfully.

20056 >0 An error occurred.

20057 **CONSEQUENCES OF ERRORS**

20058 Default.

20059 **APPLICATION USAGE**

20060 Path names consisting of numeric digits or of the form *string.string* should not be specified
 20061 directly following the **-o** list.

20062 **EXAMPLES**

20063 The **-o 0** field essentially selects the union of the join fields. For example, given file **phone**:

20064	!Name	Phone Number
20065	Don	+1 123-456-7890
20066	Hal	+1 234-567-8901
20067	Yasushi	+2 345-678-9012

20068 and file **fax**:

20069	!Name	Fax Number
20070	Don	+1 123-456-7899
20071	Keith	+1 456-789-0122
20072	Yasushi	+2 345-678-9011

20073 (where the large expanses of white space are meant to each represent a single <tab> character),
20074 the command:

```
20075 join -t "<tab>" -a 1 -a 2 -e '(unknown)' -o 0,1.2,2.2 phone fax
```

20076 would produce:

20077	!Name	Phone Number	Fax Number
20078	Don	+1 123-456-7890	+1 123-456-7899
20079	Hal	+1 234-567-8901	(unknown)
20080	Keith	(unknown)	+1 456-789-0122
20081	Yasushi	+2 345-678-9012	+2 345-678-9011

20082 **Notes to Reviewers**

20083 *This section with side shading will not appear in the final copy. - Ed.*

20084 D1, XCU, ERN 265 proposes to add the following example.

20085 The following:

20086 fa:

20087 a x

20088 a y

20089 a z

20090 fb:

20091 a p

20092 a q

20093 would produce:

20094 a x p

20095 a x q

20096 a y p

20097 a y q

20098 a z p

20099 a z q

20100 **RATIONALE**

20101 The `-e` option is only effective when used with `-o` because, unless specific fields are identified
20102 using `-o`, `join` is not aware of what fields might be empty. The exception to this is the join field,
20103 but identifying an empty join field with the `-e` string is not historical practice and some scripts
20104 might break if this were changed.

20105 The 0 field in the `-o` list was adopted from the Tenth Edition version of `join` to satisfy
20106 international objections that the `join` in the base documents do not support the “full join” or
20107 “outer join” described in relational database literature. Although it has been possible to include
20108 a join field in the output (by default, or by field number using `-o`), the join field could not be
20109 included for an unpaired line selected by `-a`. The `-o 0` field essentially selects the union of the
20110 join fields.

20111 This sort of outer join was not possible with the `join` commands in the base documents. The `-o 0`
20112 field was chosen because it is an upward-compatible change for applications. An alternative was
20113 considered: have the join field represent the union of the fields in the files (where they are
20114 identical for matched lines, and one or both are null for unmatched lines). This was not adopted
20115 because it would break some historical applications.

20116 The ability to specify `file2` as `-` is not historical practice; it was added for completeness.

- 20117 The `-v` option is not historical practice, but was considered necessary because it permitted the
20118 writing of *only* those lines that do not match on the join field, as opposed to the `-a` option, which
20119 prints both lines that do and do not match. This additional facility is parallel with the `-v` option
20120 of *grep*.
- 20121 Some historical implementations have been encountered where a blank line in one of the input
20122 files was considered to be the end of the file; the description in this volume of
20123 IEEE Std. 1003.1-200x does not cite this as an allowable case.
- 20124 **FUTURE DIRECTIONS**
- 20125 None.
- 20126 **SEE ALSO**
- 20127 *awk, comm, sort, uniq*
- 20128 **CHANGE HISTORY**
- 20129 First released in Issue 2.
- 20130 **Issue 4**
- 20131 Aligned with the ISO/IEC 9945-2:1993 standard.
- 20132 **Issue 6**
- 20133 The obsolescent `-j` options and the multi-argument `-o` option are withdrawn in this issue.
- 20134 The normative text is reworded to avoid use of the term “must” for application requirements.

20135 NAME

20136 kill — terminate or signal processes

20137 SYNOPSIS

20138 kill -s *signal_name* *pid*...

20139 kill -l [*exit_status*]

20140 DESCRIPTION

20141 The *kill* utility shall send a signal to the process or processes specified by each *pid* operand.

20142 For each *pid* operand, the *kill* utility shall perform actions equivalent to the *kill()* function
20143 defined in the System Interfaces volume of IEEE Std. 1003.1-200x called with the following
20144 arguments:

- 20145 • The value of the *pid* operand shall be used as the *pid* argument.
- 20146 • The *sig* argument is the value specified by the *-s* option, *-signal_number* option, or the
20147 *-signal_name* option, or by SIGTERM, if none of these options is specified.

20148 OPTIONS

20149 The *kill* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
20150 12.2, Utility Syntax Guidelines.

20151 The following options shall be supported:

20152 -l (The letter ell.) Write all values of *signal_name* supported by the implementation, if
20153 no operand is given. If an *exit_status* operand is given and it is a value of the '?'
20154 shell special parameter (see Section 2.5.2 (on page 2241) and *wait* (on page 3254))
20155 corresponding to a process that was terminated by a signal, the *signal_name*
20156 corresponding to the signal that terminated the process shall be written. If an
20157 *exit_status* operand is given and it is the unsigned decimal integer value of a signal
20158 number, the *signal_name* (the symbolic constant name without the **SIG** prefix
20159 defined in the Base Definitions volume of IEEE Std. 1003.1-200x) corresponding to
20160 that signal shall be written. Otherwise, the results are unspecified.

20161 -s *signal_name*

20162 Specify the signal to send, using one of the symbolic names defined in the
20163 <**signal.h**> header defined in the Base Definitions volume of IEEE Std. 1003.1-200x,
20164 Chapter 13, Headers. Values of *signal_name* shall be recognized in a case-
20165 independent fashion, without the **SIG** prefix. In addition, the symbolic name 0
20166 shall be recognized, representing the signal value zero. The corresponding signal
20167 shall be sent instead of SIGTERM.

20168 OPERANDS

20169 The following operands shall be supported:

20170 *pid* One of the following:

- 20171 1. A decimal integer specifying a process or process group to be signaled. The
20172 process or processes selected by positive, negative and zero values of the *pid*
20173 operand shall be as described for the *kill()* function defined in the System
20174 Interfaces volume of IEEE Std. 1003.1-200x. If process number 0 is specified,
20175 all processes in the current process group are signaled. For the effects of
20176 negative *pid* numbers, see the *kill()* function defined in the System Interfaces
20177 volume of IEEE Std. 1003.1-200x. If the first *pid* operand is negative, it should
20178 be preceded by "--" to keep it from being interpreted as an option.

- 20179 2. A job control job ID (see the Base Definitions volume of
 20180 IEEE Std. 1003.1-200x, Section 3.205, Job Control Job ID) that identifies a
 20181 background process group to be signaled. The job control job ID notation is
 20182 applicable only for invocations of *kill* in the current shell execution
 20183 environment; see Section 2.13 (on page 2273).
- 20184 *exit_status* A decimal integer specifying a signal number or the exit status of a process
 20185 terminated by a signal.
- 20186 **STDIN**
 20187 Not used.
- 20188 **INPUT FILES**
 20189 None.
- 20190 **ENVIRONMENT VARIABLES**
 20191 The following environment variables shall affect the execution of *kill*:
- 20192 *LANG* Provide a default value for the internationalization variables that are unset or null.
 20193 If *LANG* is unset or null, the corresponding value from the implementation-
 20194 defined default locale shall be used. If any of the internationalization variables
 20195 contains an invalid setting, the utility shall behave as if none of the variables had
 20196 been defined.
- 20197 *LC_ALL* If set to a non-empty string value, override the values of all the other
 20198 internationalization variables.
- 20199 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 20200 characters (for example, single-byte as opposed to multi-byte characters in
 20201 arguments).
- 20202 *LC_MESSAGES*
 20203 Determine the locale that should be used to affect the format and contents of
 20204 diagnostic messages written to standard error.
- 20205 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 20206 **ASYNCHRONOUS EVENTS**
 20207 Default.
- 20208 **STDOUT**
 20209 When the **-I** option is not specified, the standard output shall not be used.
- 20210 When the **-I** option is specified, the symbolic name of each signal shall be written in the
 20211 following format:
- 20212 "%s%c", <signal_name>, <separator>
- 20213 where the <signal_name> is in uppercase, without the **SIG** prefix, and the <separator> shall be
 20214 either a <newline> character or a <space> character. For the last signal written, <separator> shall
 20215 be a <newline> character.
- 20216 When both the **-I** option and *exit_status* operand are specified, the symbolic name of the
 20217 corresponding signal shall be written in the following format:
- 20218 "%s\n", <signal_name>

20219 **STDERR**

20220 Used only for diagnostic messages.

20221 **OUTPUT FILES**

20222 None.

20223 **EXTENDED DESCRIPTION**

20224 None.

20225 **EXIT STATUS**

20226 The following exit values shall be returned:

20227 0 At least one matching process was found for each *pid* operand, and the specified signal was
20228 successfully processed for at least one matching process.

20229 >0 An error occurred.

20230 **CONSEQUENCES OF ERRORS**

20231 Default.

20232 **APPLICATION USAGE**

20233 Process numbers can be found by using *ps*.

20234 The job control job ID notation is not required to work as expected when *kill* is operating in its
20235 own utility execution environment. In either of the following examples:

```
20236           nohup kill %1 &
20237           system("kill %1");
```

20238 the *kill* operates in a different environment and does not share the shell's understanding of job
20239 numbers.

20240 **EXAMPLES**

20241 Any of the commands:

```
20242           kill -s kill 100 -165
20243           kill -s KILL 100 -165
```

20244 sends the SIGKILL signal to the process whose process ID is 100 and to all processes whose
20245 process group ID is 165, assuming the sending process has permission to send that signal to the
20246 specified processes, and that they exist.

20247 The System Interfaces volume of IEEE Std. 1003.1-200x and this volume of IEEE Std. 1003.1-200x
20248 do not require specific signal numbers for any *signal_names*. Even the *-signal_number* option
20249 provides symbolic (although numeric) names for signals. If a process is terminated by a signal,
20250 its exit status indicates the signal that killed it, but the exact values are not specified. The *kill -l*
20251 option, however, can be used to map decimal signal numbers and exit status values into the
20252 name of a signal. The following example reports the status of a terminated job:

```
20253           job
20254           stat=$?
20255           if [ $stat -eq 0 ]
20256           then
20257                echo job completed successfully.
20258           elif [ $stat -gt 128 ]
20259           then
20260                echo job terminated by signal SIG$(kill -l $stat).
20261           else
20262                echo job terminated with error code $stat.
20263           fi
```

20264 To avoid an ambiguity of an initial negative number argument specifying either a signal number
20265 or a process group, the ISO/IEC 9945-2: 1993 standard mandates that it always be considered the
20266 former. Therefore, to send the default signal to a process group (say 123), an application should
20267 use a command similar to one of the following:

20268 `kill -TERM -123`

20269 `kill -- -123`

20270 RATIONALE

20271 The `-l` option originated from the C shell, and is also implemented in the KornShell. The C shell
20272 output can consist of multiple output lines because the signal names do not always fit on a
20273 single line on some terminal screens. The KornShell output also included the implementation-
20274 defined signal numbers and was considered by the standard developers to be too difficult for
20275 scripts to parse conveniently. The specified output format is intended not only to accommodate
20276 the historical C shell output, but also to permit an entirely vertical or entirely horizontal listing
20277 on systems for which this is appropriate.

20278 An early proposal invented the name `SIGNULL` as a *signal_name* for signal 0 (used by the System
20279 Interfaces volume of IEEE Std. 1003.1-200x to test for the existence of a process without sending
20280 it a signal). Since the *signal_name* 0 can be used in this case unambiguously, `SIGNULL` has been
20281 removed.

20282 An early proposal also required symbolic *signal_names* to be recognized with or without the **SIG**
20283 prefix. Historical versions of *kill* have not written the **SIG** prefix for the `-l` option and have not
20284 recognized the **SIG** prefix on *signal_names*. Since neither applications portability nor ease-of-use
20285 would be improved by requiring this extension, it is no longer required.

20286 This volume of IEEE Std. 1003.1-200x contains no utility that browses for process IDs. Values for
20287 *pid* are available via the `'!'` and `'$'` parameters of the shell command language.

20288 The `-s` option was added in response to international interest in providing some form of *kill* that
20289 meets the Utility Syntax Guidelines.

20290 The job control job ID notation is not required to work as expected when *kill* is operating in its
20291 own utility execution environment. In either of the following examples:

```
20292 nohup kill %1 &  
20293 system("kill %1");
```

20294 the *kill* operates in a different environment and does not understand how the shell has managed
20295 its job numbers.

20296 FUTURE DIRECTIONS

20297 None.

20298 SEE ALSO

20299 *ps*, *wait*, the System Interfaces volume of IEEE Std. 1003.1-200x, *kill()*, `<signal.h>`

20300 CHANGE HISTORY

20301 First released in Issue 2.

20302 Issue 4

20303 Aligned with the ISO/IEC 9945-2: 1993 standard.

20304 Issue 6

20305 The obsolescent versions of the SYNOPSIS are withdrawn in this issue.

20306 NAME

20307 `lex` — generate programs for lexical tasks (**DEVELOPMENT**)

20308 SYNOPSIS

20309 CD `lex -c [-t][-n] [-v][file ...]`

20310

20311 DESCRIPTION

20312 The *lex* utility shall generate C programs to be used in lexical processing of character input, and
 20313 that can be used as an interface to *yacc*. The C programs shall be generated from *lex* source code
 20314 and conform to the ISO C standard. Usually, the *lex* utility shall write the program it generates to
 20315 the file `lex.yy.c`; the state of this file is unspecified if *lex* exits with a non-zero exit status. See the
 20316 EXTENDED DESCRIPTION section for a complete description of the *lex* input language.

20317 OPTIONS

20318 The *lex* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 20319 12.2, Utility Syntax Guidelines.

20320 The following options shall be supported:

20321 `-n` Suppress the summary of statistics usually written with the `-v` option. If no table
 20322 sizes are specified in the *lex* source code and the `-v` option is not specified, then `-n`
 20323 is implied.

20324 `-t` Write the resulting program to standard output instead of `lex.yy.c`.

20325 `-v` Write a summary of *lex* statistics to the standard output. (See the discussion of *lex*
 20326 table sizes in **Definitions in lex** (on page 2745).) If the `-t` option is specified and
 20327 `-n` is not specified, this report shall be written to standard error. If table sizes are
 20328 specified in the *lex* source code, and if the `-n` option is not specified, the `-v` option
 20329 may be enabled.

20330 OPERANDS

20331 The following operand shall be supported:

20332 *file* A path name of an input file. If more than one such *file* is specified, all files shall be
 20333 concatenated to produce a single *lex* program. If no *file* operands are specified, or if
 20334 a *file* operand is `'-'`, the standard input shall be used.

20335 STDIN

20336 The standard input shall be used if no *file* operands are specified, or if a *file* operand is `'-'`. See
 20337 INPUT FILES.

20338 INPUT FILES

20339 The input files shall be text files containing *lex* source code, as described in the EXTENDED
 20340 DESCRIPTION section.

20341 ENVIRONMENT VARIABLES

20342 If this variable is not set to the POSIX locale, the results are unspecified.

20343 The following environment variables shall affect the execution of *lex*:

20344 *LANG* Provide a default value for the internationalization variables that are unset or null.
 20345 If *LANG* is unset or null, the corresponding value from the implementation-
 20346 defined default locale shall be used. If any of the internationalization variables
 20347 contains an invalid setting, the utility shall behave as if none of the variables had
 20348 been defined.

20349 *LC_ALL* If set to a non-empty string value, override the values of all the other
 20350 internationalization variables.

- 20351 *LC_COLLATE*
 20352 Determine the locale for the behavior of ranges, equivalence classes and multi-
 20353 character collating elements within regular expressions. If this variable is not set to
 20354 the POSIX locale, the results are unspecified.
- 20355 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 20356 characters (for example, single-byte as opposed to multi-byte characters in
 20357 arguments and input files), and the behavior of character classes within regular
 20358 expressions. If this variable is not set to the POSIX locale, the results are
 20359 unspecified.
- 20360 *LC_MESSAGES*
 20361 Determine the locale that should be used to affect the format and contents of
 20362 diagnostic messages written to standard error.
- 20363 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 20364 **ASYNCHRONOUS EVENTS**
 20365 Default.
- 20366 **STDOUT**
 20367 If the `-t` option is specified, the text file of C source code output of *lex* shall be written to
 20368 standard output.
- 20369 If the `-t` option is not specified:
- 20370 • Implementation-defined informational, error, and warning messages concerning the contents
 20371 of *lex* source code input shall be written to either the standard output or standard error.
 - 20372 • If the `-v` option is specified and the `-n` option is not specified, *lex* statistics shall also be
 20373 written to either the standard output or standard error, in an implementation-defined format.
 20374 These statistics may also be generated if table sizes are specified with a '%' operator in the
 20375 *Definitions* section, as long as the `-n` option is not specified.
- 20376 **STDERR**
 20377 If the `-t` option is specified, implementation-defined informational, error, and warning messages
 20378 concerning the contents of *lex* source code input shall be written to the standard error.
- 20379 If the `-t` option is not specified:
- 20380 1. Implementation-defined informational, error, and warning messages concerning the
 20381 contents of *lex* source code input shall be written to either the standard output or standard
 20382 error.
 - 20383 2. If the `-v` option is specified and the `-n` option is not specified, *lex* statistics shall also be
 20384 written to either the standard output or standard error, in an implementation-defined
 20385 format. These statistics may also be generated if table sizes are specified with a '%' operator
 20386 in the *Definitions* section, as long as the `-n` option is not specified.
- 20387 **OUTPUT FILES**
 20388 A text file containing C source code shall be written to `lex.yy.c`, or to the standard output if the
 20389 `-t` option is present.
- 20390 **EXTENDED DESCRIPTION**
 20391 Each input file contains *lex* source code, which is a table of regular expressions with
 20392 corresponding actions in the form of C program fragments.
- 20393 When `lex.yy.c` is compiled and linked with the *lex* library (using the `-ll` operand with *c99* or *cc*),
 20394 the resulting program reads character input from the standard input and partitions it into strings
 20395 that match the given expressions.

20396 When an expression is matched, these actions shall occur:

- 20397 • The input string that was matched is left in *yytext* as a null-terminated string; *yytext* is either
- 20398 an external character array or a pointer to a character string. As explained in **Definitions in**
- 20399 **lex**, the type can be explicitly selected using the **%array** or **%pointer** declarations, but the
- 20400 default is implementation-defined.
- 20401 • The external **int** *yylen* is set to the length of the matching string.
- 20402 • The expression's corresponding program fragment, or action, is executed.

20403 During pattern matching, *lex* shall search the set of patterns for the single longest possible

20404 match. Among rules that match the same number of characters, the rule given first shall be

20405 chosen.

20406 The general format of *lex* source shall be:

```

20407     Definitions
20408     %%
20409     Rules
20410     %%
20411     UserSubroutines

```

20412 The first "%%" is required to mark the beginning of the rules (regular expressions and actions);

20413 the second "%%" is required only if user subroutines follow.

20414 Any line in the *Definitions* section beginning with a <blank> character shall be assumed to be a C

20415 program fragment and shall be copied to the external definition area of the **lex.yy.c** file.

20416 Similarly, anything in the *Definitions* section included between delimiter lines containing only

20417 "%{" and "%}" shall also be copied unchanged to the external definition area of the **lex.yy.c** file.

20418 Any such input (beginning with a <blank> character or within "%{" and "%}" delimiter lines)

20419 appearing at the beginning of the *Rules* section before any rules are specified shall be written to

20420 **lex.yy.c** after the declarations of variables for the *yylex* function and before the first line of code

20421 in *yylex*. Thus, user variables local to *yylex* can be declared here, as well as application code to

20422 execute upon entry to *yylex*.

20423 The action taken by *lex* when encountering any input beginning with a <blank> character or

20424 within "%{" and "%}" delimiter lines appearing in the *Rules* section but coming after one or

20425 more rules is undefined. The presence of such input may result in an erroneous definition of the

20426 *yylex* function.

20427 **Definitions in lex**

20428 *Definitions* appear before the first "%%" delimiter. Any line in this section not contained between

20429 "%{" and "%}" lines and not beginning with a <blank> character shall be assumed to define a

20430 *lex* substitution string. The format of these lines shall be:

```

20431     name substitute

```

20432 If a *name* does not meet the requirements for identifiers in the ISO C standard, the result is

20433 undefined. The string *substitute* shall replace the string *{name}* when it is used in a rule. The *name*

20434 string shall be recognized in this context only when the braces are provided and when it does

20435 not appear within a bracket expression or within double-quotes.

20436 In the *Definitions* section, any line beginning with a '%' (percent sign) character and followed by

20437 an alphanumeric word beginning with either 's' or 'S' shall define a set of start conditions.

20438 Any line beginning with a '%' followed by a word beginning with either 'x' or 'X' shall define

20439 a set of exclusive start conditions. When the generated scanner is in a "%s" state, patterns with

20440 no state specified shall be also active; in a "%x" state, such patterns shall not be active. The rest
 20441 of the line, after the first word, shall be considered to be one or more <blank> character-
 20442 separated names of start conditions. Start condition names shall be constructed in the same way
 20443 as definition names. Start conditions can be used to restrict the matching of regular expressions
 20444 to one or more states as described in **Regular Expressions in lex** (on page 2747).

20445 Implementations shall accept either of the following two mutually exclusive declarations in the
 20446 *Definitions* section:

20447 **%array** Declare the type of *yytext* to be a null-terminated character array.

20448 **%pointer** Declare the type of *yytext* to be a pointer to a null-terminated character string.

20449 The default type of *yytext* is implementation-defined. If an application refers to *yytext* outside of
 20450 the scanner source file (that is, via an **extern**), the application shall include the appropriate
 20451 **%array** or **%pointer** declaration in the scanner source file.

20452 Implementations shall accept declarations in the *Definitions* section for setting certain internal
 20453 table sizes. The declarations are shown in the following table.

20454 **Table 4-9** Table Size Declarations in *lex*

Declaration	Description	Minimum Value
%p <i>n</i>	Number of positions	2 500
%n <i>n</i>	Number of states	500
%a <i>n</i>	Number of transitions	2 000
%e <i>n</i>	Number of parse tree nodes	1 000
%k <i>n</i>	Number of packed character classes	1 000
%o <i>n</i>	Size of the output array	3 000

20462 In the table, *n* represents a positive decimal integer, preceded by one or more <blank>
 20463 characters. The exact meaning of these table size numbers is implementation-defined. The
 20464 implementation shall document how these numbers affect the *lex* utility and how they are
 20465 related to any output that may be generated by the implementation should space limitations be
 20466 encountered during the execution of *lex*. It shall be possible to determine from this output which
 20467 of the table size values needs to be modified to permit *lex* to successfully generate tables for the
 20468 input language. The values in the column Minimum Value represent the lowest values
 20469 conforming implementations shall provide.

20470 **Rules in lex**

20471 The rules in *lex* source files are a table in which the left column contains regular expressions and
 20472 the right column contains actions (C program fragments) to be executed when the expressions
 20473 are recognized.

20474 *ERE action*

20475 *ERE action*

20476 ...

20477 The extended regular expression (*ERE*) portion of a row shall be separated from *action* by one or
 20478 more <blank> characters. A regular expression containing <blank> characters shall be
 20479 recognized under one of the following conditions:

- 20480 • The entire expression appears within double-quotes.
- 20481 • The <blank> characters appear within double-quotes or square brackets.

- Each <blank> character is preceded by a backslash character.

20483 User Subroutines in lex

20484 Anything in the user subroutines section shall be copied to **lex.yy.c** following *yylex*.

20485 Regular Expressions in lex

20486 The *lex* utility shall support the set of extended regular expressions (see the Base Definitions
20487 volume of IEEE Std. 1003.1-200x, Section 9.4, Extended Regular Expressions), with the following
20488 additions and exceptions to the syntax:

20489 ". . ." Any string enclosed in double-quotes shall represent the characters within the
20490 double-quotes as themselves, except that backslash escapes (which appear in the
20491 following table) shall be recognized. Any backslash-escape sequence shall be
20492 terminated by the closing quote. For example, "\01" "1" represents a single
20493 string: the octal value 1 followed by the character '1'.

20494 <state>*r*, <state1, state2, . . .>*r*

20495 The regular expression *r* shall be matched only when the program is in one of the
20496 start conditions indicated by *state*, *state1*, and so on; see **Actions in lex** (on page
20497 2749). (As an exception to the typographical conventions of the rest of this volume
20498 of IEEE Std. 1003.1-200x, in this case <state> does not represent a metavariable, but
20499 the literal angle-bracket characters surrounding a symbol.) The start condition
20500 shall be recognized as such only at the beginning of a regular expression.

20501 *r/x*

20502 The regular expression *r* shall be matched only if it is followed by an occurrence of
20503 regular expression *x* (*x* is the instance of trailing context, further defined below).
20504 The token returned in *yytext* shall only match *r*. If the trailing portion of *r* matches
20505 the beginning of *x*, the result is unspecified. The *r* expression cannot include
20506 further trailing context or the '\$' (match-end-of-line) operator; *x* cannot include
20507 the '^' (match-beginning-of-line) operator, nor trailing context, nor the '\$'
20508 operator. That is, only one occurrence of trailing context is allowed in a *lex* regular
20509 expression, and the '^' operator only can be used at the beginning of such an
expression.

20510 {*name*}

20511 When *name* is one of the substitution symbols from the *Definitions* section, the
20512 string, including the enclosing braces, shall be replaced by the *substitute* value. The
20513 *substitute* value shall be treated in the extended regular expression as if it were
20514 enclosed in parentheses. No substitution shall occur if {*name*} occurs within a
bracket expression or within double-quotes.

20515 Within an ERE, a backslash character shall be considered to begin an escape sequence as
20516 specified in the table in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 5, File
20517 Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'). In addition, the escape
20518 sequences in the following table shall be recognized.

20519 A literal <newline> character cannot occur within an ERE; the escape sequence '\n' can be
20520 used to represent a <newline> character. A <newline> character shall not be matched by a
20521 period operator.

20522

Table 4-10 Escape Sequences in *lex*

20523

20524

20525

20526

20527

20528

20529

20530

20531

20532

20533

20534

20535

20536

20537

20538

20539

20540

20541

20542

20543

20544

20545

20546

20547

20548

20549

20550

20551

20552

Escape Sequence	Description	Meaning
<code>\digits</code>	A backslash character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is implementation-defined. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading ' <code>\</code> ' for each byte.
<code>\xdigits</code>	A backslash character followed by the longest sequence of hexadecimal-digit characters (01234567abcdefABCDEF). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the hexadecimal integer.
<code>\c</code>	A backslash character followed by any character not described in this table or in the table in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 5, File Format Notation (' <code>\</code> ', ' <code>\a</code> ', ' <code>\b</code> ', ' <code>\f</code> ', ' <code>\n</code> ', ' <code>\r</code> ', ' <code>\t</code> ', ' <code>\v</code> ').	The character ' <code>c</code> ', unchanged.

20553 **Notes to Reviewers**

20554

This section with side shading will not appear in the final copy. - Ed.

20555

20556

20557

D3, XCU, ERN 120, re length limitation for hex, suggests adding a note: "Note: If a hexadecimal escape sequence which is followed by a hexadecimal digit is required, either the character in hex or the following character may be parenthesized using `\(` and `\)`."

20558

20559

20560

20561

The order of precedence given to extended regular expressions for *lex* differs from that specified in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 9.4, Extended Regular Expressions. The order of precedence for *lex* shall be as shown in the following table, from high to low.

20562

20563

20564

20565

20566

Note: The escaped characters entry is not meant to imply that these are operators, but they are included in the table to show their relationships to the true operators. The start condition, trailing context, and anchoring notations have been omitted from the table because of the placement restrictions described in this section; they can only appear at the beginning or ending of an ERE.

20567

Table 4-11 ERE Precedence in *lex*

20568

Extended Regular Expression	Precedence
<i>collation-related bracket symbols</i>	[=] [: :] [. .]
<i>escaped characters</i>	\<special character>
<i>bracket expression</i>	[]
<i>quoting</i>	" . . . "
<i>grouping</i>	()
<i>definition</i>	{ name }
<i>single-character RE duplication</i>	* + ?
<i>concatenation</i>	
<i>interval expression</i>	{ m, n }
<i>alternation</i>	

20569

20570

20571

20572

20573

20574

20575

20576

20577

20578

20579

20580

20581

20582

20583

20584

20585

20586

The ERE anchoring operators '*^*' and '*\$*' do not appear in the table. With *lex* regular expressions, these operators are restricted in their use: the '*^*' operator can only be used at the beginning of an entire regular expression, and the '*\$*' operator only at the end. The operators apply to the entire regular expression. Thus, for example, the pattern "*(^abc)|(def\$)*" is undefined; it can instead be written as two separate rules, one with the regular expression "*^abc*" and one with "*def\$*", which share a common action via the special '*|*' action (see below). If the pattern were written "*^abc|def\$*", it would match either "*abc*" or "*def*" on a line by itself.

20587

20588

20589

20590

Unlike the general ERE rules, embedded anchoring is not allowed by most historical *lex* implementations. An example of embedded anchoring would be for patterns such as "*(^|)foo(|\$)*" to match "*foo*" when it exists as a complete word. This functionality can be obtained using existing *lex* features:

20591

20592

```
^foo/[ \n]      |
" foo"/[ \n]    /* Found foo as a separate word. */
```

20593

20594

20595

Note also that '*\$*' is a form of trailing context (it is equivalent to "*/\n*") and as such cannot be used with regular expressions containing another instance of the operator (see the preceding discussion of trailing context).

20596

20597

20598

20599

20600

The additional regular expressions trailing-context operator '*/'*' can be used as an ordinary character if presented within double-quotes, "*/"*"; preceded by a backslash, "*\"*"; or within a bracket expression, "*[/]*". The start-condition '*<*' and '*>*' operators shall be special only in a start condition at the beginning of a regular expression; elsewhere in the regular expression they shall be treated as ordinary characters.

20601

Actions in *lex*

20602

20603

20604

20605

20606

20607

The action to be taken when an ERE is matched can be a C program fragment or the special actions described below; the program fragment can contain one or more C statements, and can also include special actions. The empty C statement '*;*' shall be a valid action; any string in the *lex.yy.c* input that matches the pattern portion of such a rule is effectively ignored or skipped. However, the absence of an action shall not be valid, and the action *lex* takes in such a condition is undefined.

20608

20609

The specification for an action, including C statements and special actions, can extend across several lines if enclosed in braces:

20610

20611

```
ERE <one or more blanks> { program statement
                          program statement }
```

20612 The default action when a string in the input to a **lex.yy.c** program is not matched by any
 20613 expression shall be to copy the string to the output. Because the default behavior of a program
 20614 generated by *lex* is to read the input and copy it to the output, a minimal *lex* source program that
 20615 has just "%%" shall generate a C program that simply copies the input to the output unchanged.

20616 Four special actions shall be available:

20617 | ECHO; REJECT; BEGIN

20618 | The action ' | ' means that the action for the next rule is the action for this rule.
 20619 Unlike the other three actions, ' | ' cannot be enclosed in braces or be semicolon-
 20620 terminated; the application shall ensure that it is specified alone, with no other
 20621 actions.

20622 **ECHO;** Write the contents of the string *yytext* on the output.

20623 **REJECT;** Usually only a single expression is matched by a given string in the input. **REJECT**
 20624 means "continue to the next expression that matches the current input", and shall
 20625 cause whatever rule was the second choice after the current rule to be executed for
 20626 the same input. Thus, multiple rules can be matched and executed for one input
 20627 string or overlapping input strings. For example, given the regular expressions
 20628 "xyz" and "xy" and the input "xyz", usually only the regular expression "xyz"
 20629 would match. The next attempted match would start after z. If the last action in the
 20630 "xyz" rule is **REJECT**, both this rule and the "xy" rule would be executed. The
 20631 **REJECT** action may be implemented in such a fashion that flow of control does not
 20632 continue after it, as if it were equivalent to a **goto** to another part of *yylex*. The use
 20633 of **REJECT** may result in somewhat larger and slower scanners.

20634 **BEGIN** The action:

20635 BEGIN *newstate*;

20636 switches the state (start condition) to *newstate*. If the string *newstate* has not been
 20637 declared previously as a start condition in the *Definitions* section, the results are
 20638 unspecified. The initial state is indicated by the digit '0' or the token **INITIAL**.

20639 The functions or macros described below are accessible to user code included in the *lex* input. It
 20640 is unspecified whether they appear in the C code output of *lex*, or are accessible only through the
 20641 **-ll** operand to *c99* or *cc* (the *lex* library).

20642 **int yylex(void)**

20643 Performs lexical analysis on the input; this is the primary function generated by the *lex*
 20644 utility. The function shall return zero when the end of input is reached; otherwise, it shall
 20645 return non-zero values (tokens) determined by the actions that are selected.

20646 **int yymore(void)**

20647 When called, indicates that when the next input string is recognized, it is to be appended to
 20648 the current value of *yytext* rather than replacing it; the value in *yyleng* shall be adjusted
 20649 accordingly.

20650 **int yless(int n)**

20651 Retains *n* initial characters in *yytext*, NUL-terminated, and treats the remaining characters
 20652 as if they had not been read; the value in *yyleng* shall be adjusted accordingly.

20653 **int input(void)**

20654 Returns the next character from the input, or zero on end-of-file. It shall obtain input from
 20655 the stream pointer *yyin*, although possibly via an intermediate buffer. Thus, once scanning
 20656 has begun, the effect of altering the value of *yyin* is undefined. The character read is
 20657 removed from the input stream of the scanner without any processing by the scanner.

20658 **int unput(int c)**
 20659 Returns the character 'c' to the input; *yytext* and *yylen* are undefined until the next
 20660 expression is matched. The result of using *unput* for more characters than have been input is
 20661 unspecified.

20662 The following functions appear only in the *lex* library accessible through the `-ll` operand; they
 20663 can therefore be redefined by a portable application:

20664 **int yywrap(void)**
 20665 Called by *yylex* at end-of-file; the default *yywrap* always shall return 1. If the application
 20666 requires *yylex* to continue processing with another source of input, then the application can
 20667 include a function *yywrap*, which associates another file with the external variable `FILE*yyin`
 20668 and shall return a value of zero.

20669 **int main(int argc, char *argv[])**
 20670 Calls *yylex* to perform lexical analysis, then exits. The user code can contain *main* to perform
 20671 application-specific operations, calling *yylex* as applicable.

20672 Except for *input*, *unput*, and *main*, all external and static names generated by *lex* shall begin with
 20673 the prefix `yy` or `YY`.

20674 EXIT STATUS

20675 The following exit values shall be returned:

20676 0 Successful completion.

20677 >0 An error occurred.

20678 CONSEQUENCES OF ERRORS

20679 Default.

20680 APPLICATION USAGE

20681 Portable applications are warned that in the *Rules* section, an *ERE* without an action is not
 20682 acceptable, but need not be detected as erroneous by *lex*. This may result in compilation or
 20683 runtime errors.

20684 The purpose of *input* is to take characters off the input stream and discard them as far as the
 20685 lexical analysis is concerned. A common use is to discard the body of a comment once the
 20686 beginning of a comment is recognized.

20687 The *lex* utility is not fully internationalized in its treatment of regular expressions in the *lex*
 20688 source code or generated lexical analyzer. It would seem desirable to have the lexical analyzer
 20689 interpret the regular expressions given in the *lex* source according to the environment specified
 20690 when the lexical analyzer is executed, but this is not possible with the current *lex* technology.
 20691 Furthermore, the very nature of the lexical analyzers produced by *lex* must be closely tied to the
 20692 lexical requirements of the input language being described, which is frequently locale-specific
 20693 anyway. (For example, writing an analyzer that is used for French text is not automatically
 20694 useful for processing other languages.)

20695 EXAMPLES

20696 The following is an example of a *lex* program that implements a rudimentary scanner for a
 20697 Pascal-like syntax:

```
20698 %{\n
20699 /* Need this for the call to atof() below. */\n
20700 #include <math.h>\n
20701 /* Need this for printf(), fopen(), and stdin below. */\n
20702 #include <stdio.h>\n
20703 %}
```

```

20704     DIGIT    [0-9]
20705     ID      [a-z][a-z0-9]*
20706     %%
20707     {DIGIT}+ {
20708         printf("An integer: %s (%d)\n", yytext,
20709             atoi(yytext));
20710     }
20711     {DIGIT}+"."{DIGIT}* {
20712         printf("A float: %s (%g)\n", yytext,
20713             atof(yytext));
20714     }
20715     if|then|begin|end|procedure|function {
20716         printf("A keyword: %s\n", yytext);
20717     }
20718     {ID}    printf("An identifier: %s\n", yytext);
20719     "+"|"-"|"*"|"|" /"    printf("An operator: %s\n", yytext);
20720     "{ "[^]\n]*" /* Eat up one-line comments. */
20721     [ \t\n]+ /* Eat up white space. */
20722     . printf("Unrecognized character: %s\n", yytext);
20723     %%
20724     int main(int argc, char *argv[])
20725     {
20726         ++argv, --argc; /* Skip over program name. */
20727         if (argc > 0)
20728             yyin = fopen(argv[0], "r");
20729         else
20730             yyin = stdin;
20731         yylex();
20732     }

```

20733 RATIONALE

20734 Even though the `-c` option and references to the C language are retained in this description, *lex*
 20735 may be generalized to other languages, as was done at one time for EFL, the Extended
 20736 FORTRAN Language. Since the *lex* input specification is essentially language-independent,
 20737 versions of this utility could be written to produce Ada, Modula-2, or Pascal code, and there are
 20738 known historical implementations that do so.

20739 The current description of *lex* bypasses the issue of dealing with internationalized EREs in the *lex*
 20740 source code or generated lexical analyzer. If it follows the model used by *awk* (the source code is
 20741 assumed to be presented in the POSIX locale, but input and output are in the locale specified by
 20742 the environment variables), then the tables in the lexical analyzer produced by *lex* would
 20743 interpret EREs specified in the *lex* source in terms of the environment variables specified when
 20744 *lex* was executed. The desired effect would be to have the lexical analyzer interpret the EREs
 20745 given in the *lex* source according to the environment specified when the lexical analyzer is
 20746 executed, but this is not possible with the current *lex* technology.

20747 The description of octal and hexadecimal-digit escape sequences agrees with the ISO C standard
 20748 use of escape sequences. See the RATIONALE for *ed* (on page 2546) for a discussion of bytes

- 20749 larger than 9 bits being represented by octal values. Hexadecimal values can represent larger
20750 bytes and multi-byte characters directly, using as many digits as required.
- 20751 There is no detailed output format specification. The observed behavior of *lex* under four
20752 different historical implementations was that none of these implementations consistently
20753 reported the line numbers for error and warning messages. Furthermore, there was a desire that
20754 *lex* be allowed to output additional diagnostic messages. Leaving message formats unspecified
20755 avoids these formatting questions and problems with internationalization.
- 20756 Although the `%x` specifier for *exclusive* start conditions is not historical practice, it is believed to
20757 be a minor change to historical implementations and greatly enhances the usability of *lex*
20758 programs since it permits an application to obtain the expected functionality with fewer
20759 statements.
- 20760 The `%array` and `%pointer` declarations were added as a compromise between historical systems.
20761 The System V-based *lex* copies the matched text to a *yytext* array. The *flex* program, supported in
20762 BSD and GNU systems, uses a pointer. In the latter case, significant performance improvements
20763 are available for some scanners. Most historical programs should require no change in porting
20764 from one system to another because the string being referenced is null-terminated in both cases.
20765 (The method used by *flex* in its case is to null-terminate the token in place by remembering the
20766 character that used to come right after the token and replacing it before continuing on to the next
20767 scan.) Multi-file programs with external references to *yytext* outside the scanner source file
20768 should continue to operate on their historical systems, but would require one of the new
20769 declarations to be considered strictly portable.
- 20770 The description of EREs avoids unnecessary duplication of ERE details because their meanings
20771 within a *lex* ERE are the same as that for the ERE in this volume of IEEE Std. 1003.1-200x.
- 20772 The reason for the undefined condition associated with text beginning with a <blank> or within
20773 "%{ " and "%}" delimiter lines appearing in the *Rules* section is historical practice. Both the BSD
20774 and System V *lex* copy the indented (or enclosed) input in the *Rules* section (except at the
20775 beginning) to unreachable areas of the *yylex* function (the code is written directly after a *break*
20776 statement). In some cases, the System V *lex* generates an error message or a syntax error,
20777 depending on the form of indented input.
- 20778 The intention in breaking the list of functions into those that may appear in *lex.yy.c* versus those
20779 that only appear in *libl.a* is that only those functions in *libl.a* can be reliably redefined by a
20780 portable application.
- 20781 The descriptions of standard output and standard error are somewhat complicated because
20782 historical *lex* implementations chose to issue diagnostic messages to standard output (unless `-t`
20783 was given). This standard allows this behavior, but leaves an opening for the more expected
20784 behavior of using standard error for diagnostics. Also, the System V behavior of writing the
20785 statistics when any table sizes are given is allowed, while BSD-derived systems can avoid it. The
20786 programmer can always precisely obtain the desired results by using either the `-t` or `-n` options.
- 20787 The OPERANDS section does not mention the use of `-` as a synonym for standard input; not all
20788 historical implementations support such usage for any of the *file* operands.
- 20789 A description of the *translation table* was deleted from early proposals because of its relatively
20790 low usage in historical applications.
- 20791 The change to the definition of the *input* function that allows buffering of input presents the
20792 opportunity for major performance gains in some applications.
- 20793 The following examples clarify the differences between *lex* regular expressions and regular
20794 expressions appearing elsewhere in this volume of IEEE Std. 1003.1-200x. For regular
20795 expressions of the form `"r/x"`, the string matching *r* is always returned; confusion may arise

- 20796 when the beginning of *x* matches the trailing portion of *r*. For example, given the regular
20797 expression "a*b/cc" and the input "aaabcc", *yytext* would contain the string "aaab" on this
20798 match. But given the regular expression "x*/xy" and the input "xxxxy", the token **xxx**, not **xx**,
20799 is returned by some implementations because **xxx** matches "x*".
- 20800 In the rule "ab*/bc", the "b*" at the end of *r* extends *r*'s match into the beginning of the
20801 trailing context, so the result is unspecified. If this rule were "ab/bc", however, the rule
20802 matches the text "ab" when it is followed by the text "bc". In this latter case, the matching of *r*
20803 cannot extend into the beginning of *x*, so the result is specified.
- 20804 **FUTURE DIRECTIONS**
- 20805 None.
- 20806 **SEE ALSO**
- 20807 *c99, yacc*
- 20808 **CHANGE HISTORY**
- 20809 First released in Issue 2.
- 20810 **Issue 4**
- 20811 Aligned with the ISO/IEC 9945-2:1993 standard.
- 20812 **Issue 6**
- 20813 This utility is now marked as part of the C-Language Development Utilities option.
- 20814 The obsolescent `-c` option is withdrawn in this issue.
- 20815 The normative text is reworded to avoid use of the term "must" for application requirements.

20816 **NAME**

20817 link — call *link()* function

20818 **SYNOPSIS**

20819 XSI link *file1 file2*

20820

20821 **DESCRIPTION**

20822 The *link* utility shall perform the function call:

20823 link(*file1*, *file2*);

20824 A user may need appropriate privilege to invoke the *link* utility.

20825 **OPTIONS**

20826 None.

20827 **OPERANDS**

20828 The following operands shall be supported:

20829 *file1* The path name of an existing file.

20830 *file2* The path name of the new directory entry to be created.

20831 **STDIN**

20832 Not used.

20833 **INPUT FILES**

20834 Not used.

20835 **ENVIRONMENT VARIABLES**

20836 The following environment variables shall affect the execution of *link*:

20837 *LANG* Provide a default value for the internationalization variables that are unset or null.
20838 If *LANG* is unset or null, the corresponding value from the implementation-
20839 defined default locale shall be used. If any of the internationalization variables
20840 contain an invalid setting, the utility behaves as if none of the variables had been
20841 set.

20842 *LC_ALL* If set to a non-empty string value, override the values of all the other
20843 internationalization variables.

20844 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
20845 characters (for example, single-byte as opposed to multi-byte characters in
20846 arguments).

20847 *LC_MESSAGES*

20848 Determine the locale that should be used to affect the format and contents of
20849 diagnostic messages written to standard error.

20850 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

20851 **ASYNCHRONOUS EVENTS**

20852 Default.

20853 **STDOUT**

20854 None.

20855 **STDERR**

20856 Used only for diagnostic messages.

20857 **OUTPUT FILES**

20858 None.

20859 **EXTENDED DESCRIPTION**

20860 None.

20861 **EXIT STATUS**

20862 The following exit values shall be returned:

20863 0 Successful completion.

20864 >0 An error occurred.

20865 **CONSEQUENCES OF ERRORS**

20866 Default.

20867 **APPLICATION USAGE**

20868 None.

20869 **EXAMPLES**

20870 None.

20871 **RATIONALE**

20872 None.

20873 **FUTURE DIRECTIONS**

20874 None.

20875 **SEE ALSO**20876 *In, unlink*, the System Interfaces volume of IEEE Std. 1003.1-200x, *link()*20877 **CHANGE HISTORY**

20878 First released in Issue 5.

20879 **NAME**20880 **ln** — link files20881 **SYNOPSIS**20882 **ln** [-fs] *source_file target_file*20883 **ln** [-fs] *source_file ... target_dir*20884 **DESCRIPTION**

20885 In the first synopsis form, the *ln* utility shall create a new directory entry (link), or if the **-s**
 20886 option is specified a symbolic link, for the file specified by the *source_file* operand, at the
 20887 *destination* path specified by the *target_file* operand. This first synopsis form shall be assumed
 20888 when the final operand does not name an existing directory; if more than two operands are
 20889 specified and the final is not an existing directory, an error shall result.

20890 In the second synopsis form, the *ln* utility shall create a new directory entry (link), or if the **-s**
 20891 option is specified a symbolic link, for each file specified by a *source_file* operand, at a *destination*
 20892 path in the existing directory named by *target_dir*.

20893 If the last operand specifies an existing file of a type not specified by the System Interfaces
 20894 volume of IEEE Std. 1003.1-200x, the behavior is implementation-defined.

20895 The corresponding *destination* path for each *source_file* shall be the concatenation of the target
 20896 directory path name, a slash character, and the last path name component of the *source_file*. The
 20897 second synopsis form shall be assumed when the final operand names an existing directory.

20898 For each *source_file*:

- 20899 1. If the *destination* path exists:
 - 20900 a. If the **-f** option is not specified, *ln* shall write a diagnostic message to standard error,
 20901 do nothing more with the current *source_file*, and go on to any remaining *source_files*.
 - 20902 b. Actions shall be performed equivalent to the *unlink()* function defined in the System
 20903 Interfaces volume of IEEE Std. 1003.1-200x, called using *destination* as the *path*
 20904 argument. If this fails for any reason, *ln* shall write a diagnostic message to standard
 20905 error, do nothing more with the current *source_file*, and go on to any remaining
 20906 *source_files*.
- 20907 2. If the **-s** option is specified, *ln* shall create a symbolic link named by the *destination* path
 20908 and containing as its path name *source_file*. The *ln* utility shall do nothing more with
 20909 *source_file* and shall go on to any remaining files.
- 20910 3. If *source_file* is a symbolic link, actions shall be performed equivalent to the *link()* function
 20911 using the object that *source_file* references as the *path1* argument and the destination path
 20912 as the *path2* argument. The *ln* utility shall do nothing more with *source_file* and shall go on
 20913 to any remaining files.
- 20914 4. Actions shall be performed equivalent to the *link()* function defined in the System
 20915 Interfaces volume of IEEE Std. 1003.1-200x using *source_file* as the *path1* argument, and the
 20916 *destination* path as the *path2* argument.

20917 **OPTIONS**

20918 The *ln* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
 20919 Utility Syntax Guidelines.

20920 The following option shall be supported:

20921 **-f** Force existing *destination* path names to be removed to allow the link.

- 20922 **-s** Create symbolic links instead of hard links.
- 20923 **OPERANDS**
- 20924 The following operands shall be supported:
- 20925 *source_file* A path name of a file to be linked. If the **-s** option is specified, no restrictions on the type of file or on its existence shall be made. If the **-s** option is not specified, whether a directory can be linked is implementation-defined.
- 20926
- 20927
- 20928 *target_file* The path name of the new directory entry to be created.
- 20929 *target_dir* A path name of an existing directory in which the new directory entries are created.
- 20930
- 20931 **STDIN**
- 20932 Not used.
- 20933 **INPUT FILES**
- 20934 None.
- 20935 **ENVIRONMENT VARIABLES**
- 20936 The following environment variables shall affect the execution of *ln*:
- 20937 *LANG* Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-defined default locale shall be used. If any of the internationalization variables contains an invalid setting, the utility shall behave as if none of the variables had been defined.
- 20938
- 20939
- 20940
- 20941
- 20942 *LC_ALL* If set to a non-empty string value, override the values of all the other internationalization variables.
- 20943
- 20944 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
- 20945
- 20946
- 20947 *LC_MESSAGES*
- 20948 Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
- 20949
- 20950 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 20951 **ASYNCHRONOUS EVENTS**
- 20952 Default.
- 20953 **STDOUT**
- 20954 Not used.
- 20955 **STDERR**
- 20956 Used only for diagnostic messages.
- 20957 **OUTPUT FILES**
- 20958 None.
- 20959 **EXTENDED DESCRIPTION**
- 20960 None.
- 20961 **EXIT STATUS**
- 20962 The following exit values shall be returned:
- 20963 0 All the specified files were linked successfully.

20964 >0 An error occurred.

20965 **CONSEQUENCES OF ERRORS**

20966 Default.

20967 **APPLICATION USAGE**

20968 None.

20969 **EXAMPLES**

20970 None.

20971 **RATIONALE**

20972 Some historic versions of *ln* (including the one specified by the SVID, unlink the destination file, if it exists, by default. If the mode does not permit writing, these versions prompt for confirmation before attempting the unlink. In these versions the `-f` option causes *ln* not to attempt to prompt for confirmation.

20976 This allows *ln* to succeed in creating links when the target file already exists, even if the file itself is not writable (although the directory must be). Early proposals specified this functionality.

20978 This volume of IEEE Std. 1003.1-200x does not allow the *ln* utility to unlink existing destination paths by default for the following reasons:

- 20980 • The *ln* utility has historically been used to provide locking for shell applications, a usage that is incompatible with *ln* unlinking the destination path by default. There was no corresponding technical advantage to adding this functionality.
- 20981 • This functionality gave *ln* the ability to destroy the link structure of files, which changes the historical behavior of *ln*.
- 20982 • This functionality is easily replicated with a combination of *rm* and *ln*.
- 20983 • It is not historical practice in many systems; BSD and BSD-derived systems do not support this behavior. Unfortunately, whichever behavior is selected can cause scripts written expecting the other behavior to fail.
- 20984 • It is preferable that *ln* perform in the same manner as the *link()* function, which does not permit the target to exist already.

20985 This volume of IEEE Std. 1003.1-200x retains the `-f` option to provide support for shell scripts depending on the SVID semantics. It seems likely that shell scripts would not be written to handle prompting by *ln* and would therefore have specified the `-f` option.

20991 The `-f` option is an undocumented feature of many historical versions of the *ln* utility, allowing linking to directories. These versions require modification.

20996 Early proposals of this volume of IEEE Std. 1003.1-200x also required an `-i` option, which behaved like the `-i` options in *cp* and *mv*, prompting for confirmation before unlinking existing files. This was not historical practice for the *ln* utility and has been omitted.

20999 **FUTURE DIRECTIONS**

21000 None.

21001 **SEE ALSO**

21002 *chmod*, *find*, *pax*, *rm*, the System Interfaces volume of IEEE Std. 1003.1-200x, *link()*

21003 **CHANGE HISTORY**

21004 First released in Issue 2.

21005 **Issue 4**

21006 Aligned with the ISO/IEC 9945-2:1993 standard.

21007 **Issue 6**21008 The *ln* utility is updated to include symbolic link processing as defined in the IEEE P1003.2b
21009 draft standard.

21010 **NAME**

21011 locale — get locale-specific information

21012 **SYNOPSIS**

21013 locale [-a | -m]

21014 locale [-ck] *name*...21015 **DESCRIPTION**

21016 The *locale* utility shall write information about the current locale environment, or all public
 21017 locales, to the standard output. For the purposes of this section, a *public locale* is one provided by
 21018 the implementation that is accessible to the application.

21019 When *locale* is invoked without any arguments, it shall summarize the current locale
 21020 environment for each locale category as determined by the settings of the environment variables
 21021 defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 7, Locale.

21022 When invoked with operands, it shall write values that have been assigned to the keywords in
 21023 the locale categories, as follows:

- 21024 • Specifying a keyword name shall select the named keyword and the category containing that
 21025 keyword.
- 21026 • Specifying a category name shall select the named category and all keywords in that
 21027 category.

21028 **OPTIONS**

21029 The *locale* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 21030 12.2, Utility Syntax Guidelines.

21031 The following options shall be supported:

- 21032 **-a** Write information about all available public locales. The available locales shall
 21033 include **POSIX**, representing the POSIX locale. The manner in which the
 21034 implementation determines what other locales are available is implementation-
 21035 defined.
- 21036 **-c** Write the names of selected locale categories; see the **STDOUT** section. The **-c**
 21037 option increases readability when more than one category is selected (for example,
 21038 via more than one keyword name or via a category name). It is valid both with
 21039 and without the **-k** option.
- 21040 **-k** Write the names and values of selected keywords. The implementation may omit
 21041 values for some keywords; see the **OPERANDS** section.
- 21042 **-m** Write names of available charmaps; see the Base Definitions volume of
 21043 IEEE Std. 1003.1-200x, Section 6.1, Portable Character Set.

21044 **OPERANDS**

21045 The following operand shall be supported:

- 21046 *name* The name of a locale category as defined in the Base Definitions volume of
 21047 IEEE Std. 1003.1-200x, Chapter 7, Locale, the name of a keyword in a locale
 21048 category, or the reserved name **charmap**. The named category or keyword shall be
 21049 selected for output. If a single *name* represents both a locale category name and a
 21050 keyword name in the current locale, the results are unspecified. Otherwise, both
 21051 category and keyword names can be specified as *name* operands, in any sequence.
 21052 It is implementation-defined whether any keyword values are written for the
 21053 categories *LC_CTYPE* and *LC_COLLATE*.

21054 **STDIN**

21055 Not used.

21056 **INPUT FILES**

21057 None.

21058 **ENVIRONMENT VARIABLES**21059 The following environment variables shall affect the execution of *locale*:

21060 *LANG* Provide a default value for the internationalization variables that are unset or null.
 21061 If *LANG* is unset or null, the corresponding value from the implementation-
 21062 defined default locale shall be used. If any of the internationalization variables
 21063 contains an invalid setting, the utility shall behave as if none of the variables had
 21064 been defined.

21065 *LC_ALL* If set to a non-empty string value, override the values of all the other
 21066 internationalization variables.

21067 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 21068 characters (for example, single-byte as opposed to multi-byte characters in
 21069 arguments and input files).

21070 *LC_MESSAGES*

21071 Determine the locale that should be used to affect the format and contents of
 21072 diagnostic messages written to standard error.

21073 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

21074 XSI The application shall ensure that the *LANG*, *LC_**, and *NLSPATH* environment variables specify
 21075 the current locale environment to be written out; they shall be used if the *-a* option is not
 21076 specified.

21077 **ASYNCHRONOUS EVENTS**

21078 Default.

21079 **STDOUT**

21080 If *locale* is invoked without any options or operands, the names and values of the *LANG* and
 21081 *LC_** environment variables described in this volume of IEEE Std. 1003.1-200x shall be written to
 21082 the standard output, one variable per line, with *LANG* first, and each line using the following
 21083 format. Only those variables set in the environment and not overridden by *LC_ALL* shall be
 21084 written using this format:

21085 "%s=%s\n", <variable_name>, <value>

21086 The names of those *LC_** variables associated with locale categories defined in this volume of
 21087 IEEE Std. 1003.1-200x that are not set in the environment or are overridden by *LC_ALL* shall be
 21088 written in the following format:

21089 "%s=\""%s\""\n", <variable_name>, <implied value>

21090 The <implied value> shall be the name of the locale that has been selected for that category by the
 21091 implementation, based on the values in *LANG* and *LC_ALL*, as described in the Base Definitions
 21092 volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables.

21093 The <value> and <implied value> shown above shall be properly quoted for possible later reentry
 21094 to the shell. The <value> shall not be quoted using double-quotes (so that it can be distinguished
 21095 by the user from the <implied value> case, which always requires double-quotes).

21096 The *LC_ALL* variable shall be written last, using the first format shown above. If it is not set, it
 21097 shall be written as:

21098 "LC_ALL=\n"

21099 If any arguments are specified:

21100 1. If the **-a** option is specified, the names of all the public locales shall be written, each in the
21101 following format:

21102 "%s\n", <locale name>

21103 2. If the **-c** option is specified, the names of all selected categories shall be written, each in the
21104 following format:

21105 "%s\n", <category name>

21106 If keywords are also selected for writing (see following items), the category name output
21107 shall precede the keyword output for that category.

21108 If the **-c** option is not specified, the names of the categories shall not be written; only the
21109 keywords, as selected by the <name> operand, shall be written.

21110 3. If the **-k** option is specified, the names and values of selected keywords shall be written. If
21111 a value is non-numeric, it shall be written in the following format:

21112 "%s=\"%s\"\\n", <keyword name>, <keyword value>

21113 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the
21114 *localedef* **-f** option when the locale was created shall be written, with the word **charmap** as
21115 <keyword name>.

21116 If a value is numeric, it shall be written in one of the following formats:

21117 "%s=%d\n", <keyword name>, <keyword value>

21118 "%s=%c%o\n", <keyword name>, <escape character>, <keyword value>

21119 "%s=%cx%x\n", <keyword name>, <escape character>, <keyword value>

21120 where the <escape character> is that identified by the **escape_char** keyword in the current
21121 locale; see the Base Definitions volume of IEEE Std. 1003.1-200x, Section 7.3, Locale
21122 Definition.

21123 Compound keyword values (list entries) shall be separated in the output by semicolons.
21124 When included in keyword values, the semicolon, the double-quote, the backslash, and
21125 any control character shall be preceded (escaped) with the escape character.

21126 4. If the **-k** option is not specified, selected keyword values shall be written, each in the
21127 following format:

21128 "%s\n", <keyword value>

21129 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the
21130 *localedef* **-f** option when the locale was created shall be written.

21131 5. If the **-m** option is specified, then a list of all available charmaps shall be written, each in
21132 the format:

21133 "%s\n", <charmap>

21134 where <charmap> is in a format suitable for use as the option-argument to the *localedef* **-f**
21135 option.

21136 **STDERR**

21137 Used only for diagnostic messages.

21138 **OUTPUT FILES**

21139 None.

21140 **EXTENDED DESCRIPTION**

21141 None.

21142 **EXIT STATUS**

21143 The following exit values shall be returned:

21144 0 All the requested information was found and output successfully.

21145 >0 An error occurred.

21146 **CONSEQUENCES OF ERRORS**

21147 Default.

21148 **APPLICATION USAGE**

21149 If the *LANG* environment variable is not set or set to an empty value, or one of the *LC_**
21150 environment variables is set to an unrecognized value, the actual locales assumed (if any) are
21151 implementation-defined as described in the Base Definitions volume of IEEE Std. 1003.1-200x,
21152 Chapter 8, Environment Variables.

21153 Implementations are not required to write out the actual values for keywords in the categories
21154 *LC_CTYPE* and *LC_COLLATE*; however, they must write out the categories (allowing an
21155 application to determine, for example, which character classes are available).

21156 **EXAMPLES**

21157 In the following examples, the assumption is that locale environment variables are set as
21158 follows:

21159 LANG=locale_x

21160 LC_COLLATE=locale_y

21161 The command *locale* would result in the following output:

21162 LANG=locale_x

21163 LC_CTYPE="locale_x"

21164 LC_COLLATE=locale_y

21165 LC_TIME="locale_x"

21166 LC_NUMERIC="locale_x"

21167 LC_MONETARY="locale_x"

21168 LC_MESSAGES="locale_x"

21169 LC_ALL=

21170 The order of presentation of the categories is not specified by this volume of
21171 IEEE Std. 1003.1-200x.

21172 The command:

21173 LC_ALL=POSIX locale -ck decimal_point

21174 would produce:

21175 LC_NUMERIC

21176 decimal_point="."

21177 The following command shows an application of *locale* to determine whether a user-supplied
21178 response is affirmative:


```
21179     if printf "%s\n" "$response" | grep -Eq "$(locale yesexpr)"
21180     then
21181         affirmative processing goes here
21182     else
21183         non-affirmative processing goes here
21184     fi
```

21185 RATIONALE

21186 The output for categories *LC_CTYPE* and *LC_COLLATE* has been made implementation-defined |
21187 because there is a questionable value in having a shell script receive an entire array of characters. |
21188 It is also difficult to return a logical collation description, short of returning a complete *localedef*
21189 source.

21190 The **-m** option was included to allow applications to query for the existence of charmaps. The
21191 output is a list of the charmaps (implementation-supplied and user-supplied, if any) on the
21192 system.

21193 The **-c** option was included for readability when more than one category is selected (for
21194 example, via more than one keyword name or via a category name). It is valid both with and
21195 without the **-k** option.

21196 The **charmap** keyword, which returns the name of the charmap (if any) that was used when the
21197 current locale was created, was included to allow applications needing the information to
21198 retrieve it.

21199 FUTURE DIRECTIONS

21200 None.

21201 SEE ALSO

21202 *localedef*, the Base Definitions volume of IEEE Std. 1003.1-200x, Section 7.3, Locale Definition |

21203 CHANGE HISTORY

21204 First released in Issue 4.

21205 Issue 5

21206 FUTURE DIRECTIONS section added.

21207 Issue 6

21208 The normative text is reworded to avoid use of the term “must” for application requirements.

21209 NAME

21210 localedef — define locale environment

21211 SYNOPSIS

21212 localedef [-c][-f *charmap*][-i *sourcefile*][-u *code_set_name*] *name*

21213 DESCRIPTION

21214 The *localedef* utility shall convert source definitions for locale categories into a format usable by
 21215 the functions and utilities whose operational behavior is determined by the setting of the locale
 21216 environment variables defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter
 21217 7, Locale. It is implementation-defined whether users have the capability to create new locales,
 21218 in addition to those supplied by the implementation. If the symbolic constant
 21219 POSIX2_LOCALEDEF is defined, the system supports the creation of new locales. On XSI-
 21220 conformant systems, the symbolic constant POSIX2_LOCALEDEF shall be defined.

21221 The utility shall read source definitions for one or more locale categories belonging to the same
 21222 locale from the file named in the *-i* option (if specified) or from standard input.

21223 The *name* operand identifies the target locale. The utility shall support the creation of *public*, or
 21224 generally accessible locales, as well as *private*, or restricted-access locales. Implementations may
 21225 restrict the capability to create or modify public locales to users with the appropriate privileges.

21226 Each category source definition shall be identified by the corresponding environment variable
 21227 name and terminated by an **END** *category-name* statement. The following categories shall be
 21228 supported. In addition, the input may contain source for implementation-defined categories.

21229 LC_CTYPE Defines character classification and case conversion.

21230 LC_COLLATE

21231 Defines collation rules.

21232 LC_MONETARY

21233 Defines the format and symbols used in formatting of monetary information.

21234 LC_NUMERIC

21235 Defines the decimal delimiter, grouping, and grouping symbol for non-monetary
 21236 numeric editing.

21237 LC_TIME Defines the format and content of date and time information.

21238 LC_MESSAGES

21239 Defines the format and values of affirmative and negative responses.

21240 OPTIONS

21241 The *localedef* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x,
 21242 Section 12.2, Utility Syntax Guidelines.

21243 The following options shall be supported:

21244 **-c** Create permanent output even if warning messages have been issued.

21245 **-f** *charmap* Specify the path name of a file containing a mapping of character symbols and
 21246 collating element symbols to actual character encodings. The format of the
 21247 *charmap* is described under the Base Definitions volume of IEEE Std. 1003.1-200x,
 21248 Section 6.4, Character Set Description File. The application shall ensure that this
 21249 option is specified if symbolic names (other than collating symbols defined in a
 21250 **collating-symbol** keyword) are used. If the *-f* option is not present, an
 21251 implementation-defined character mapping shall be used.

21252 **-i** *inputfile* The path name of a file containing the source definitions. If this option is not
 21253 present, source definitions shall be read from standard input. The format of the
 21254 *inputfile* is described in the Base Definitions volume of IEEE Std. 1003.1-200x,
 21255 Section 7.3, Locale Definition.

21256 **-u** *code_set_name* Specify the name of a codeset used as the target mapping of character symbols
 21257 and collating element symbols whose encoding values are defined in terms of the
 21258 ISO/IEC 10646-1: 1993 standard position constant values.

21259 OPERANDS

21260 The following operand shall be supported:

21261 *name* Identifies the locale; see the Base Definitions volume of IEEE Std. 1003.1-200x,
 21262 Chapter 7, Locale for a description of the use of this name. If the name contains one
 21263 or more slash characters, *name* shall be interpreted as a path name where the
 21264 created locale definitions shall be stored. If *name* does not contain any slash
 21265 characters, the interpretation of the name is implementation-defined and the locale
 21266 shall be public. This capability may be restricted to users with appropriate
 21267 privileges. (As a consequence of specifying one *name*, although several categories
 21268 can be processed in one execution, only categories belonging to the same locale can
 21269 be processed.)

21270 STDIN

21271 Unless the **-i** option is specified, the standard input shall be a text file containing one or more
 21272 locale category source definitions, as described in the Base Definitions volume of
 21273 IEEE Std. 1003.1-200x, Section 7.3, Locale Definition. When lines are continued using the escape
 21274 character mechanism, there is no limit to the length of the accumulated continued line.

21275 INPUT FILES

21276 The character set mapping file specified as the *charmap* option-argument is described under the
 21277 Base Definitions volume of IEEE Std. 1003.1-200x, Section 6.4, Character Set Description File. If a
 21278 locale category source definition contains a **copy** statement, as defined in the Base Definitions
 21279 volume of IEEE Std. 1003.1-200x, Chapter 7, Locale, and the **copy** statement names a valid,
 21280 existing locale, then *localedef* shall behave as if the source definition had contained a valid
 21281 category source definition for the named locale.

21282 ENVIRONMENT VARIABLES

21283 The following environment variables shall affect the execution of *localedef*:

21284 **LANG** Provide a default value for the internationalization variables that are unset or null.
 21285 If **LANG** is unset or null, the corresponding value from the implementation-
 21286 defined default locale shall be used. If any of the internationalization variables
 21287 contains an invalid setting, the utility shall behave as if none of the variables had
 21288 been defined.

21289 **LC_ALL** If set to a non-empty string value, override the values of all the other
 21290 internationalization variables.

21291 **LC_COLLATE**

21292 (This variable has no affect on *localedef*; the POSIX locale is used for this category.)

21293 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 21294 characters (for example, single-byte as opposed to multi-byte characters in
 21295 arguments and input files). This variable has no affect on the processing of *localedef*
 21296 input data; the POSIX locale is used for this purpose, regardless of the value of this
 21297 variable.

- 21298 *LC_MESSAGES*
 21299 Determine the locale that should be used to affect the format and contents of
 21300 diagnostic messages written to standard error.
- 21301 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 21302 **ASYNCHRONOUS EVENTS**
 21303 Default.
- 21304 **STDOUT**
 21305 The utility shall report all categories successfully processed, in an unspecified format.
- 21306 **STDERR**
 21307 Used only for diagnostic messages.
- 21308 **OUTPUT FILES**
 21309 The format of the created output is unspecified. If the *name* operand does not contain a slash, the
 21310 existence of an output file for the locale is unspecified.
- 21311 **EXTENDED DESCRIPTION**
 21312 When the *-u* option is used, the *code_set_name* option-argument shall be interpreted as an
 21313 implementation-defined name of a codeset to which the ISO/IEC 10646-1:1993 standard
 21314 position constant values shall be converted via an implementation-defined method. Both the
 21315 ISO/IEC 10646-1:1993 standard position constant values and other formats (decimal,
 21316 hexadecimal, or octal) shall be valid as encoding values within the *charmap* file. The codeset
 21317 represented by the implementation-defined name can be any codeset that is supported by the
 21318 implementation.
- 21319 When conflicts occur between the *charmap* specification of *<code_set_name>*, *<mb_cur_max>*, or
 21320 *<mb_cur_min>* and the implementation-defined interpretation of these respective items for the
 21321 codeset represented by the *-u* option-argument *code_set_name*, the result is unspecified.
- 21322 When conflicts occur between the *charmap* encoding values specified for symbolic names of
 21323 characters of the portable character set and the implementation-defined assignment of character
 21324 encoding values, the result is unspecified.
- 21325 If a non-printable character in the *charmap* has a width specified that is not *-1*, *localedef* shall
 21326 generate a warning.
- 21327 **EXIT STATUS**
 21328 The following exit values shall be returned:
- 21329 0 No errors occurred and the locales were successfully created.
 21330 1 Warnings occurred and the locales were successfully created.
 21331 2 The locale specification exceeded implementation limits or the coded character set or sets
 21332 used were not supported by the implementation, and no locale was created.
 21333 3 The capability to create new locales is not supported by the implementation.
 21334 >3 Warnings or errors occurred and no output was created.
- 21335 **CONSEQUENCES OF ERRORS**
 21336 If an error is detected, no permanent output shall be created.
- 21337 If warnings occur, permanent output shall be created if the *-c* option was specified. The
 21338 following conditions shall cause warning messages to be issued:
- 21339 • If a symbolic name not found in the *charmap* file is used for the descriptions of the *LC_CTYPE*
 21340 or *LC_COLLATE* categories (for other categories, this shall be an error condition).

- 21341 • If the number of operands to the **order** keyword exceeds the {COLL_WEIGHTS_MAX} limit.
- 21342 • If optional keywords not supported by the implementation are present in the source.
- 21343 • If a non-printable character has a width specified other than -1.
- 21344 Other implementation-defined conditions may also cause warnings.

21345 APPLICATION USAGE

21346 The *charmap* definition is optional, and is contained outside the locale definition. This allows
21347 both completely self-defined source files, and generic sources (applicable to more than one
21348 codeset). To aid portability, all *charmap* definitions must use the same symbolic names for the
21349 portable character set. As explained in the Base Definitions volume of IEEE Std. 1003.1-200x,
21350 Section 6.4, Character Set Description File, it is implementation-defined whether or not users or
21351 applications can provide additional character set description files. Therefore, the **-f** option might
21352 be operable only when an implementation-defined *charmap* is named.

21353 EXAMPLES

21354 None.

21355 RATIONALE

21356 The output produced by the *localedef* utility is implementation-defined. The *name* operand is
21357 used to identify the specific locale. (As a consequence, although several categories can be
21358 processed in one execution, only categories belonging to the same locale can be processed.)

21359 FUTURE DIRECTIONS

21360 None.

21361 SEE ALSO

21362 *locale*, the Base Definitions volume of IEEE Std. 1003.1-200x, Section 7.3, Locale Definition

21363 CHANGE HISTORY

21364 First released in Issue 4.

21365 Issue 6

21366 The **-u** option is added, as specified in the IEEE P1003.2b draft standard.

21367 The normative text is reworded to avoid use of the term “must” for application requirements.

21368 **NAME**

21369 logger — log messages

21370 **SYNOPSIS**21371 logger *string* ...21372 **DESCRIPTION**

21373 The *logger* utility saves a message, in an unspecified manner and format, containing the *string*
 21374 operands provided by the user. The messages are expected to be evaluated later by personnel
 21375 performing system administration tasks.

21376 It is implementation-defined whether messages written in locales other than the POSIX locale
 21377 are effective.

21378 **OPTIONS**

21379 None.

21380 **OPERANDS**

21381 The following operand shall be supported:

21382 *string* One of the string arguments whose contents are concatenated together, in the
 21383 order specified, separated by single <space> characters.

21384 **STDIN**

21385 Not used.

21386 **INPUT FILES**

21387 None.

21388 **ENVIRONMENT VARIABLES**21389 The following environment variables shall affect the execution of *logger*:

21390 *LANG* Provide a default value for the internationalization variables that are unset or null.
 21391 If *LANG* is unset or null, the corresponding value from the implementation-
 21392 defined default locale shall be used. If any of the internationalization variables
 21393 contains an invalid setting, the utility shall behave as if none of the variables had
 21394 been defined.

21395 *LC_ALL* If set to a non-empty string value, override the values of all the other
 21396 internationalization variables.

21397 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 21398 characters (for example, single-byte as opposed to multi-byte characters in
 21399 arguments).

21400 *LC_MESSAGES*

21401 Determine the locale that should be used to affect the format and contents of
 21402 diagnostic messages written to standard error. (This means diagnostics from *logger*
 21403 to the user or application, not diagnostic messages that the user is sending to the
 21404 system administrator.)

21405 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

21406 **ASYNCHRONOUS EVENTS**

21407 Default.

21408 **STDOUT**

21409 Not used.

21410 **STDERR**

21411 Used only for diagnostic messages.

21412 **OUTPUT FILES**

21413 Unspecified.

21414 **EXTENDED DESCRIPTION**

21415 None.

21416 **EXIT STATUS**

21417 The following exit values shall be returned:

21418 0 Successful completion.

21419 >0 An error occurred.

21420 **CONSEQUENCES OF ERRORS**

21421 Default.

21422 **APPLICATION USAGE**

21423 This utility allows logging of information for later use by a system administrator or programmer
21424 in determining why non-interactive utilities have failed. The locations of the saved messages,
21425 their format, and retention period are all unspecified. There is no method for a portable
21426 application to read messages, once written.

21427 **EXAMPLES**

21428 A batch application, running non-interactively, tries to read a configuration file and fails; it may
21429 attempt to notify the system administrator with:

21430 logger myname: unable to read file foo. [timestamp]

21431 **RATIONALE**

21432 The standard developers believed strongly that some method of alerting administrators to errors
21433 was necessary. The obvious example is a batch utility, running non-interactively, that is unable
21434 to read its configuration files or that is unable to create or write its results file. However, the
21435 standard developers did not wish to define the format or delivery mechanisms as they have
21436 historically been (and will probably continue to be) very system-specific, as well as involving
21437 functionality clearly outside of the scope of this volume of IEEE Std. 1003.1-200x.

21438 The text with *LC_MESSAGES* about diagnostic messages means diagnostics from *logger* to the
21439 user or application, not diagnostic messages that the user is sending to the system administrator.

21440 Multiple *string* arguments are allowed, similar to *echo*, for ease-of-use.

21441 Like the utilities *mailx* and *lp*, *logger* is admittedly difficult to test. This was not deemed sufficient
21442 justification to exclude these utilities from this volume of IEEE Std. 1003.1-200x. It is also
21443 arguable that they are, in fact, testable, but that the tests themselves are not portable.

21444 **FUTURE DIRECTIONS**

21445 None.

21446 **SEE ALSO**

21447 *mailx*, *write*

21448 **CHANGE HISTORY**

21449 First released in Issue 4.

21450 **NAME**

21451 logname — return the user's login name

21452 **SYNOPSIS**

21453 logname

21454 **DESCRIPTION**

21455 The *logname* utility shall write the user's login name to standard output. The login name shall be
21456 the string that would be returned by the *getlogin()* function defined in the System Interfaces
21457 volume of IEEE Std. 1003.1-200x. Under the conditions where the *getlogin()* function would fail,
21458 the *logname* utility shall write a diagnostic message to standard error and exit with a non-zero
21459 exit status.

21460 **OPTIONS**

21461 None.

21462 **OPERANDS**

21463 None.

21464 **STDIN**

21465 Not used.

21466 **INPUT FILES**

21467 None.

21468 **ENVIRONMENT VARIABLES**

21469 The following environment variables shall affect the execution of *logname*:

21470 *LANG* Provide a default value for the internationalization variables that are unset or null.
21471 If *LANG* is unset or null, the corresponding value from the implementation-
21472 defined default locale shall be used. If any of the internationalization variables
21473 contains an invalid setting, the utility shall behave as if none of the variables had
21474 been defined.

21475 *LC_ALL* If set to a non-empty string value, override the values of all the other
21476 internationalization variables.

21477 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
21478 characters (for example, single-byte as opposed to multi-byte characters in
21479 arguments).

21480 *LC_MESSAGES*

21481 Determine the locale that should be used to affect the format and contents of
21482 diagnostic messages written to standard error.

21483 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

21484 **ASYNCHRONOUS EVENTS**

21485 Default.

21486 **STDOUT**

21487 The *logname* utility output shall be a single line consisting of the user's login name:

21488 "%s\n", <login name>

21489 **STDERR**

21490 Used only for diagnostic messages.

21491 **OUTPUT FILES**

21492 None.

21493 **EXTENDED DESCRIPTION**

21494 None.

21495 **EXIT STATUS**

21496 The following exit values shall be returned:

21497 0 Successful completion.

21498 >0 An error occurred.

21499 **CONSEQUENCES OF ERRORS**

21500 Default.

21501 **APPLICATION USAGE**21502 The *logname* utility explicitly ignores the *LOGNAME* environment variable because environment changes could produce erroneous results.21504 **EXAMPLES**

21505 None.

21506 **RATIONALE**21507 The **passwd** file is not listed as required because the implementation may have other means of mapping login names.21509 **FUTURE DIRECTIONS**

21510 None.

21511 **SEE ALSO**21512 *id, who*21513 **CHANGE HISTORY**

21514 First released in Issue 2.

21515 **Issue 4**

21516 Aligned with the ISO/IEC 9945-2:1993 standard.

21517 NAME

21518 lp — send files to a printer

21519 SYNOPSIS

21520 lp [-c][-d *dest*][-n *copies*][-msw][-o *option*]... [-t *title*][*file*...]

21521 DESCRIPTION

21522 The *lp* utility shall copy the input files to an output destination in an unspecified manner. The
 21523 default output destination should be to a hardcopy device, such as a printer or microfilm
 21524 recorder, that produces non-volatile, human-readable documents. If such a device is not
 21525 available to the application, or if the system provides no such device, the *lp* utility shall exit with
 21526 a non-zero exit status.

21527 The actual writing to the output device may occur some time after the *lp* utility successfully
 21528 exits. During the portion of the writing that corresponds to each input file, the implementation
 21529 shall guarantee exclusive access to the device.

21530 The *lp* utility shall associate a unique *request ID* with each request.

21531 Normally, a banner page is produced to separate and identify each print job. This page may be
 21532 suppressed by implementation-defined conditions, such as an operator command or one of the
 21533 **-o** *option* values.

21534 OPTIONS

21535 The *lp* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
 21536 Utility Syntax Guidelines.

21537 The following options shall be supported:

21538 **-c** Exit only after further access to any of the input files is no longer required. The
 21539 application can then safely delete or modify the files without affecting the output
 21540 operation. Normally, files are not copied, but are linked whenever possible. If the
 21541 **-c** option is not given, then the user should be careful not to remove any of the
 21542 files before the request has been printed in its entirety. It should also be noted that
 21543 in the absence of the **-c** option, any changes made to the named files after the
 21544 request is made but before it is printed are reflected in the printed output. On some
 21545 systems, **-c** may be on by default.

21546 **-d** *dest* Specify a string that names the destination (*dest*). If *dest* is a printer, the request
 21547 shall be printed only on that specific printer. If *dest* is a class of printers, the request
 21548 shall be printed on the first available printer that is a member of the class. Under
 21549 certain conditions (printer unavailability, file space limitation, and so on), requests
 21550 for specific destinations need not be accepted. Destination names vary between
 21551 systems.

21552 If **-d** is not specified, and neither the *LPDEST* nor *PRINTER* environment variable
 21553 is set, an unspecified destination is used. The **-d** *dest* option shall take precedence
 21554 over *LPDEST*, which in turn shall take precedence over *PRINTER*. Results are
 21555 undefined when *dest* contains a value that is not a valid destination name.

21556 **-m** Send mail (see *mailx* (on page 2794)) after the files have been printed. By default,
 21557 no mail is sent upon normal completion of the print request.

21558 **-n** *copies* Write *copies* number of copies of the files, where *copies* is a positive decimal integer.
 21559 The methods for producing multiple copies and for arranging the multiple copies
 21560 when multiple *file* operands are used are unspecified, except that each file shall be
 21561 output as an integral whole, not interleaved with portions of other files.

21606 *PRINTER*. Results are undefined when **-d** is not specified, *LPDEST* is unset, and
 21607 *PRINTER* contains a value that is not a valid device or destination name.

21608 ASYNCHRONOUS EVENTS

21609 Default.

21610 STDOUT

21611 The *lp* utility shall write a *request ID* to the standard output, unless **-s** is specified. The format of
 21612 the message is unspecified. The request ID can be used on systems supporting the historical
 21613 *cancel* and *lpstat* utilities.

21614 STDERR

21615 Used only for diagnostic messages.

21616 OUTPUT FILES

21617 None.

21618 EXTENDED DESCRIPTION

21619 None.

21620 EXIT STATUS

21621 The following exit values shall be returned:

21622 0 All input files were processed successfully.

21623 >0 No output device was available, or an error occurred.

21624 CONSEQUENCES OF ERRORS

21625 Default.

21626 APPLICATION USAGE

21627 The *pr* and *fold* utilities can be used to achieve reasonable formatting for the implementation's
 21628 default page size.

21629 A portable application can use one of the *file* operands only with the **-c** option or if the file is
 21630 publicly readable and guaranteed to be available at the time of printing. This is because the
 21631 standard gives the implementation the freedom to queue up the request for printing at some
 21632 later time by a different process that might not be able to access the file.

21633 EXAMPLES

21634 1. To print file *file*:

21635 `lp -c file`

21636 2. To print multiple files with headers:

21637 `pr file1 file2 | lp`

21638 RATIONALE

21639 The *lp* utility was designed to be a basic version of a utility that is already available in many
 21640 historical implementations. The standard developers considered that it should be implementable
 21641 simply as:

21642 `cat "$@" > /dev/lp`

21643 after appropriate processing of options, if that is how the implementation chose to do it and if
 21644 exclusive access could be granted (so that two users did not write to the device simultaneously).
 21645 Although in the future the standard developers may add other options to this utility, it should
 21646 always be able to execute with no options or operands and send the standard input to an
 21647 unspecified output device.

21648 This volume of IEEE Std. 1003.1-200x makes no representations concerning the format of the
 21649 printed output, except that it must be “human-readable” and “non-volatile”. Thus, writing by
 21650 default to a disk or tape drive or a display terminal would not qualify. (Such destinations are not
 21651 prohibited when `-d dest`, `LPDEST`, or `PRINTER` are used, however.)

21652 This volume of IEEE Std. 1003.1-200x is worded such that a “print job” consisting of multiple
 21653 input files, possibly in multiple copies, is guaranteed to print so that any one file is not
 21654 intermixed with another, but there is no statement that all the files or copies have to print out
 21655 together.

21656 The `-c` option may imply a spooling operation, but this is not required. The utility can be
 21657 implemented to wait until the printer is ready and then wait until it is finished. Because of that,
 21658 there is no attempt to define a queuing mechanism (priorities, classes of output, and so on).

21659 On some historical systems, the request ID reported on the `STDOUT` can be used to later cancel
 21660 or find the status of a request using utilities not defined in this volume of IEEE Std. 1003.1-200x.

21661 Although the historical System V `lp` and BSD `lpr` utilities have provided similar functionality,
 21662 they used different names for the environment variable specifying the destination printer. Since
 21663 the name of the utility here is `lp`, `LPDEST` (used by the System V `lp` utility) was given precedence
 21664 over `PRINTER` (used by the BSD `lpr` utility). Since environments of users frequently contain one
 21665 or the other environment variable, the `lp` utility is required to recognize both. If this was not
 21666 done, many applications would send output to unexpected output devices when users moved
 21667 from system to system.

21668 Some have commented that `lp` has far too little functionality to make it worthwhile. Requests
 21669 have proposed additional options or operands or both that added functionality. The requests
 21670 included:

- 21671 • Wording *requiring* the output to be “hardcopy”
- 21672 • A requirement for multiple printers
- 21673 • Options for supporting various page-description languages

21674 Given that a compliant system is not required to even have a printer, placing further restrictions
 21675 upon the behavior of the printer is not useful. Since hardcopy format is so application-
 21676 dependent, it is difficult, if not impossible, to select a reasonable subset of functionality that
 21677 should be required on all compliant systems.

21678 The term “unspecified” is used in this section in lieu of “implementation-defined” as most
 21679 known implementations would not be able to make definitive statements in their conformance
 21680 documents: the existence and usage of printers is very dependent on how the system
 21681 administrator configures each individual system.

21682 Since the default destination, device type, queuing mechanisms, and acceptable forms of input
 21683 are all unspecified, usage guidelines for what a portable application can do are as follows:

- 21684 • Use the command in a pipeline, or with `-c`, so that there are no permission problems and the
 21685 files can be safely deleted or modified.
- 21686 • Limit output to text files of reasonable line lengths and printable characters and include no
 21687 device-specific formatting information, such as a page description language. The meaning of
 21688 “reasonable” in this context can only be answered as a quality-of-implementation issue, but
 21689 it should be apparent from historical usage patterns in the industry and the locale. The `pr` and
 21690 `fold` utilities can be used to achieve reasonable formatting for the default page size of the
 21691 implementation.

21692 Alternatively, the application can arrange its installation in such a way that it requires the
 21693 system administrator or operator to provide the appropriate information on *lp* options and
 21694 environment variable values.

21695 At a minimum, having this utility in this volume of IEEE Std. 1003.1-200x tells the industry that
 21696 portable applications require a means to print output and provides at least a command name
 21697 and *LPDEST* routing mechanism that can be used for discussions between vendors, application
 21698 writers, and users. The use of “should” in the DESCRIPTION of *lp* clearly shows the intent of
 21699 the standard developers, even if they cannot mandate that all systems (such as laptops) have
 21700 printers.

21701 This volume of IEEE Std. 1003.1-200x does not specify what the ownership of the process
 21702 performing the writing to the output device may be. If *-c* is not used, it is unspecified whether
 21703 the process performing the writing to the output device has permission to read *file* if there are
 21704 any restrictions in place on who may read *file* until after it is printed. Also, if *-c* is not used, the
 21705 results of deleting *file* before it is printed are unspecified.

21706 FUTURE DIRECTIONS

21707 None.

21708 SEE ALSO

21709 *mailx*

21710 CHANGE HISTORY

21711 First released in Issue 2.

21712 Issue 4

21713 Aligned with the ISO/IEC 9945-2: 1993 standard.

21714 Issue 6

21715 The following new requirements on POSIX implementations derive from alignment with the
 21716 Single UNIX Specification:

- 21717 • In the DESCRIPTION, the requirement to associate a unique request ID, and the normal
 21718 generation of a banner page is added.
- 21719 • In the OPTIONS section:
 - 21720 — The *-d dest* description is expanded, but references to *lpstat* are removed.
 - 21721 — The *-m*, *-o*, *-s*, *-t*, and *-w* options are added.
- 21722 • In the ENVIRONMENT VARIABLES section, *LC_TIME* may now affect the execution.
- 21723 • The STDOUT section is added.

21724 The normative text is reworded to avoid use of the term “must” for application requirements.

21725 **NAME**21726 `ls` — list directory contents21727 **SYNOPSIS**21728 xSI `ls [-CFRacdilqrutl][-H | -L][-fgmnoptsx][file...]`21729 **DESCRIPTION**

21730 For each operand that names a file of a type other than directory or symbolic link to a directory,
 21731 *ls* shall write the name of the file as well as any requested, associated information. For each
 21732 operand that names a file of type directory, *ls* shall write the names of files contained within the
 21733 directory as well as any requested, associated information. If one of the `-d`, `-F`, or `-l` options are
 21734 specified, and one of the `-H` or `-L` options are not specified, for each operand that names a file of
 21735 type symbolic link to a directory, *ls* shall write the name of the file as well as any requested,
 21736 associated information. If none of the `-d`, `-F`, or `-l` options are specified, or the `-H` or `-L` options
 21737 are specified, for each operand that names a file of type symbolic link to a directory, *ls* shall write
 21738 the names of files contained within the directory as well as any requested, associated
 21739 information.

21740 If no operands are specified, *ls* shall write the contents of the current directory. If more than one
 21741 operand is specified, *ls* shall write non-directory operands first; it shall sort directory and non-
 21742 directory operands separately according to the collating sequence in the current locale.

21743 The *ls* utility shall detect infinite loops; that is, entering a previously visited directory that is an
 21744 ancestor of the last file encountered. When it detects an infinite loop, *ls* shall write a diagnostic
 21745 message to standard error and shall either recover its position in the hierarchy or terminate.

21746 **OPTIONS**

21747 The *ls* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
 21748 Utility Syntax Guidelines.

21749 The following options shall be supported:

21750 **-C** Write multi-text-column output with entries sorted down the columns, according
 21751 to the collating sequence. The number of text columns and the column separator
 21752 characters are unspecified, but should be adapted to the nature of the output
 21753 device.

21754 **-F** Do not follow symbolic links named as operands unless the `-H` or `-L` options are
 21755 specified. Write a slash (`/'`) immediately after each path name that is a directory,
 21756 an asterisk (`'*'`) after each that is executable, a vertical bar (`'|'`) after each that is
 21757 a FIFO, and an at sign (`'@'`) after each that is a symbolic link. For other file types,
 21758 other symbols may be written.

21759 **-H** If a symbolic link referencing a file of type directory is specified on the command
 21760 line, *ls* shall evaluate the file information and file type to be those of the file
 21761 referenced by the link, and not the link itself; however, *ls* shall write the name of
 21762 the link itself and not the file referenced by the link.

21763 **-L** Evaluate the file information and file type for all symbolic links (whether named
 21764 on the command line or encountered in a file hierarchy) to be those of the file
 21765 referenced by the link, and not the link itself; however, *ls* shall write the name of
 21766 the link itself and not the file referenced by the link. When `-L` is used with `-l`, write
 21767 the contents of symbolic links in the long format (see the STDOUT section).

21768 **-R** Recursively list subdirectories encountered.

21769 **-a** Write out all directory entries, including those whose names begin with a period
 21770 (`'.'`). Entries beginning with a period shall not be written out unless explicitly

21771		referenced, the -a option is supplied, or an implementation-defined condition shall
21772		cause them to be written.
21773	-c	Use time of last modification of the file status information (see <sys/stat.h> in the
21774		System Interfaces volume of IEEE Std. 1003.1-200x) instead of last modification of
21775		the file itself for sorting (-t) or writing (-l).
21776	-d	Do not follow symbolic links named as operands unless the -H or -L options are
21777		specified. Do not treat directories differently than other types of files. The use of -d
21778		with -R produces unspecified results.
21779 XSI	-f	Force each argument to be interpreted as a directory and list the name found in
21780		each slot. This option shall turn off -l , -t , -s , and -r , and shall turn on -a ; the order
21781		is the order in which entries appear in the directory.
21782 XSI	-g	The same as -l , except that the owner shall not be written.
21783	-i	For each file, write the file's file serial number (see <i>stat()</i> in the System Interfaces
21784		volume of IEEE Std. 1003.1-200x).
21785	-l	(The letter ell.) Do not follow symbolic links named as operands unless the -H or
21786		-L options are specified. Write out in long format (see the STDOUT section). When
21787		-l (ell) is specified, -1 (one) shall be assumed.
21788 XSI	-m	Stream output format; list files across the page, separated by commas.
21789 XSI	-n	The same as -l , except that the owner's UID and GID numbers are written, rather
21790		than the associated character strings.
21791 XSI	-o	The same as -l , except that the group is not written.
21792 XSI	-p	Write a slash (/) after each file name if that file is a directory.
21793	-q	Force each instance of non-printable file name characters and <tab> characters to
21794		be written as the question-mark (' ? ') character. Implementations may provide
21795		this option by default if the output is to a terminal device.
21796	-r	Reverse the order of the sort to get reverse collating sequence or oldest first.
21797 XSI	-s	Indicate the total number of file system blocks consumed by each file displayed.
21798		The block size is implementation-defined.
21799	-t	Sort by time modified (most recently modified first) before sorting the operands by
21800		the collating sequence.
21801	-u	Use time of last access (see <sys/stat.h> in the System Interfaces volume of
21802		IEEE Std. 1003.1-200x) instead of last modification of the file for sorting (-t) or
21803		writing (-l).
21804 XSI	-x	The same as -C , except that the multi-text-column output is produced with entries
21805		sorted across, rather than down, the columns.
21806	-1	(The numeric digit one.) Force output to be one entry per line.
21807		Specifying more than one of the options in the following mutually exclusive pairs shall not be
21808 XSI		considered an error: -C and -l (ell), -m and -l (ell), -x and -l (ell), -C and -1 (one), -H and -L ,
21809		-c and -u . The last option specified in each pair shall determine the output format.

21810 **OPERANDS**

21811 The following operand shall be supported:

21812 *file* A path name of a file to be written. If the file specified is not found, a diagnostic
21813 message shall be output on standard error.

21814 **STDIN**

21815 Not used.

21816 **INPUT FILES**

21817 None.

21818 **ENVIRONMENT VARIABLES**

21819 The following environment variables shall affect the execution of *ls*:

21820 *COLUMNS* Determine the user's preferred column position width for writing multiple text-
21821 column output. If this variable contains a string representing a decimal integer, the
21822 *ls* utility shall calculate how many path name text columns to write (see *-C*) based
21823 on the width provided. If *COLUMNS* is not set or invalid, an implementation-
21824 defined number of column positions shall be assumed, based on the
21825 implementation's knowledge of the output device. The column width chosen to
21826 write the names of files in any given directory shall be constant. File names shall
21827 not be truncated to fit into the multiple text-column output.

21828 *LANG* Provide a default value for the internationalization variables that are unset or null.
21829 If *LANG* is unset or null, the corresponding value from the implementation-
21830 defined default locale shall be used. If any of the internationalization variables
21831 contains an invalid setting, the utility shall behave as if none of the variables had
21832 been defined.

21833 *LC_ALL* If set to a non-empty string value, override the values of all the other
21834 internationalization variables.

21835 *LC_COLLATE*

21836 Determine the locale for character collation information in determining the path
21837 name collation sequence.

21838 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
21839 characters (for example, single-byte as opposed to multi-byte characters in
21840 arguments) and which characters are defined as printable (character class **print**).

21841 *LC_MESSAGES*

21842 Determine the locale that should be used to affect the format and contents of
21843 diagnostic messages written to standard error.

21844 *LC_TIME* Determine the format and contents for date and time strings written by *ls*.

21845 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

21846 *TZ* Determine the timezone for date and time strings written by *ls*.

21847 **ASYNCHRONOUS EVENTS**

21848 Default.

21849 **STDOUT**

21850 The default format shall be to list one entry per line to standard output; the exceptions are to
21851 XSI terminals or when one of the *-C*, *-m*, or *-x* options is specified. If the output is to a terminal, the
21852 format is implementation-defined.

21853 XSI When **-m** is specified, the format used shall be:

21854 "%s, %s, ...\n", <filename1>, <filename2>

21855 where the largest number of file names shall be written without exceeding the length of the line.

21856 If the **-i** option is specified, the file's file serial number (see <sys/stat.h> in the System Interfaces
21857 volume of IEEE Std. 1003.1-200x) shall be written in the following format before any other
21858 output for the corresponding entry:

21859 %u ", <file serial number>

21860 If the **-l** option is specified without **-L**, the following information shall be written:

21861 "%s %u %s %s %u %s %s\n", <file mode>, <number of links>,
21862 <owner name>, <group name>, <number of bytes in the file>,
21863 <date and time>, <pathname>

21864 If the file is a symbolic link, this information shall be about the link itself and the <pathname>
21865 field shall be of the form:

21866 "%s -> %s", <pathname of link>, <contents of link>

21867 If both **-l** and **-L** are specified, the following information shall be written:

21868 "%s %u %s %s %u %s %s0, <file mode>, <number of links>,
21869 <owner name>, <group name>, <number of bytes in the file>,
21870 <date and time>, <pathname of link>

21871 where all fields except <pathname of link> shall be for the file resolved from the symbolic link.

21872 XSI The **-g**, **-n**, and **-o** options use the same format as **-l**, but with omitted items and their
21873 associated <blank> characters. See the OPTIONS section.

21874 XSI In both the preceding **-l** forms, If <owner name> or <group name> cannot be determined, or if **-n**
21875 is given, they shall be replaced with their associated numeric values using the format %u.

21876 The <date and time> field shall contain the appropriate date and timestamp of when the file was
21877 last modified. In the POSIX locale, the field shall be the equivalent of the output of the following
21878 date command:

21879 date "+%b %e %H:%M"

21880 if the file has been modified in the last six months, or:

21881 date "+%b %e %Y"

21882 (where two <space> characters are used between %e and %Y) if the file has not been modified in
21883 the last six months or if the modification date is in the future, except that, in both cases, the final
21884 <newline> character produced by date shall not be included and the output shall be as if the date
21885 command were executed at the time of the last modification date of the file rather than the
21886 current time. When the LC_TIME locale category is not set to the POSIX locale, a different format
21887 and order of presentation of this field may be used.

21888 If the file is a character special or block special file, the size of the file may be replaced with
21889 implementation-defined information associated with the device in question.

21890 If the path name was specified as a file operand, it shall be written as specified.

21891 XSI The file mode written under the **-l**, **-g**, **-n**, and **-o** options shall consist of the following format:

21892 "%c%s%s%c", <entry type>, <owner permissions>,
21893 <group permissions>, <other permissions>,

- 21894 <optional alternate access method flag>
- 21895 The <optional alternate access method flag> shall be a single <space> character if there is no
21896 alternate or additional access control method associated with the file; otherwise, a printable
21897 character shall be used.
- 21898 The <entry type> character shall describe the type of file, as follows:
- 21899 d Directory.
- 21900 b Block special file.
- 21901 c Character special file.
- 21902 l (ell) Symbolic link.
- 21903 p FIFO.
- 21904 - Regular file.
- 21905 Implementations may add other characters to this list to represent other implementation-defined
21906 file types.
- 21907 The next three fields shall be three characters each:
- 21908 <owner permissions>
- 21909 Permissions for the file owner class (see the Base Definitions volume of
21910 IEEE Std. 1003.1-200x, Section 4.1, File Access Permissions).
- 21911 <group permissions>
- 21912 Permissions for the file group class.
- 21913 <other permissions>
- 21914 Permissions for the file other class.
- 21915 Each field shall have three character positions:
- 21916 1. If 'r', the file is readable; if '-', the file is not readable.
- 21917 2. If 'w', the file is writable; if '-', the file is not writable.
- 21918 3. The first of the following that applies:
- 21919 S If in <owner permissions>, the file is not executable and set-user-ID mode is set. If in
21920 <group permissions>, the file is not executable and set-group-ID mode is set.
- 21921 s If in <owner permissions>, the file is executable and set-user-ID mode is set. If in
21922 <group permissions>, the file is executable and set-group-ID mode is set.
- 21923 x The file is executable or the directory is searchable.
- 21924 - None of the attributes of 'S', 's', or 'x' applies.
- 21925 Implementations may add other characters to this list for the third character position. Such
21926 additions shall, however, be written in lowercase if the file is executable or searchable, and
21927 in uppercase if it is not.
- 21928 XSI If any of the **-l**, **-g**, **-n**, **-o**, or **-s** options is specified, each list of files within the directory shall be
21929 preceded by a status line indicating the number of file system blocks occupied by files in the
21930 directory in 512-byte units, rounded up to the next integral number of units, if necessary. In the
21931 POSIX locale, the format shall be:
- 21932 "total %u\n", <number of units in the directory>

21933 If more than one directory, or a combination of non-directory files and directories are written,
21934 either as a result of specifying multiple operands, or the **-R** option, each list of files within a
21935 directory shall be preceded by:

21936 "\n%s:\n", <directory name>

21937 If this string is the first thing to be written, the first <newline> character shall not be written.
21938 This output shall precede the number of units in the directory.

21939 XSI If the **-s** option is given, each file shall be written with the number of blocks used by the file.
21940 Along with **-C**, **-l**, **-m**, or **-x**, the number and a <space> character shall precede the file name;
21941 with **-g**, **-l**, **-n**, or **-o**, they shall precede each line describing a file.

21942 **STDERR**

21943 Used only for diagnostic messages.

21944 **OUTPUT FILES**

21945 None.

21946 **EXTENDED DESCRIPTION**

21947 None.

21948 **EXIT STATUS**

21949 The following exit values shall be returned:

21950 0 Successful completion.

21951 >0 An error occurred.

21952 **CONSEQUENCES OF ERRORS**

21953 Default.

21954 **APPLICATION USAGE**

21955 Many implementations use the equal sign ('=') and the at sign('@') to denote sockets bound to
21956 the file system and symbolic links, respectively, for the **-F** option. Similarly, many historical
21957 implementations use the 's' character and the 'l' character to denote sockets and symbolic
21958 links, respectively, as the entry type characters for the **-l** option.

21959 It is difficult for an application to use every part of the file modes field of *ls -l* in a portable
21960 manner. Certain file types and executable bits are not guaranteed to be exactly as shown, as
21961 implementations may have extensions. Applications can use this field to pass directly to a user
21962 printout or prompt, but actions based on its contents should generally be deferred, instead, to
21963 the *test* utility.

21964 The output of *ls* (with the **-l** and related options) contains information that logically could be
21965 used by utilities such as *chmod* and *touch* to restore files to a known state. However, this
21966 information is presented in a format that cannot be used directly by those utilities or be easily
21967 translated into a format that can be used. A character has been added to the end of the
21968 permissions string so that applications at least have an indication that they may be working in
21969 an area they do not understand instead of assuming that they can translate the permissions
21970 string into something that can be used. Future issues or related documents may define one or
21971 more specific characters to be used based on different standard additional or alternative access
21972 control mechanisms.

21973 As with many of the utilities that deal with file names, the output of *ls* for multiple files or in one
21974 of the long listing formats must be used carefully on systems where file names can contain
21975 embedded white space. Systems and system administrators should institute policies and user
21976 training to limit the use of such file names.

21977 The number of disk blocks occupied by the file that it reports varies depending on underlying
 21978 file system type, block size units reported, and the method of calculating the number of blocks.
 21979 On some file system types, the number is the actual number of blocks occupied by the file
 21980 (counting indirect blocks and ignoring holes in the file); on others it is calculated based on the
 21981 file size (usually making an allowance for indirect blocks, but ignoring holes).

21982 EXAMPLES

21983 An example of a small directory tree being fully listed with `ls -laRF a` in the POSIX locale:

```
21984 total 11
21985 drwxr-xr-x  3 hlj   prog           64 Jul  4 12:07 ./
21986 drwxrwxrwx  4 hlj   prog        3264 Jul  4 12:09 ../
21987 drwxr-xr-x  2 hlj   prog           48 Jul  4 12:07 b/
21988 -rwxr--r--  1 hlj   prog          572 Jul  4 12:07 foo*

21989 a/b:
21990 total 4
21991 drwxr-xr-x  2 hlj   prog           48 Jul  4 12:07 ./
21992 drwxr-xr-x  3 hlj   prog           64 Jul  4 12:07 ../
21993 -rw-r--r--  1 hlj   prog          700 Jul  4 12:07 bar
```

21994 RATIONALE

21995 Some historical implementations of the `ls` utility show all entries in a directory except dot and
 21996 dot-dot when a superuser invokes `ls` without specifying the `-a` option. When “normal” users
 21997 invoke `ls` without specifying `-a`, they should not see information about any files with names
 21998 beginning with period unless they were named as *file* operands.

21999 Implementations are expected to traverse arbitrary depths when processing the `-R` option. The
 22000 only limitation on depth should be based on running out of physical storage for keeping track of
 22001 untraversed directories.

22002 The `-1` (one) option is currently found in BSD and BSD-derived implementations only. It is
 22003 required in this volume of IEEE Std. 1003.1-200x so that portable applications might ensure that
 22004 output is one entry per line, even if the output is to a terminal.

22005 Generally, this volume of IEEE Std. 1003.1-200x is silent about what happens when options are
 22006 given multiple times. In the cases of `-C`, `-l`, and `-1`, however, it does specify the results of these
 22007 overlapping options. Since `ls` is one of the most aliased commands, it is important that the
 22008 implementation perform intuitively. For example, if the alias were:

```
22009 alias ls="ls -C"
```

22010 and the user typed `ls -1`, single-text-column output should result, not an error.

22011 The BSD `ls` provides a `-A` option (like `-a`, but dot and dot-dot are not written out). The small
 22012 difference from `-a` did not seem important enough to require both.

22013 Implementations are allowed to make `-q` the default for terminals to prevent trojan horse
 22014 attacks on terminals with special escape sequences. This is not required because:

- 22015 • Some control characters may be useful on some terminals; for example, a system might write
 22016 them as `"\001"` or `"^A"`.

- 22017 • Special behavior for terminals is not relevant to application portability.

22018 An early proposal specified that the optional alternate access method flag had to be `'+'` if there
 22019 was an alternate access method used on the file or `<space>` if there was not. This was changed to
 22020 be `<space>` if there is not and a single printable character if there is. This was done for three
 22021 reasons:

- 22022 1. There are historical implementations using characters other than '+'.
- 22023 2. There are implementations that vary this character used in that position to distinguish
22024 between various alternate access methods in use.
- 22025 3. The standard developers did not want to preclude futures specifications that might need a
22026 way to specify more than one alternate access method.
- 22027 Nonetheless, implementations providing a single alternate access method are encouraged to use
22028 '+'.
- 22029 In an early proposal, the units used to specify the number of blocks occupied by files in a
22030 directory in an *ls -l* listing was implementation-defined. This was because BSD systems have
22031 historically used 1 024-byte units and System V systems have historically used 512-byte units. It
22032 was pointed out by BSD developers that their system has used 512-byte units in some places and
22033 1 024-byte units in other places. (System V has consistently used 512.) Therefore, this volume of
22034 IEEE Std. 1003.1-200x usually specifies 512. Future releases of BSD are expected to consistently
22035 provide 512 bytes as a default with a way of specifying 1 024-byte units where appropriate.
- 22036 The *<date and time>* field in the *-l* format is specified only for the POSIX locale. As noted, the
22037 format can be different in other locales. No mechanism for defining this is present in this volume
22038 of IEEE Std. 1003.1-200x, as the appropriate vehicle is a messaging system; that is, the format
22039 should be specified as a "message".
- 22040 **FUTURE DIRECTIONS**
- 22041 The *-s* uses implementation-defined units and cannot be used portably; it may be withdrawn in
22042 a future issue.
- 22043 **SEE ALSO**
- 22044 *chmod*, *find*, the System Interfaces volume of IEEE Std. 1003.1-200x, *<sys/stat.h>*
- 22045 **CHANGE HISTORY**
- 22046 First released in Issue 2.
- 22047 **Issue 4**
- 22048 Aligned with the ISO/IEC 9945-2: 1993 standard.
- 22049 **Issue 5**
- 22050 Second FUTURE DIRECTION added.
- 22051 **Issue 6**
- 22052 The following new requirements on POSIX implementations derive from alignment with the
22053 Single UNIX Specification:
- 22054 • In the *-F* option, other symbols are allowed for other file types.
- 22055 Treatment of symbolic links is added, as defined in the IEEE P1003.2b draft standard.

22056 **NAME**22057 m4 — macro processor (**DEVELOPMENT**)22058 **SYNOPSIS**22059 xSI m4 [-s][-D *name*[=*val*]]...[-U *name*]... *file*...

22060

22061 **DESCRIPTION**22062 The *m4* utility is a macro processor that shall read one or more text files, process them according
22063 to their included macro statements, and write the results to standard output.22064 **OPTIONS**22065 The *m4* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
22066 12.2, Utility Syntax Guidelines, except that the order of the **-D** and **-U** options shall be
22067 significant.

22068 The following options shall be supported:

22069 **-s** Enable line synchronization output for the *c99* preprocessor phase (that is, **#line**
22070 directives).22071 **-D *name*[=*val*]**
22072 Define *name* to *val* or to null if *=val* is omitted.22073 **-U *name*** Undefine *name*.22074 **OPERANDS**

22075 The following operand shall be supported:

22076 *file* A path name of a text file to be processed. If no *file* is given, or if it is '-', the
22077 standard input shall be read.22078 **STDIN**22079 The standard input shall be a text file that is used if no *file* operand is given, or if it is '-'.22080 **INPUT FILES**22081 The input file named by the *file* operand shall be a text file.22082 **ENVIRONMENT VARIABLES**22083 The following environment variables shall affect the execution of *m4*:22084 **LANG** Provide a default value for the internationalization variables that are unset or null.
22085 If **LANG** is unset or null, the corresponding value from the implementation-
22086 defined default locale shall be used. If any of the internationalization variables
22087 contains an invalid setting, the utility shall behave as if none of the variables had
22088 been defined.22089 **LC_ALL** If set to a non-empty string value, override the values of all the other
22090 internationalization variables.22091 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
22092 characters (for example, single-byte as opposed to multi-byte characters in
22093 arguments and input files).22094 **LC_MESSAGES**
22095 Determine the locale that should be used to affect the format and contents of
22096 diagnostic messages written to standard error.22097 **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

22098 **ASYNCHRONOUS EVENTS**

22099 Default.

22100 **STDOUT**22101 The standard output shall be the same as the input files, after being processed for macro
22102 expansion.22103 **STDERR**22104 Used to display strings with the **errprint** macro, macro tracing enabled by the **traceon** macro, the
22105 defined text for macros written by the **dumpdef** macro, or for diagnostic messages.22106 **OUTPUT FILES**

22107 None.

22108 **EXTENDED DESCRIPTION**22109 The *m4* utility shall compare each token from the input against the set of built-in and user-
22110 defined macros. If the token matches the name of a macro, then the token shall be replaced by
22111 the macros defining text, if any, and rescanned for matching macro names. Once no portion of
22112 the token matches the name of a macro, it shall be written to standard output. Macros may have
22113 arguments, in which case the arguments shall be substituted into the defining text before it is
22114 rescanned.

22115 Macro calls have the form:

22116 *name(arg1, arg2, ..., argn)*22117 Macro names shall consist of letters, digits, and underscores, where the first character is not a
22118 digit. Tokens not of this form shall not be treated as macro names.22119 The application shall ensure that the left parenthesis immediately follows the name of the
22120 macro. If a token matching the name of a macro is not followed by a left parenthesis, it is
22121 handled as a use of that macro without arguments.22122 If a macro name is followed by a left parenthesis, its arguments are the comma-separated tokens
22123 between the left parenthesis and the matching right parenthesis. Unquoted <blank> and
22124 <newline> characters preceding each argument shall be ignored. All other characters, including
22125 trailing <blank> and <newline> characters, are retained. Commas enclosed between left and
22126 right parenthesis characters do not delimit arguments.22127 Arguments are positionally defined and referenced. The string "\$1" in the defining text shall be
22128 replaced by the first argument. Systems shall support at least nine arguments; only the first nine
22129 can be referenced, using the strings "\$1" to "\$9", inclusive. The string "\$0" is replaced with
22130 the name of the macro. The string "\$#" is replaced by the number of arguments as a string. The
22131 string "\$*" is replaced by a list of all of the arguments, separated by commas. The string "\$@"
22132 is replaced by a list of all of the arguments separated by commas, and each argument is quoted
22133 using the current left and right quoting strings.22134 If fewer arguments are supplied than are in the macro definition, the omitted arguments are
22135 taken to be null. It is not an error if more arguments are supplied than are in the macro
22136 definition.22137 No special meaning is given to any characters enclosed between matching left and right quoting
22138 strings, but the quoting strings are themselves discarded. By default, the left quoting string
22139 consists of a grave accent (``) and the right quoting string consists of an acute accent (``) see
22140 also the **changequote** macro.22141 Comments are written but not scanned for matching macro names; by default, the begin-
22142 comment string consists of the number sign character and the end-comment string consists of a
22143 <newline> character. See also the **changecom** and **dnl** macros.

- 22144 The *m4* utility makes available the following built-in macros. They can be redefined, but once
 22145 this is done the original meaning is lost. Their values are null unless otherwise stated. In the
 22146 descriptions below, the term *defining text* refers to the value of the macro: the second argument
 22147 to the **define** macro, among other things. Except for the first argument to the **eval** macro, all
 22148 numeric built-in macro arguments shall be interpreted as decimal values. The string values
 22149 produced as the defining text of the **decr**, **divnum**, **incr**, **index**, **len**, and **sysval** built-in macros
 22150 shall be in the form of a decimal-constant as defined in the C language.
- 22151 **changecom** The **changecom** macro sets the begin-comment and end-comment strings. With no
 22152 arguments, the comment mechanism is disabled. With a single argument, that
 22153 argument becomes the begin-comment string and the <newline> character
 22154 becomes the end-comment string. With two arguments, the first argument
 22155 becomes the begin-comment string and the second argument becomes the end-
 22156 comment string. Systems support comment strings of at least five characters.
- 22157 **changequote** The **changequote** macro sets the begin-quote and end-quote strings. With no
 22158 arguments, the quote strings are set to the default values (that is, ` `). With a
 22159 single argument, that argument becomes the begin-quote string and the <newline>
 22160 character becomes the end-quote string. With two arguments, the first argument
 22161 becomes the begin-quote string and the second argument becomes the end-quote
 22162 string. Systems support quote strings of at least five characters.
- 22163 **decr** The defining text of the **decr** macro is its first argument decremented by 1. It is an
 22164 error to specify an argument containing any non-numeric characters.
- 22165 **define** The second argument is specified as the defining text of the macro whose name is
 22166 the first argument.
- 22167 **defn** The defining text of the **defn** macro is the quoted definition (using the current
 22168 quoting strings) of its arguments.
- 22169 **divert** The *m4* utility maintains ten temporary buffers, numbered 0 to 9, inclusive.

22170 *Notes to Reviewers*

- 22171 *This section with side shading will not appear in the final copy. - Ed.*
- 22172 Re D1, XCU, ERN 286: Buffer 0 seems strange: it's one of the 10 buffers, and thus
 22173 should be a diversion buffer, but at 19704 it implies that it's the name of the main
 22174 output. What is it (or are there really only 9 diversion buffers?) Also, see austin-
 22175 group mail sequence #295.
- 22176 When the last of the input has been processed, any output that has been placed in
 22177 these buffers is written to standard output in buffer-numerical order. The **divert**
 22178 macro diverts future output to the buffer specified by its argument. Specifying no
 22179 argument or an argument of 0 resumes the normal output process. Output
 22180 diverted to a stream other than 0 to 9 is discarded. It is an error to specify an
 22181 argument containing any non-numeric characters.
- 22182 **divnum** The defining text of the **divnum** macro is the number of the current output stream
 22183 as a string.
- 22184 **dnl** The **dnl** macro shall cause *m4* to discard all input characters up to and including
 22185 the next <newline> character.
- 22186 **dumpdef** The **dumpdef** macro writes the defined text to standard error for each of the
 22187 macros specified as arguments, or, if no arguments are specified, for all macros.

22188	errprint	The errprint macro writes its arguments to standard error.
22189	eval	The eval macro evaluates its first argument as an arithmetic expression, using 32-bit signed integer arithmetic. All of the C-language operators are supported, except for:
22190		
22191		
22192		[]
22193		->
22194		++
22195		--
22196		(<i>type</i>)
22197		unary *
22198		<i>sizeof</i>
22199		,
22200		.
22201		?:
22202		unary &
22203		and all assignment operators. It is an error to specify any of these operators.
22204		Precedence and associativity are as in C. Systems support octal and hexadecimal
22205		numbers as in C. The second argument, if specified, sets the radix for the result; the
22206		default is 10. The third argument, if specified, sets the minimum number of digits
22207		in the result. It is an error to specify the second or third argument containing any
22208		non-numeric characters.
22209	ifdef	If the first argument to the ifdef macro is defined, the defining text is the second
22210		argument. Otherwise, the defining text is the third argument, if specified, or the
22211		null string, if not.
22212	ifndef	If the first argument (or the defining text of the first argument if it is a macro
22213		name) to the ifndef macro is the same as the second argument (or the defining text
22214		of the second argument if it is a macro name), then the defining text is the third
22215		argument.
22216	Notes to Reviewers	
22217		<i>This section with side shading will not appear in the final copy. - Ed.</i>
22218		D1, XCU, ERN 287 (as modified by email #297) suggests the following replacement
22219		text for ifelse : "This function takes 3n+0 or 3n+1 arguments. For each group of 3
22220		arguments, if the first and second are the same, the result is the third of the group.
22221		If the strings are not equal, and no arguments remain, the defining text is null. If
22222		one argument remains, it becomes the defining text. If three or more arguments
22223		remain, the process is repeated with the new group of three arguments. If 3n+2
22224		arguments are provided, the evaluation proceeds as above, but a warning is
22225		generated and the last argument ignored.
22226		If there are more than four arguments, the initial comparison of the first and
22227		second arguments are repeated for each group of three arguments. If no match is
22228		found, the defining text is the argument following the last set of three compared;
22229		otherwise, it is null.
22230	include	The defining text for the include macro is the contents of the file named by the first
22231		argument. It is an error if the file cannot be read.
22232	incr	The defining text of the incr macro is its first argument incremented by 1. It is an
22233		error to specify an argument containing any non-numeric characters.

22234	index	The defining text of the index macro is the first character position (as a string) in the first argument where a string matching the second argument begins (zero origin), or -1 if the second argument does not occur.
22235		
22236		
22237	len	The defining text of the len macro is the length (as a string) of the first argument.
22238	m4exit	Exit from the <i>m4</i> utility. If the first argument is specified, it is the exit code. The default is zero. It is an error to specify an argument containing any non-numeric characters.
22239		
22240		
22241	m4wrap	The first argument is processed when EOF is reached. If the m4wrap macro is used multiple times, the arguments specified are processed in the order in which the m4wrap macros were processed.
22242		
22243		
22244	maketemp	The defining text is the first argument, with any trailing 'x' characters replaced with the current process ID as a string.
22245		
22246	popdef	The popdef macro deletes the current definition of its arguments, replacing that definition with the previous one. If there is no previous definition, the macro is undefined.
22247		
22248		
22249	pushdef	The pushdef macro is identical to the define macro with the exception that it preserves any current definition for future retrieval using the popdef macro.
22250		
22251	shift	The defining text for the shift macro is all of its arguments except for the first one.
22252	sinclude	The sinclude macro is identical to the include macro, except that it is not an error if the file is inaccessible.
22253		
22254	substr	The defining text for the substr macro is the substring of the first argument beginning at the zero-offset character position specified by the second argument. The third argument, if specified, is the number of characters to select; if not specified, the characters from the starting point to the end of the first argument become the defining text. It is not an error to specify a starting point beyond the end of the first argument and the defining text is null. It is an error to specify an argument containing any non-numeric characters.
22255		
22256		
22257		
22258		
22259		
22260		
22261	syscmd	The syscmd macro interprets its first argument as a shell command line. The defining text is the string result of that command. No output redirection is performed by the <i>m4</i> utility. The exit status value from the command can be retrieved using the sysval macro.
22262		
22263		
22264		
22265	sysval	The defining text of the sysval macro is the exit value of the utility last invoked by the syscmd macro (as a string).
22266		
22267	traceon	The traceon macro enables tracing for the macros specified as arguments, or, if no arguments are specified, for all macros. The trace output is written to standard error in an unspecified format.
22268		
22269		
22270	traceoff	The traceoff macro disables tracing for the macros specified as arguments, or, if no arguments are specified, for all macros.
22271		
22272	translit	The defining text of the translit macro is the first argument with every character that occurs in the second argument replaced with the corresponding character from the third argument.
22273		
22274		
22275	undefine	The undefine macro deletes all definitions (including those preserved using the pushdef macro) of the macros named by its arguments.
22276		

22277 **undivert** The **undivert** macro shall cause immediate output of any text in temporary buffers
 22278 named as arguments, or all temporary buffers if no arguments are specified.
 22279 Buffers can be undiverted into other temporary buffers. Undiverting discards the
 22280 contents of the temporary buffer. It is an error to specify an argument containing
 22281 any non-numeric characters.

22282 **EXIT STATUS**

22283 The following exit values shall be returned:

22284 0 Successful completion.

22285 >0 An error occurred

22286 If the **m4exit** macro is used, the exit value can be specified by the input file.

22287 **CONSEQUENCES OF ERRORS**

22288 Default.

22289 **APPLICATION USAGE**

22290 The **defn** macro is useful for renaming macros, especially built-ins.

22291 **EXAMPLES**

22292 An example of a single *m4* input file capable of generating two output files follows. The file
 22293 **file1.m4** could contain lines such as:

22294 if(VER, 1, *do_something*)

22295 if(VER, 2, *do_something*)

22296 The makefile for the program might include:

22297 file1.1.c : file1.m4

22298 m4 -D VER=1 file1.m4 > file1.1.c

22299 ...

22300 file1.2.c : file1.m4

22301 m4 -D VER=2 file1.m4 > file1.2.c

22302 ...

22303 The **-U** option can be used to undefine **VER**. If **file1.m4** contains:

22304 if(VER, 1, *do_something*)

22305 if(VER, 2, *do_something*)

22306 ifndef(VER, *do_something*)

22307 then the makefile would contain:

22308 file1.0.c : file1.m4

22309 m4 -U VER file1.m4 > file1.0.c

22310 ...

22311 file1.1.c : file1.m4

22312 m4 -D VER=1 file1.m4 > file1.1.c

22313 ...

22314 file1.2.c : file1.m4

22315 m4 -D VER=2 file1.m4 > file1.2.c

22316 ...

22317 **RATIONALE**

22318 None.

22319 **FUTURE DIRECTIONS**

22320 None.

22321 **SEE ALSO**22322 *c99*22323 **CHANGE HISTORY**

22324 First released in Issue 2.

22325 **Issue 4**

22326 Format reorganized.

22327 Utility Syntax Guideline support mandated.

22328 Internationalized environment variable support mandated.

22329 **Issue 5**

22330 The phrase “the defined text for macros written by the **dumpdef** macro” is added to the
22331 description of **STDERR**, and the description of **dumpdef** is updated to indicate that output is
22332 written to standard error. The description of **eval** is updated to indicate that the list of excluded
22333 C operators excludes unary **&** and **.**. In the description of **ifdef**, the phrase “and it is not
22334 defined to be zero” is deleted.

22335 **Issue 6**

22336 In the **EXTENDED DESCRIPTION**, the **eval** text is updated to include a **&** character in the
22337 excepted list.

22338 The normative text is reworded to avoid use of the term “must” for application requirements.

22339 The Open Group Base Resolution bwg2000-006 is applied.

22340 NAME

22341 mailx — process messages

22342 SYNOPSIS

22343 **Send Mode**22344 mailx [-s *subject*] *address...*22345 **Receive Mode**

22346 mailx -e

22347 mailx [-HiNn][-F][-u *user*]22348 mailx -f[-HiNn][-F][*file*]

22349 DESCRIPTION

22350 The *mailx* utility provides a message sending and receiving facility. It has two major modes,
 22351 selected by the options used: Send Mode and Receive Mode.

22352 On systems that do not support the User Portability Utilities option, an application using *mailx*
 22353 shall have the ability to send messages in an unspecified manner (Send Mode). Unless the first
 22354 character of one or more lines is tilde ('~'), all characters in the input message shall appear in
 22355 the delivered message, but additional characters may be inserted in the message before it is
 22356 retrieved.

22357 On systems supporting the User Portability Utilities option, mail-receiving capabilities and other
 22358 interactive features, Receive Mode, described below, also shall be enabled.

22359 **Send Mode**

22360 Send Mode can be used by applications or users to send messages from the text in standard
 22361 input.

22362 **Receive Mode**

22363 Receive Mode is more oriented to interactive users. Mail can be read and sent in this interactive
 22364 mode.

22365 When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to
 22366 messages. When sending mail, *mailx* allows editing, reviewing, and other modification of the
 22367 message as it is entered.

22368 Incoming mail shall be stored in one or more unspecified locations for each user, collectively
 22369 called the system *mailbox* for that user. When *mailx* is invoked in Receive Mode, the system
 22370 mailbox shall be the default place to find new mail. As messages are read, they shall be marked
 22371 to be moved to a secondary file for storage, unless specific action is taken. This secondary file is
 22372 called the **mbox** and is normally located in the directory referred to by the *HOME* environment
 22373 variable (see *MBOX* in the ENVIRONMENT VARIABLES section for a description of this file).
 22374 Messages shall remain in this file until explicitly removed. When the **-f** option is used to read
 22375 mail messages from secondary files, messages shall be retained in those files unless specifically
 22376 removed. All three of these locations—system mailbox, **mbox**, and secondary file—are referred
 22377 to in this section as simply “mailboxes”, unless more specific identification is required.

22378 **OPTIONS**

22379 The *mailx* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
22380 12.2, Utility Syntax Guidelines.

22381 The following options shall be supported. (Only the **-s subject** option shall be required on all
22382 systems. The other options are required only on systems supporting the User Portability Utilities
22383 option.)

22384 **-e** Test for the presence of mail in the system mailbox. The *mailx* utility shall write
22385 nothing and exit with a successful return code if there is mail to read.

22386 **-f** Read messages from the file named by the *file* operand instead of the system
22387 mailbox. (See also **folder**.) If no *file* operand is specified, read messages from the
22388 **mbox** instead of the system mailbox.

22389 **-F** Record the message in a file named after the first recipient. The name is the login-
22390 name portion of the address found first on the **To:** line in the mail header.
22391 Overrides the **record** variable, if set (see **Internal Variables in mailx** (on page
22392 2801).)

22393 **-H** Write a header summary only.

22394 **-i** Ignore interrupts. (See also **ignore**).

22395 **-n** Do not initialize from the system default start-up file. See the EXTENDED
22396 DESCRIPTION section.

22397 **-N** Do not write an initial header summary.

22398 **-s subject** Set the **Subject** header field to *subject*. All characters in the *subject* string shall
22399 appear in the delivered message. The results are unspecified if *subject* is longer
22400 than {LINE_MAX} – 10 bytes or contains a <newline> character.

22401 **-u user** Read the system mailbox of the login name *user*. This shall only be successful if
22402 the invoking user has the appropriate privileges to read the system mailbox of that
22403 user.

22404 **OPERANDS**

22405 The following operands shall be supported:

22406 *address* Addressee of message. When **-n** is specified and no user start-up files are accessed
22407 (see the EXTENDED DESCRIPTION section), the user or application shall ensure
22408 this is an address to pass to the mail delivery system. Any system or user start-up
22409 files may enable aliases (see **alias** under **Commands in mailx** (on page 2804)) that
22410 may modify the form of *address* before it is passed to the mail delivery system.

22411 *file* A path name of a file to be read instead of the system mailbox when **-f** is specified.
22412 The meaning of the *file* option-argument shall be affected by the contents of the
22413 **folder** internal variable; see **Internal Variables in mailx** (on page 2801).

22414 **STDIN**

22415 When *mailx* is invoked in Send Mode (the first synopsis line), standard input shall be the
22416 message to be delivered to the specified addresses. When in Receive Mode, user commands are
22417 accepted from *stdin*. If the User Portability Utilities option is not supported, standard input lines
22418 beginning with a tilde ('~') character produce unspecified results.

22419 If the User Portability Utilities option is supported, then in both Send and Receive Modes,
22420 standard input lines beginning with the escape character (usually tilde ('~')) affect processing
22421 as described in **Command Escapes in mailx** (on page 2812).

22422 INPUT FILES

22423 When *mailx* is used as described by this volume of IEEE Std. 1003.1-200x, the *file* option-
 22424 argument (see the *-f* option) and the **mbox** shall be text files containing mail messages,
 22425 formatted as described in the OUTPUT FILES section. The nature of the system mailbox is
 22426 unspecified; it need not be a file.

22427 ENVIRONMENT VARIABLES

22428 The following environment variables shall affect the execution of *mailx*:

22429 **DEAD** Determine the path name of the file in which to save partial messages in case of
 22430 interrupts or delivery errors. The default shall be **dead.letter** in the directory
 22431 named by the *HOME* variable. The behavior of *mailx* in saving partial messages is
 22432 unspecified if the User Portability Utilities option is not supported and *DEAD* is
 22433 not defined with the value **/dev/null**.

22434 **EDITOR** Determine the name of a utility to invoke when the **edit** (see **Commands in mailx**
 22435 (on page 2804)) or **~e** (see **Command Escapes in mailx** (on page 2812)) command is
 22436 XSI used. The default editor is unspecified. On XSI-conformant systems it is *ed*. The
 22437 effects of this variable are unspecified if the User Portability Utilities option is not
 22438 supported.

22439 **HOME** Determine the path name of the user's home directory.

22440 **LANG** Provide a default value for the internationalization variables that are unset or null.
 22441 If *LANG* is unset or null, the corresponding value from the implementation-
 22442 defined default locale shall be used. If any of the internationalization variables
 22443 contains an invalid setting, the utility shall behave as if none of the variables had
 22444 been defined.

22445 **LC_ALL** If set to a non-empty string value, override the values of all the other
 22446 internationalization variables.

22447 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 22448 characters (for example, single-byte as opposed to multi-byte characters in
 22449 arguments and input files) and the handling of case-insensitive address and
 22450 header-field comparisons.

22451 **LC_TIME** Determine the format and contents of the date and time strings written by *mailx*.

22452 LC_MESSAGES

22453 Determine the locale that should be used to affect the format and contents of
 22454 diagnostic messages written to standard error and informative messages written to
 22455 standard output.

22456 **LISTER** Determine a string representing the command for writing the contents of the
 22457 **folder** directory to standard output when the **folders** command is given (see
 22458 **folders** in **Commands in mailx** (on page 2804)). Any string acceptable as a
 22459 *command_string* operand to the *sh -c* command shall be valid. If this variable is null
 22460 or not set, the output command shall be *ls*. The effects of this variable are
 22461 unspecified if the User Portability Utilities option is not supported.

22462 **MAILRC** Determine the path name of the start-up file. The default shall be **.mailrc** in the
 22463 directory referred to by the *HOME* environment variable. The behavior of *mailx* is
 22464 unspecified if the User Portability Utilities option is not supported and *MAILRC* is
 22465 not defined with the value **/dev/null**.

22466 **MBOX** Determine a path name of the file to save messages from the system mailbox that
 22467 have been read. The **exit** command shall override this function, as shall saving the

- 22468 message explicitly in another file. The default shall be **mbx** in the directory
 22469 named by the *HOME* variable. The effects of this variable are unspecified if the
 22470 User Portability Utilities option is not supported.
- 22471 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 22472 **PAGER** Determine a string representing an output filtering or pagination command for
 22473 writing the output to the terminal. Any string acceptable as a *command_string*
 22474 operand to the *sh -c* command shall be valid. When standard output is a terminal
 22475 device, the message output shall be piped through the command if the *mailx*
 22476 internal variable **cr** is set to a value less the number of lines in the message; see
 22477 **Internal Variables in mailx** (on page 2801). If the *PAGER* variable is null or not
 22478 set, the paginator shall be either *more* or another paginator utility documented in
 22479 the system documentation. The effects of this variable are unspecified if the User
 22480 Portability Utilities option is not supported.
- 22481 **SHELL** Determine the name of a preferred command interpreter. The default shall be *sh*.
 22482 The effects of this variable are unspecified if the User Portability Utilities option is
 22483 not supported.
- 22484 **TERM** Determine the name of the terminal type, to indicate in an unspecified manner, if
 22485 the internal variable **screen** is not specified, the number of lines in a screenful of
 22486 headers. If *TERM* is not set or is set to null, an unspecified default terminal type
 22487 shall be used and the value of a screenful is unspecified. The effects of this variable
 22488 are unspecified if the User Portability Utilities option is not supported.
- 22489 **VISUAL** Determine a path name of a utility to invoke when the **visual** command (see
 22490 **Commands in mailx** (on page 2804)) or *~v* command-escape (see **Command**
 22491 **Escapes in mailx** (on page 2812)) is used. If this variable is null or not set, the full-
 22492 screen editor shall be *vi*. The effects of this variable are unspecified if the User
 22493 Portability Utilities option is not supported.
- 22494 **ASYNCHRONOUS EVENTS**
- 22495 When *mailx* is in Send Mode and standard input is not a terminal, it shall take the standard
 22496 action for all signals.
- 22497 In Receive Mode, or in Send Mode when standard input is a terminal, if a SIGINT signal is
 22498 received:
- 22499 1. If in command mode, the current command, if there is one, shall be aborted, and a
 22500 command-mode prompt shall be written.
 - 22501 2. If in input mode:
 - 22502 a. If **ignore** is set, *mailx* shall write "@\n", discard the current input line, and continue
 22503 processing, bypassing the message-abort mechanism described in item 2b.
 - 22504 b. If the interrupt was received while sending mail, either when in Receive Mode or in
 22505 Send Mode, a message shall be written, and another subsequent interrupt, with no
 22506 other intervening characters typed, shall be required to abort the mail message. If in
 22507 Receive Mode and another interrupt is received, a command-mode prompt shall be
 22508 written. If in Send Mode and another interrupt is received, *mailx* shall terminate with
 22509 a non-zero status.
- 22510 In both cases listed in item b, if the message is not empty:
- 22511 i. If **save** is enabled and the file named by *DEAD* can be created, the message
 22512 shall be written to the file named by *DEAD*. If the file exists, the message shall
 22513 be written to replace the contents of the file.

22514 ii. If **save** is not enabled, or the file named by *DEAD* cannot be created, the
22515 message shall not be saved.

22516 The *mailx* utility shall take the standard action for all other signals.

22517 **STDOUT**

22518 In command and input modes, all output, including prompts and messages, shall be written to
22519 standard output.

22520 **STDERR**

22521 Used only for diagnostic messages.

22522 **OUTPUT FILES**

22523 Various *mailx* commands and command escapes can create or add to files, including the **mbox**,
22524 the dead-letter file, and secondary mailboxes. When *mailx* is used as described in this volume of
22525 IEEE Std. 1003.1-200x, these files shall be text files, formatted as follows:

22526 line beginning with **From**<space>
22527 [one or more *header-lines*; see **Commands in mailx** (on page 2804)]
22528 *empty line*
22529 [zero or more *body lines*
22530 *empty line*]
22531 [line beginning with **From**<space>...]

22532 where each message begins with the **From** <space> line shown, preceded by the beginning of
22533 the file or an empty line. (The **From** <space> line is considered to be part of the message header,
22534 but not one of the header-lines referred to in **Commands in mailx** (on page 2804); thus, it shall
22535 not be affected by the **discard**, **ignore**, or **retain** commands.) The formats of the remainder of the
22536 **From** <space> line and any additional header lines are unspecified, except that none shall be
22537 empty. The format of a message body line is also unspecified, except that no line following an
22538 empty line shall start with **From** <space>; *mailx* shall modify any such user-entered message
22539 body lines (following an empty line and beginning with **From** <space>) by adding one or more
22540 characters to precede the 'F'; it may add these characters to **From** <space> lines that are not
22541 preceded by an empty line.

22542 When a message from the system mailbox or entered by the user is not a text file, it is
22543 implementation-defined how such a message is stored in files written by *mailx*.

22544 **EXTENDED DESCRIPTION**

22545 The entire EXTENDED DESCRIPTION section shall apply only to implementations supporting
22546 the User Portability Utilities option.

22547 The *mailx* utility cannot guarantee support for all character encodings in all circumstances. For
22548 example, inter-system mail may be restricted to 7-bit data by the underlying network, 8-bit data
22549 need not be portable to non-internationalized systems, and so on. Under these circumstances, it
22550 is recommended that only characters defined in the ISO/IEC 646:1991 standard International
22551 Reference Version (equivalent to ASCII) 7-bit range of characters be used.

22552 When *mailx* is invoked using one of the Receive Mode synopsis forms, it shall write a page of
22553 header-summary lines (if **-N** was not specified and there are messages, see below), followed by
22554 a prompt indicating that *mailx* can accept regular commands (see **Commands in mailx** (on page
22555 2804)); this is termed *command mode*. The page of header-summary lines shall contain the first
22556 new message if there are new messages, or the first unread message if there are unread
22557 messages, or the first message. When *mailx* is invoked using the Send Mode synopsis and
22558 standard input is a terminal, if no subject is specified on the command line and the **asksub**
22559 variable is set, a prompt for the subject shall be written. At this point, *mailx* is in input mode.
22560 This input mode is also entered when using one of the Receive Mode synopsis forms and a reply

22561 or new message is composed using the **reply**, **Reply**, **followup**, **Followup**, or **mail** commands
 22562 and standard input is a terminal. When the message is typed and the end of message is
 22563 encountered, the message shall be passed to the mail delivery software. Commands can be
 22564 entered by beginning a line with the escape character (by default, tilde ('~')) followed by a
 22565 single command letter and optional arguments. See **Commands in mailx** (on page 2804) for a
 22566 summary of these commands. It is unspecified what effect these commands will have if
 22567 standard input is not a terminal when a message is entered using either the Send Mode synopsis,
 22568 or the Read Mode commands **reply**, **Reply**, **followup**, **Followup**, or **mail**.

22569 **Note:** For notational convenience, this section uses the default escape character, tilde, in all
 22570 references and examples.

22571 At any time, the behavior of *mailx* shall be governed by a set of environmental and internal
 22572 variables. These are flags and valued parameters that can be set and cleared via the *mailx set*
 22573 and *unset* commands.

22574 Regular commands are of the form:

22575 [*command*] [*msglist*] [*argument ...*]

22576 If no *command* is specified in command mode, **next** shall be assumed. In input mode, commands
 22577 shall be recognized by the escape character, and lines not treated as commands shall be taken as
 22578 input for the message.

22579 In command mode, each message shall be assigned a sequential number, starting with 1.

22580 All messages have a state that affects how they are displayed in the header summary and how
 22581 they are retained or deleted upon termination of *mailx*. There is at any time the notion of a
 22582 *current* message, marked by a '>' at the beginning of a line in the header summary. When *mailx*
 22583 is invoked using one of the Receive Mode synopsis forms, the current message shall be the first
 22584 new message, if there is a new message, or the first unread message if there is an unread
 22585 message, or the first message if there are any messages, or unspecified if there are no messages
 22586 in the mailbox. Each command that takes an optional list of messages (*msglist*) or an optional
 22587 single message (*message*) on which to operate shall leave the current message set to the highest-
 22588 numbered message of the messages specified, unless the command deletes messages, in which
 22589 case the current message shall be set to the first undeleted message (that is, a message not in the
 22590 deleted state) after the highest-numbered message deleted by the command, if one exists, or the
 22591 first undeleted message before the highest-numbered message deleted by the command, if one
 22592 exists, or to an unspecified value if there are no remaining undeleted messages. All messages are
 22593 in one of the following states:

22594 *new* The message is present in the system mailbox and has not been viewed by the user
 22595 or moved to any other state. Messages in state *new* when *mailx* quits shall be
 22596 retained in the system mailbox.

22597 *unread* The message has been present in the system mailbox for more than one invocation
 22598 of *mailx* and has not been viewed by the user or moved to any other state.
 22599 Messages in state *unread* when *mailx* quits shall be retained in the system mailbox.

22600 *read* The message has been processed by one of the following commands: **~f**, **~m**, **~F**, **~M**,
 22601 **copy**, **mbox**, **next**, **pipe**, **print**, **Print**, **top**, **type**, **Type**, **undelete**. The **delete**, **dp**, and
 22602 **dt** commands may also cause the next message to be marked as *read*, depending on
 22603 the value of the **autoprint** variable. Messages that are in the system mailbox and in
 22604 state *read* when *mailx* quits shall be saved in the **mbox**, unless the internal variable
 22605 **hold** was set. Messages that are in the **mbox** or in a secondary mailbox and in state
 22606 *read* when *mailx* quits shall be retained in their current location.

- 22607 *deleted* The message has been processed by one of the following commands: **delete**, **dp**,
 22608 **dt**. Messages in state *deleted* when *mailx* quits shall be deleted. Deleted messages
 22609 shall be ignored until *mailx* quits or changes mailboxes or they are specified to the
 22610 undelete command; for example, the message specification */string* shall only
 22611 search the subject lines of messages that have not yet been deleted, unless the
 22612 command operating on the list of messages is **undelete**. No deleted message or
 22613 deleted message header shall be displayed by any *mailx* command other than
 22614 **undelete**.
- 22615 *preserved* The message has been processed by a **preserve** command. When *mailx* quits, the
 22616 message shall be retained in its current location.
- 22617 *saved* The message has been processed by one of the following commands: **save** or
 22618 **write**. If the current mailbox is the system mailbox, and the internal variable
 22619 **keepsave** is set, messages in the state *saved* shall be saved to the file designated by
 22620 the *MBOX* variable (see the ENVIRONMENT VARIABLES section). If the current
 22621 mailbox is the system mailbox, messages in the state *saved* shall be deleted from
 22622 the current mailbox, when the **quit** or **file** command is used to exit the current
 22623 mailbox.
- 22624 The header-summary line for each message shall indicate the state of the message.
- 22625 Many commands take an optional list of messages (*msglist*) on which to operate, which defaults
 22626 to the current message. A *msglist* is a list of message specifications separated by <blank>
 22627 characters, which can include:
- 22628 *n* Message number *n*.
- 22629 **+** The next undeleted message, or the next deleted message for the **undelete** command.
- 22630 **-** The next previous undeleted message, or the next previous deleted message for the
 22631 **undelete** command.
- 22632 **.** The current message.
- 22633 **^** The first undeleted message, or the first deleted message for the **undelete** command.
- 22634 **\$** The last message.
- 22635 ***** All messages.
- 22636 *n-m* An inclusive range of message numbers.
- 22637 *address* All messages from *address*; any address as shown in a header summary shall be
 22638 matchable in this form.
- 22639 */string* All messages with *string* in the subject line (case ignored).
- 22640 **:c** All messages of type *c*, where *c* shall be one of:
- 22641 **d** Deleted messages.
- 22642 **n** New messages.
- 22643 **o** Old messages (any not in state *read* or *new*).
- 22644 **r** Read messages.
- 22645 **u** Unread messages.
- 22646 Other commands take an optional message (*message*) on which to operate, which defaults to the
 22647 current message. All of the forms allowed for *msglist* are also allowed for *message*, but if more
 22648 than one message is specified, only the first shall be operated on.

22649 Other arguments are usually arbitrary strings whose usage depends on the command involved.

22650 **Start-Up in mailx**

22651 At start-up time, *mailx* shall take the following steps in sequence:

- 22652 1. Establish all variables at their stated default values.
- 22653 2. Process command line options, overriding corresponding default values.
- 22654 3. Import any of the *DEAD*, *EDITOR*, *MBOX*, *LISTER*, *PAGER*, *SHELL*, or *VISUAL* variables
22655 that are present in the environment, overriding the corresponding default values.
- 22656 4. Read *mailx* commands from an unspecified system start-up file, unless the `-n` option is
22657 given, to initialize any internal *mailx* variables and aliases.
- 22658 5. Process the start-up file of *mailx* commands named in the user *MAILRC* variable.

22659 Most regular *mailx* commands are valid inside start-up files, the most common use being to set
22660 up initial display options and alias lists. The following commands shall be invalid in the start-up
22661 file: **!**, **edit**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **shell**, **visual**, **Copy**, **followup**, and **Followup**.
22662 Any errors in the start-up file shall either cause *mailx* to terminate with a diagnostic message and
22663 a non-zero status or to continue after writing a diagnostic message, ignoring the remainder of
22664 the lines in the start-up file.

22665 A blank line in a start-up file shall be ignored.

22666 **Internal Variables in mailx**

22667 The following variables are internal *mailx* variables. Each internal variable can be set via the
22668 *mailx set* command at any time. The **unset** and **set no name** commands can be used to erase
22669 variables.

22670 In the following list, variables shown as:

22671 `variable`

22672 represent Boolean values. Variables shown as:

22673 `variable=value`

22674 shall be assigned string or numeric values. For string values, the rules in **Commands in mailx**
22675 (on page 2804) concerning file names and quoting also apply.

22676 The defaults specified here may be changed by the implementation-defined system start-up file
22677 unless the user specifies the `-n` option.

22678 **allnet** All network names whose login name components match are treated as identical.
22679 This shall cause the *msglist* message specifications to behave similarly. The default
22680 shall be **noallnet**. See also the **alternates** command and the **metoo** variable.

22681 **append** Append messages to the end of the **mbox** file upon termination instead of placing
22682 them at the beginning. The default shall be **noappend**. This variable shall not
22683 affect the **save** command when saving to the **mbox**.

22684 **ask, asksub**

22685 Prompt for a subject line on outgoing mail if one is not specified on the command
22686 line with the `-s` option. The **ask** and **asksub** forms are synonyms; the system shall
22687 refer to **asksub** and **noasksub** in its messages, but shall accept **ask** and **noask** as
22688 user input to mean **asksub** and **noasksub**. It shall not be possible to set both **ask**
22689 and **noasksub**, or **noask** and **asksub**. The default shall be **asksub**, but no

22690		prompting shall be done if standard input is not a terminal.
22691	askbcc	Prompt for the blind copy list. The default shall be noaskbcc .
22692	askcc	Prompt for the copy list. The default shall be noaskcc .
22693	autoprint	Enable automatic writing of messages after delete and undelete commands. The default shall be noautoprint .
22694		
22695	bang	Enable the special-case treatment of exclamation marks ('!') in escape command lines; see the escape command and Command Escapes in mailx (on page 2812). The default shall be nobang , disabling the expansion of '!' in the <i>command</i> argument to the '~!' command and the '~! <i>command</i> escape.
22696		
22697		
22698		
22699	cmd=command	
22700		Set the default command to be invoked by the pipe command. The default shall be nocmd .
22701		
22702	crt=number	Pipe messages having more than <i>number</i> lines through the command specified by the value of the <i>PAGER</i> variable. The default shall be nocrt . If it is set to null, the value used is implementation-defined.
22703		
22704		
22705 XSI	debug	Enable verbose diagnostics for debugging. Messages are not delivered. The default shall be nodebug .
22706		
22707	dot	When dot is set, a period on a line by itself during message input from a terminal shall also signify end-of-file (in addition to normal end-of-file). The default shall be nodot . If ignoreeof is set (see below), a setting of nodot shall be ignored and the period is the only method to terminate input mode.
22708		
22709		
22710		
22711	escape=c	Set the command escape character to be the character 'c'. By default, the command escape character shall be tilde. If escape is unset, tilde shall be used; if it is set to null, command escaping shall be disabled.
22712		
22713		
22714	flipr	Reverse the meanings of the R and r commands. The default shall be noflipr .
22715	folder=directory	
22716		The default directory for saving mail files. User-specified file names beginning with a plus sign ('+') shall be expanded by preceding the file name with this directory name to obtain the real path name. If <i>directory</i> does not start with a slash ('/'), the contents of <i>HOME</i> shall be prefixed to it. The default shall be nofolder .
22717		If folder is unset or set to null, user-specified file names beginning with '+' shall refer to files in the current directory that begin with the literal '+' character. See also outfolder below. The folder value need not affect the processing of the files named in <i>MBOX</i> and <i>DEAD</i> .
22718		
22719		
22720		
22721		
22722		
22723		
22724	header	Enable writing of the header summary when entering <i>mailx</i> in Receive Mode. The default shall be header .
22725		
22726	hold	Preserve all messages that are read in the system mailbox instead of putting them in the mbox save file. The default shall be nohold .
22727		
22728	ignore	Ignore interrupts while entering messages. The default shall be noignore .
22729	ignoreeof	Ignore normal end-of-file during message input. Input can be terminated only by entering a period ('.') on a line by itself or by the '~.' command escape. The default shall be noignoreeof . See also dot above.
22730		
22731		
22732	indentprefix=string	
22733		A string that shall be added as a prefix to each line that is inserted into the message

22734		by the <code>~m</code> command escape. This variable shall default to one <code><tab></code> character.
22735	keep	When a system mailbox, secondary mailbox, or mbox is empty, truncate it to zero length instead of removing it. The default shall be nokeep .
22736		
22737	keepsave	Keep the messages that have been saved from the system mailbox into other files in the file designated by the variable <i>MBOX</i> , instead of deleting them. The default shall be nokeepsave .
22738		
22739		
22740	metoo	Suppress the deletion of the login name of the user from the recipient list when replying to a message or sending to a group. The default shall be nometoo .
22741		
22742 XSI	onehop	When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (that is, one hop away). The default shall be noonehop .
22743		
22744		
22745		
22746		
22747	outfolder	Cause the files used to record outgoing messages to be located in the directory specified by the folder variable unless the path name is absolute. The default shall be nooutfolder . See the record variable.
22748		
22749		
22750	page	Insert a <code><form-feed></code> after each message sent through the pipe created by the pipe command. The default shall be nopage .
22751		
22752	prompt=string	
22753		Set the command-mode prompt to <i>string</i> . If <i>string</i> is null or if noprompt is set, no prompting shall occur. The default shall be to prompt with the string " ? ".
22754		
22755	quiet	Refrain from writing the opening message and version when entering <i>mailx</i> . The default shall be noquiet .
22756		
22757	record=file	Record all outgoing mail in the file with the path name <i>file</i> . The default shall be norecord . See also outfolder above.
22758		
22759	save	Enable saving of messages in the dead-letter file on interrupt or delivery error. See the variable <i>DEAD</i> for the location of the dead-letter file. The default shall be save .
22760		
22761	screen=number	
22762		Set the number of lines in a screenful of headers for the headers and z commands. If screen is not specified, a value based on the terminal type identified by the <i>TERM</i> environment variable, the window size, the baud rate, or some combination of these shall be used.
22763		
22764		
22765		
22766	sendwait	Wait for the background mailer to finish before returning. The default shall be nosendwait .
22767		
22768	showto	When the sender of the message was the user who is invoking <i>mailx</i> , write the information from the To: line instead of the From: line in the header summary. The default shall be noshowto .
22769		
22770		
22771	sign=string	Set the variable inserted into the text of a message when the <code>~a</code> command escape is given. The default shall be nosign . The character sequences <code>'\t'</code> and <code>'\n'</code> shall be recognized in the variable as <code><tab></code> and <code><newline></code> characters, respectively. (See also <code>~i</code> in Command Escapes in mailx (on page 2812).)
22772		
22773		
22774		
22775	Sign=string	Set the variable inserted into the text of a message when the <code>~A</code> command escape is given. The default shall be noSign . The character sequences <code>'\t'</code> and <code>'\n'</code> shall be recognized in the variable as <code><tab></code> and <code><newline></code> characters, respectively.
22776		
22777		

22778 **toplines=number**
 22779 Set the number of lines of the message to write with the **top** command. The default
 22780 shall be 5.

22781 **Commands in mailx**

22782 The following *mailx* commands shall be provided. In the following list, header refers to lines
 22783 from the message header, as shown in the OUTPUT FILES section. Header-line refers to lines
 22784 within the header that begin with one or more non-white-space characters, immediately
 22785 followed by a colon and white space and continuing until the next line beginning with a non-
 22786 white-space character or an empty line. Header-field refers to the portion of a header line prior
 22787 to the first colon in that line.

22788 For each of the commands listed below, the command can be entered as the abbreviation (those
 22789 characters in the Synopsis command word preceding the '['), the full command (all characters
 22790 shown for the command word, omitting the '[' and ']'), or any truncation of the full
 22791 command down to the abbreviation. For example, the **exit** command (shown as **ex[it]** in the
 22792 Synopsis) can be entered as **ex**, **exi**, or **exit**.

22793 The arguments to commands can be quoted, using the following methods:

- 22794 • An argument can be enclosed between paired double-quotes (" ") or single-quotes (' '); any
 22795 white space, shell word expansion, or backslash characters within the quotes shall be treated
 22796 literally as part of the argument. A double-quote shall be treated literally within single-
 22797 quotes and *vice versa*. These special properties of the quote marks shall occur only when they
 22798 are paired at the beginning and end of the argument.
- 22799 • A backslash outside of the enclosing quotes shall be discarded and the following character
 22800 treated literally as part of the argument.
- 22801 • An unquoted backslash at the end of a command line shall be discarded and the next line
 22802 shall continue the command.

22803 File names, where expected, shall be subjected to the process of shell word expansions (see
 22804 Section 2.6 (on page 2244)); if more than a single path name results and the command is
 22805 expecting one file, the effects are unspecified. If the file name begins with an unquoted plus sign,
 22806 it shall not be expanded, but treated as the named file (less the leading plus) in the **folder**
 22807 directory. (See the **folder** variable.)

22808 **Declare Aliases**

22809 *Synopsis:* a[lias] [alias [address...]]
 22810 g[roup] [alias [address...]]

22811 Add the given addresses to the alias specified by *alias*. The names shall be substituted when
 22812 *alias* is used as a recipient address specified by the user in an outgoing message (that is, other
 22813 recipients addressed indirectly through the **reply** command shall not be substituted in this
 22814 manner). Mail address alias substitution shall apply only when the alias string is used as a full
 22815 address; for example, when **hlj** is an alias, *hlj@posix.com* does not trigger the alias substitution. If
 22816 no arguments are given, write a listing of the current aliases to standard output. If only an *alias*
 22817 argument is given, write a listing of the specified alias to standard output. These listings need
 22818 not reflect the same order of addresses that were entered.

22819 **Declare Alternatives**22820 *Synopsis:* alt[ernates] name...

22821 (See also the **metoo** command.) Declare a list of alternative names for the user's login. When
 22822 responding to a message, these names shall be removed from the list of recipients for the
 22823 response. The comparison of names shall be in a case-insensitive manner. With no arguments,
 22824 **alternates** shall write the current list of alternative names.

22825 **Change Current Directory**22826 *Synopsis:* cd [directory]

22827 ch[dir] [directory]

22828 Change directory. If *directory* is not specified, the contents of *HOME* shall be used.22829 **Copy Messages**22830 *Synopsis:* c[opy] [file]

22831 c[opy] [msglist] file

22832 C[opy] [msglist]

22833 Copy messages to the file named by the path name *file* without marking the messages as saved.
 22834 Otherwise, it shall be equivalent to the **save** command.

22835 In the capitalized form, save the specified messages in a file whose name is derived from the
 22836 author of the message to be saved, without marking the messages as saved. Otherwise, it shall
 22837 be equivalent to the **Save** command.

22838 **Delete Messages**22839 *Synopsis:* d[ele]te [msglist]

22840 Mark messages for deletion from the mailbox. The deletions shall not occur until *mailx* quits (see
 22841 the **quit** command) or changes mailboxes (see the **folder** command). If **autoprint** is set and there
 22842 are messages remaining after the **delete** command, the current message shall be written as
 22843 described for the **print** command (see the **print** command); otherwise, the *mailx* prompt shall be
 22844 written.

22845 **Discard Header Fields**22846 *Synopsis:* di[scard] [header-field...]

22847 ig[nore] [header-field...]

22848 Suppress the specified header fields when writing messages. Specified *header-fields* shall be
 22849 added to the list of suppressed header fields. Examples of header fields to ignore are **status** and
 22850 **cc**. The fields shall be included when the message is saved. The **Print** and **Type** commands shall
 22851 override this command. The comparison of header fields shall be in a case-insensitive manner. If
 22852 no arguments are specified, write a list of the currently suppressed header fields to standard
 22853 output; the listing need not reflect the same order of header fields that were entered.

22854 If both **retain** and **discard** commands are given, **discard** commands shall be ignored.

22855 **Delete Messages and Display**

22856 *Synopsis:* dp [*msglist*]
 22857 dt [*msglist*]

22858 Delete the specified messages as described for the **delete** command, except that the **autoprint** variable shall have no effect, and the current message shall be written only if it was set to a message after the last message deleted by the command. Otherwise, an informational message to the effect that there are no further messages in the mailbox shall be written, followed by the *mailx* prompt.

22863 **Echo a String**

22864 *Synopsis:* ec[ho] *string* ...

22865 Echo the given strings, equivalent to the shell *echo* utility.

22866 **Edit Messages**

22867 *Synopsis:* e[dit] [*msglist*]

22868 Edit the given messages. The messages shall be placed in a temporary file and the utility named by the *EDITOR* variable is invoked to edit each file in sequence. The default *EDITOR* is unspecified.

22871 The **edit** command does not modify the contents of those messages in the mailbox.

22872 **Exit**

22873 *Synopsis:* ex[it]
 22874 x[it]

22875 Exit from *mailx* without changing the mailbox. No messages shall be saved in the **mbox** (see also **quit**).

22877 **Change Folder**

22878 *Synopsis:* fi[le] [*file*]
 22879 fold[er] [*file*]

22880 Quit (see the **quit** command) from the current file of messages and read in the file named by the path name *file*. If no argument is given, the name and status of the current mailbox shall be written.

22883 Several unquoted special characters shall be recognized when used as *file* names, with the following substitutions:

22885 % The system mailbox for the invoking user.

22886 %*user* The system mailbox for *user*.

22887 # The previous file.

22888 & The current **mbox**.

22889 +*file* The named file in the **folder** directory. (See the **folder** variable.)

22890 The default file shall be the current mailbox.

22891 **Display List of Folders**22892 *Synopsis:* folders22893 Write the names of the files in the directory set by the **folder** variable. The command specified by
22894 the *LISTER* environment variable shall be used (see the ENVIRONMENT VARIABLES section).22895 **Follow Up Specified Messages**22896 **Notes to Reviewers**22897 *This section with side shading will not appear in the final copy. - Ed.*

22898 D1, XCU, ERN 300 says that it appears the second sentence below applies to both forms.

22899 *Synopsis:* fo[llowup] [*message*]22900 F[ollowup] [*msglist*]22901 In the lowercase form, respond to a message, recording the response in a file whose name is
22902 derived from the author of the message. Overrides the **record** variable, if set. See also the **save**
22903 and **copy** commands and **outfolder**.22904 In the capitalized form, respond to the first message in the *msglist*, sending the message to the
22905 author of each message in the *msglist*. The subject line shall be taken from the first message and
22906 the response shall be recorded in a file whose name is derived from the author of the first
22907 message. See also the **Save** and **Copy** commands and **outfolder**.22908 **Display Header Summary for Specified Messages**22909 *Synopsis:* f[rom] [*msglist*]

22910 Write the header summary for the specified messages.

22911 **Display Header Summary**22912 *Synopsis:* h[eaders] [*message*]22913 Write the page of headers that includes the message specified. If the *message* argument is not
22914 specified, the current message shall not change. However, if the *message* argument is specified,
22915 the current message shall become the message that appears at the top of the page of headers that
22916 includes the message specified. The **screen** variable sets the number of headers per page. See
22917 also the **z** command.22918 **Help**22919 *Synopsis:* hel[p]

22920 ?

22921 Write a summary of commands.

22922 **Hold Messages**22923 *Synopsis:* ho[ld] [*msglist*]22924 pre[serve] [*msglist*]22925 Mark the messages in *msglist* to be retained in the mailbox when *mailx* terminates. This shall
22926 override any commands that might previously have marked the messages to be deleted. During
22927 the current invocation of *mailx*, only the **delete**, **dp**, or **dt** commands shall remove the *preserve*
22928 marking of a message.

22929 **Execute Commands Conditionally**

22930 *Synopsis:* i[f] s|r
 22931 mail-commands
 22932 el[se]
 22933 mail-commands
 22934 en[dif]

22935 Execute commands conditionally, where **if s** executes the following *mail-commands*, up to an
 22936 **else** or **endif**, if the program is in Send Mode, and **if r** shall cause the *mail-commands* to be
 22937 executed only in Receive Mode.

22938 **List Available Commands**

22939 *Synopsis:* l[ist]

22940 Write a list of all commands available. No explanation shall be given.

22941 **Mail a Message**

22942 *Synopsis:* m[ail] address...

22943 Mail a message to the specified addresses or aliases.

22944 **Direct Messages to mbox**

22945 *Synopsis:* mb[ox] [msglist]

22946 Arrange for the given messages to end up in the **mbox** save file when *mailx* terminates normally.
 22947 See *MBOX*. See also the **exit** and **quit** commands.

22948 **Process Next Specified Message**

22949 *Synopsis:* n[ext] [message]

22950 If the current message has not been written (for example, by the **print** command) since *mailx*
 22951 started or since any other message was the current message, behave as if the **print** command
 22952 was entered. Otherwise, if there is an undeleted message after the current message, make it the
 22953 current message and behave as if the **print** command was entered. Otherwise, an informational
 22954 message to the effect that there are no further messages in the mailbox shall be written, followed
 22955 by the *mailx* prompt.

22956 **Pipe Message**

22957 *Synopsis:* pi[pe] [[msglist] command]
 22958 | [[msglist] command]

22959 Pipe the messages through the given *command* by invoking the command interpreter specified
 22960 by *SHELL* with two arguments: **-c** and *command*. (See also *sh -c*.) The application shall ensure
 22961 that the command is given as a single argument. Quoting, described previously, can be used to
 22962 accomplish this. If no arguments are given, the current message shall be piped through the
 22963 command specified by the value of the **cmd** variable. If the **page** variable is set, a <form-feed>
 22964 character shall be inserted after each message.

22965 **Display Message with Headers**

22966 *Synopsis:* P[rint] [msglist]
 22967 T[ype] [msglist]

22968 Write the specified messages, including all header lines, to standard output. Override
 22969 suppression of lines by the **discard**, **ignore**, and **retain** commands. If **crt** is set, the messages
 22970 longer than the number of lines specified by the **crt** variable shall be paged through the
 22971 command specified by the *PAGER* environment variable.

22972 **Display Message**

22973 *Synopsis:* p[rint] [msglist]
 22974 t[ype] [msglist]

22975 Write the specified messages to standard output. If **crt** is set, the messages longer than the
 22976 number of lines specified by the **crt** variable shall be paged through the command specified by
 22977 the *PAGER* environment variable.

22978 **Quit**

22979 *Synopsis:* q[uit]
 22980 end-of-file

22981 Terminate *mailx*, storing messages that were read in **mbox** (if the current mailbox is the system
 22982 mailbox and unless **hold** is set), deleting messages that have been explicitly saved (unless
 22983 **keepsave** is set), discarding messages that have been deleted, and saving all remaining messages
 22984 in the mailbox.

22985 **Reply to a Message List**

22986 *Synopsis:* R[eply] [msglist]
 22987 R[espond] [msglist]

22988 Mail a reply message to the sender of each message in the *msglist*. The subject line shall be
 22989 formed by concatenating **Re:**<space> (unless it already begins with that string) and the subject
 22990 from the first message. If **record** is set to a file name, the response shall be saved at the end of
 22991 that file.

22992 See also the **flipr** variable.

22993 **Reply to a Message**

22994 *Synopsis:* r[eply] [message]
 22995 r[espond] [message]

22996 Mail a reply message to all recipients included in the header of the message. The subject line
 22997 shall be formed by concatenating **Re:**<space> (unless it already begins with that string) and the
 22998 subject from the message. If **record** is set to a file name, the response shall be saved at the end of
 22999 that file.

23000 See also the **flipr** variable.

23001 **Retain Header Fields**23002 *Synopsis:* ret[ain] [*header-field...*]

23003 Retain the specified header fields when writing messages. This command shall override all
 23004 **discard** and **ignore** commands. The comparison of header fields shall be in a case-insensitive
 23005 manner. If no arguments are specified, write a list of the currently retained header fields to
 23006 standard output; the listing need not reflect the same order of header fields that were entered.

23007 **Save Messages**23008 *Synopsis:* s[ave] [*file*]23009 s[ave] [*msglist*] *file*23010 S[ave] [*msglist*]

23011 Save the specified messages in the file named by the path name *file*, or the **mbox** if the *file*
 23012 argument is omitted. The file shall be created if it does not exist; otherwise, the messages shall be
 23013 appended to the file. The message shall be put in the state *saved*, and shall behave as specified in
 23014 the description of the *saved* state when the current mailbox is exited by the **quit** or **file**
 23015 command.

23016 In the capitalized form, save the specified messages in a file whose name is derived from the
 23017 author of the first message. The name of the file shall be taken to be the author's name with all
 23018 network addressing stripped off. See also the **Copy**, **followup**, and **Followup** commands and
 23019 **outfolder** variable.

23020 **Set Variables**23021 *Synopsis:* se[t] [*name*=[*string*]] ...] [*name=number* ...] [*noname* ...]

23022 Define one or more variables called *name*. The variable can be given a null, string, or numeric
 23023 value. Quoting and backslash escapes can occur anywhere in *string*, as described previously, as
 23024 if the *string* portion of the argument were the entire argument. The forms *name* and *name=* shall
 23025 be equivalent to *name=""* for variables that take string values. The **set** command without
 23026 arguments shall write a list of all defined variables and their values. The **no name** form shall be
 23027 equivalent to **unset name**.

23028 **Invoke a Shell**23029 *Synopsis:* sh[ell]23030 Invoke an interactive command interpreter (see also *SHELL*).23031 **Display Message Size**23032 *Synopsis:* si[ze] [*msglist*]

23033 Write the size in bytes of each of the specified messages.

23034 **Read mailx Commands From a File**23035 *Synopsis:* so[urce] *file*

23036 Read and execute commands from the file named by the path name *file* and return to command
 23037 mode.

23038 Display Beginning of Messages

23039 *Synopsis:* to[p] [msglist]

23040 Write the top few lines of each of the specified messages. If the **toplines** variable is set, it is taken
23041 as the number of lines to write. The default shall be 5.

23042 Touch Messages

23043 *Synopsis:* tou[ch] [msglist]

23044 Touch the specified messages. If any message in *msglist* is not specifically deleted nor saved in a
23045 file, it shall be placed in the **mbox** upon normal termination. See **exit** and **quit**.

23046 Delete Aliases

23047 *Synopsis:* una[lias] [alias]...

23048 Delete the specified alias names. If a specified alias does not exist, the results are unspecified.

23049 Undelete Messages

23050 *Synopsis:* u[ndelete] [msglist]

23051 Change the state of the specified messages from deleted to read. If **autoprint** is set, the last
23052 message of those restored shall be written. If *msglist* is not specified, the message shall be
23053 selected as follows:

- 23054 • If there are any deleted messages that follow the current message, the first of these shall be
23055 chosen.
- 23056 • Otherwise, the last deleted message that also precedes the current message shall be chosen.

23057 Unset Variables

23058 *Synopsis:* uns[et] name...

23059 Cause the specified variables to be erased.

23060 Edit Message with Full-Screen Editor

23061 *Synopsis:* v[isual] [msglist]

23062 Edit the given messages with a screen editor. Each message shall be placed in a temporary file,
23063 and the utility named by the *VISUAL* variable shall be invoked to edit each file in sequence. The
23064 default editor shall be *vi*.

23065 The **visual** command does not modify the contents of those messages in the mailbox.

23066 Write Messages to a File

23067 *Synopsis:* w[rite] [msglist] file

23068 Write the given messages to the file specified by the path name *file*, minus the message header.
23069 Otherwise, it shall be equivalent to the **save** command.

23070 **Scroll Header Display**23071 *Synopsis:* z[+|-]

23072 Scroll the header display forward (if '+' is specified or if no option is specified) or backward (if
 23073 '-' is specified) one screenful. The number of headers written shall be set by the **screen**
 23074 variable.

23075 **Invoke Shell Command**23076 *Synopsis:* !*command*

23077 Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and *command*.
 23078 (See also *sh -c*.) If the **bang** variable is set, each unescaped occurrence of '!' in *command* shall
 23079 be replaced with the command executed by the previous ! *command* or '! *command* escape.

23080 **Null Command**23081 *Synopsis:* # *comment*23082 This null command (comment) shall be ignored by *mailx*.23083 **Display Current Message Number**23084 *Synopsis:* =

23085 Write the current message number.

23086 **Command Escapes in mailx**

23087 The following commands can be entered only from input mode, by beginning a line with the
 23088 escape character (by default, tilde ('~')). See the **escape** variable description for changing this
 23089 special character. The format for the commands shall be:

23090 <ESC><*command-char*><*separator*>[<*arguments*>]23091 where the <*separator*> can be zero or more <blank> characters.

23092 In the following descriptions, the application shall ensure that the argument *command* (but not
 23093 *mailx-command*) is a shell command string. Any string acceptable to the command interpreter
 23094 specified by the *SHELL* variable when it is invoked as *SHELL -c command_string* shall be valid.
 23095 The command can be presented as multiple arguments (that is, quoting is not required).

23096 Command escapes that are listed with *msglist* or *mailx-command* arguments are invalid in Send
 23097 Mode and produce unspecified results.

23098 **~! *command*** Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and
 23099 *command*; and then return to input mode. If the **bang** variable is set, each
 23100 unescaped occurrence of '!' in *command* shall be replaced with the command
 23101 executed by the previous ! *command* or '! *command* escape.

23102 **~.** Simulate end-of-file (terminate message input).

23103 **~: *mailx-command*, ~_ *mailx-command***
 23104 Perform the command-level request.

23105 **~?** Write a summary of command escapes.23106 **~A** This shall be equivalent to **~i Sign**.23107 **~a** This shall be equivalent to **~i sign**.

23108	~b <i>name...</i>	Add the <i>names</i> to the blind carbon copy (Bcc) list.
23109	~c <i>name...</i>	Add the <i>names</i> to the carbon copy (Cc) list.
23110	~d	Read in the dead-letter file. See <i>DEAD</i> for a description of this file.
23111	~e	Invoke the editor, as specified by the <i>EDITOR</i> environment variable, on the partial message.
23112		
23113	~f [<i>msglist</i>]	Forward the specified messages. The specified messages shall be inserted into the current message without alteration. This command escape also shall insert message headers into the message with field selection affected by the discard , ignore , and retain commands.
23114		
23115		
23116		
23117	~F [<i>msglist</i>]	This shall be the equivalent of the ~f command escape, except that all headers shall be included in the message, regardless of previous discard , ignore , and retain commands.
23118		
23119		
23120	~h	If standard input is a terminal, prompt for a Subject line and the To , Cc , and Bcc lists. Other implementation-defined headers may also be presented for editing. If the field is written with an initial value, it can be edited as if it had just been typed.
23121		
23122		
23123	~i <i>string</i>	Insert the value of the named variable, followed by a <newline> character, into the text of the message. If the string is unset or null, the message shall not be changed.
23124		
23125	~m [<i>msglist</i>]	Insert the specified messages into the message, prefixing non-empty lines with the string in the indentprefix variable. This command escape also shall insert message headers into the message, with field selection affected by the discard , ignore , and retain commands.
23126		
23127		
23128		
23129	~M [<i>msglist</i>]	This shall be the equivalent of the ~m command escape, except that all headers shall be included in the message, regardless of previous discard , ignore , and retain commands.
23130		
23131		
23132	~p	Write the message being entered. If the message is longer than crt lines (see Internal Variables in mailx (on page 2801)), the output shall be paginated as described for the <i>PAGER</i> variable.
23133		
23134		
23135	~q	Quit (see the quit command) from input mode by simulating an interrupt. If the body of the message is not empty, the partial message shall be saved in the dead-letter file. See <i>DEAD</i> for a description of this file.
23136		
23137		
23138	~r <i>file</i> , ~< <i>file</i> , ~r ! <i>command</i> , ~< ! <i>command</i> "	
23139		Read in the file specified by the path name <i>file</i> . If the argument begins with an exclamation mark (' ! '), the rest of the string shall be taken as an arbitrary system command; the command interpreter specified by <i>SHELL</i> shall be invoked with two arguments: -c and <i>command</i> . The standard output of <i>command</i> shall be inserted into the message.
23140		
23141		
23142		
23143		
23144	~s <i>string</i>	Set the subject line to <i>string</i> .
23145	~t <i>name...</i>	Add the given <i>names</i> to the To list.
23146	~v	Invoke the full-screen editor, as specified by the <i>VISUAL</i> environment variable, on the partial message.
23147		
23148	~w <i>file</i>	Write the partial message, without the header, onto the file named by the path name <i>file</i> . The file shall be created or the message shall be appended to it if the file exists.
23149		
23150		

23151 ~x Exit as with ~q, except the message shall not be saved in the dead-letter file.

23152 ~| *command* Pipe the body of the message through the given *command* by invoking the
23153 command interpreter specified by *SHELL* with two arguments: -c and *command*.
23154 If the *command* returns a successful exit status, the standard output of the
23155 command shall replace the message. Otherwise, the message shall remain
23156 unchanged. If the *command* fails, an error message giving the exit status shall be
23157 written.

23158 **EXIT STATUS**

23159 When the -e option is specified, the following exit values are returned:

23160 0 Mail was found.

23161 >0 Mail was not found or an error occurred.

23162 Otherwise, the following exit values are returned:

23163 0 Successful completion; note that this status implies that all messages were *sent*, but it gives
23164 no assurances that any of them were actually *delivered*.

23165 >0 An error occurred.

23166 **CONSEQUENCES OF ERRORS**

23167 When in input mode (Receive Mode) or Send Mode:

23168 • If an error is encountered processing a command escape (see **Command Escapes in mailx**
23169 on page 2812)), a diagnostic message shall be written to standard error, and the message
23170 being composed may be modified, but this condition shall not prevent the message from
23171 being sent.

23172 • Other errors shall prevent the sending of the message.

23173 When in command mode:

23174 • Default.

23175 **APPLICATION USAGE**

23176 Delivery of messages to remote systems requires the existence of communication paths to such
23177 systems. These need not exist.

23178 Input lines are limited to {LINE_MAX} bytes, but mailers between systems may impose more
23179 severe line-length restrictions. This volume of IEEE Std. 1003.1-200x does not place any
23180 restrictions on the length of messages handled by *mailx*, and for delivery of local messages the
23181 only limitations should be the normal problems of available disk space for the target mail file.
23182 When sending messages to external machines, applications are advised to limit messages to less
23183 than 100 kilobytes because some mail gateways impose message-length restrictions.

23184 The format of the system mailbox is intentionally unspecified. Not all systems implement
23185 system mailboxes as flat files, particularly with the advent of multimedia mail messages. Some
23186 system mailboxes may be multiple files, others records in a database. The internal format of the
23187 messages themselves are specified with the historical format from Version 7, but only after they
23188 have been saved in some file other than the system mailbox. This was done so that many
23189 historical applications expecting text-file mailboxes are not broken.

23190 Some new formats for messages can be expected in the future, probably including binary data,
23191 bit maps, and various multimedia objects. As described here, *mailx* is not prohibited from
23192 handling such messages, but it must store them as text files in secondary mailboxes (unless
23193 some extension, such as a variable or command line option, is used to change the stored format).
23194 Its method of doing so is implementation-defined and might include translating the data into

23195 text file-compatible or readable form or omitting certain portions of the message from the stored
23196 output.

23197 The **discard** and **ignore** commands are not inverses of the **retain** command. The **retain**
23198 command discards all header-fields except those explicitly retained. The **discard** command
23199 keeps all header-fields except those explicitly discarded. If headers exist on the retained header
23200 list, **discard** and **ignore** commands are ignored.

23201 EXAMPLES

23202 None.

23203 RATIONALE

23204 The standard developers felt strongly that a method for applications to send messages to
23205 specific users was necessary. The obvious example is a batch utility, running non-interactively,
23206 that wishes to communicate errors or results to a user. However, the actual format, delivery
23207 mechanism, and method of reading the message are clearly beyond the scope of this volume of
23208 IEEE Std. 1003.1-200x.

23209 The intent of this command is to provide a simple, portable interface for sending messages non-
23210 interactively. It merely defines a “front-end” to the historical mail system. It is suggested that
23211 implementations explicitly denote the sender and recipient in the body of the delivered message.
23212 Further specification of formats for either the message envelope or the message itself were
23213 deliberately not made, as the industry is in the midst of changing from the current standards to
23214 a more internationalized standard and it is probably incorrect, at this time, to require either one.

23215 Implementations are encouraged to conform to the various delivery mechanisms described in
23216 the CCITT X.400 standards or to the equivalent Internet standards, described in Internet Request
23217 for Comment (RFC) documents RFC 819, RFC 822, RFC 920, RFC 921, and RFC 1123.

23218 Many historical systems modified each body line that started with **From** by prefixing the ‘F’
23219 with ‘>’. It is unnecessary, but allowed, to do that when the string does not follow a blank line
23220 because it cannot be confused with the next header.

23221 The *edit* and *visual* commands merely edit the specified messages in a temporary file. They do
23222 not modify the contents of those messages in the mailbox; such a capability could be added as an
23223 extension, such as by using different command names.

23224 The restriction on a subject line being {LINE_MAX}-10 bytes is based on the historical format
23225 that consumes 10 bytes for **Subject:** and the trailing <newline>. Many historical mailers that a
23226 message may encounter on other systems are not able to handle lines that long, however.

23227 Like the utilities *logger* and *lp*, *mailx* admittedly is difficult to test. This was not deemed sufficient
23228 justification to exclude this utility from this volume of IEEE Std. 1003.1-200x. It is also arguable
23229 that it is, in fact, testable, but that the tests themselves are not portable.

23230 When *mailx* is being used by an application that wishes to receive the results as if none of the
23231 User Portability Utilities option features were supported, the *DEAD* environment variable must
23232 be set to **/dev/null**. Otherwise, it may be subject to the file creations described in *mailx*
23233 ASYNCHRONOUS EVENTS. Similarly, if the *MAILRC* environment variable is not set to
23234 **/dev/null**, historical versions of *mailx* and *Mail* read initialization commands from a file before
23235 processing begins. Since the initialization that a user specifies could alter the contents of
23236 messages an application is trying to send, such applications must set *MAILRC* to **/dev/null**.

23237 The description of *LC_TIME* uses “may affect” because many historical implementations do not
23238 or cannot manipulate the date and time strings in the incoming mail headers. Some headers
23239 found in incoming mail do not have enough information to determine the timezone in which the
23240 mail originated, and, therefore, *mailx* cannot convert the date and time strings into the internal
23241 form that then is parsed by routines like *strftime()* that can take *LC_TIME* settings into account.

23242 Changing all these times to a user-specified format is allowed, but not required.

23243 The paginator selected when *PAGER* is null or unset is partially unspecified to allow the System
23244 V historical practice of using *pg* as the default. Bypassing the pagination function, such as by
23245 declaring that *cat* is the paginator, would not meet with the intended meaning of this
23246 description. However, any “portable user” would have to set *PAGER* explicitly to get his or her
23247 preferred paginator on all systems. The paginator choice was made partially unspecified, unlike
23248 the *VISUAL* editor choice (mandated to be *vi*) because most historical pagers follow a common
23249 theme of user input, whereas editors differ dramatically.

23250 Options to specify addresses as **cc** (carbon copy) or **bcc** (blind carbon copy) were considered to
23251 be format details and were omitted.

23252 A zero exit status implies that all messages were *sent*, but it gives no assurances that any of them
23253 were actually *delivered*. The reliability of the delivery mechanism is unspecified and is an
23254 appropriate marketing distinction between systems.

23255 In order to conform to the Utility Syntax Guidelines, a solution was required to the optional *file*
23256 option-argument to *-f*. By making *file* an operand, the guidelines are satisfied and users remain
23257 portable. However, it does force implementations to support usage such as:

23258 `mailx -fin mymail.box`

23259 The **no name** method of unsetting variables is not present in all historical systems, but it is in
23260 System V and provides a logical set of commands corresponding to the format of the display of
23261 options from the *mailx set* command without arguments.

23262 The **ask** and **asksub** variables are the names selected by BSD and System V, respectively, for the
23263 same feature. They are synonyms in this volume of IEEE Std. 1003.1-200x.

23264 The *mailx echo* command was not documented in the BSD version and has been omitted here
23265 because it is not obviously useful for interactive users.

23266 The default prompt on the System V *mailx* is a question mark, on BSD *Mail* an ampersand. Since
23267 this volume of IEEE Std. 1003.1-200x chose the *mailx* name, it kept the System V default,
23268 assuming that BSD users would not have difficulty with this minor incompatibility (that they
23269 can override).

23270 The meanings of **r** and **R** are reversed between System V *mailx* and SunOS *Mail*. Once again,
23271 since this volume of IEEE Std. 1003.1-200x chose the *mailx* name, it kept the System V default,
23272 but allows the SunOS user to achieve the desired results using **flpr**, an internal variable in
23273 System V *mailx*, although it has not been documented in the SVID

23274 The **indentprefix** variable, the **retain** and **unalias** commands, and the **~F** and **~M** command
23275 escapes were adopted from 4.3 BSD *Mail*.

23276 The **version** command was not included because no sufficiently general specification of the
23277 version information could be devised that would still be useful to a portable user. This
23278 command name should be used by suppliers who wish to provide version information about the
23279 *mailx* command.

23280 The “implementation-specific (unspecified) system start-up file” historically has been named
23281 **/etc/mailx.rc**, but this specific name and location are not required.

23282 The intent of the wording for the **next** command is that if any command has already displayed
23283 the current message it should display a following message, but, otherwise, it should display the
23284 current message. Consider the command sequence:

23285 `next 3`
23286 `delete 3`

- 23287 next
- 23288 where the **autoprint** option was not set. The normative text specifies that the second **next**
23289 command should display a message following the third message, because even though the
23290 current message has not been displayed since it was set by the **delete** command, it has been
23291 displayed since the current message was anything other than message number 3. This does not
23292 always match historical practice in some implementations, where the command file address
23293 followed by **next** (or the default command) would skip the message for which the user had
23294 searched.
- 23295 **FUTURE DIRECTIONS**
- 23296 None.
- 23297 **SEE ALSO**
- 23298 *ed, ls, more, vi*
- 23299 **CHANGE HISTORY**
- 23300 First released in Issue 2.
- 23301 **Issue 4**
- 23302 Aligned with the ISO/IEC 9945-2: 1993 standard.
- 23303 This utility is now mandatory; it is optional in Issue 3.
- 23304 **Issue 5**
- 23305 The description of the EDITOR environment variable is changed to indicate that *ed* is the default
23306 editor if this variable is not set. In previous issues, this default was not stated explicitly at this
23307 point but was implied further down in the text.
- 23308 **FUTURE DIRECTIONS** section added.
- 23309 **Issue 6**
- 23310 The following new requirements on POSIX implementations derive from alignment with the
23311 Single UNIX Specification:
- 23312 • The **-F** option is added.
 - 23313 • The **allnet**, **debug**, and **sendwait** internal variables are added.
 - 23314 • The **C**, **ec**, **fo**, **F**, and **S** *mailx* commands are added.
- 23315 In the DESCRIPTION and ENVIRONMENT VARIABLES sections, text stating “*HOME*
23316 directory” is replaced by “directory referred to by the *HOME* environment variable”.
- 23317 The *mailx* utility is aligned with the IEEE P1003.2b draft standard, which included various
23318 clarifications to resolve IEEE PASC Interpretations submitted for the ISO POSIX-2:1993
23319 standard. In particular, the changes here address IEEE PASC Interpretations 1003.2 #10, #11,
23320 #103, #106, #108, #114, #115, #122, and #129.
- 23321 The normative text is reworded to avoid use of the term “must” for application requirements.

23322 NAME

23323 make — maintain, update, and regenerate groups of programs (**DEVELOPMENT**)

23324 SYNOPSIS

```
23325 SD make [-einpqrst][-f makefile...] [-k | -S][macro=value]...
23326 [ target_name... ]
```

23327

23328 DESCRIPTION

23329 The *make* utility can be used as a part of software development to update files that are derived
 23330 from other files. A typical case is one where object files are derived from the corresponding
 23331 source files. The *make* utility examines time relationships and updates those derived files (called
 23332 targets) that have modified times earlier than the modified times of the files (called
 23333 prerequisites) from which they are derived. A description file (makefile) contains a description
 23334 of the relationships between files, and the commands that need to be executed to update the
 23335 targets to reflect changes in their prerequisites. Each specification, or rule, shall consist of a
 23336 target, optional prerequisites, and optional commands to be executed when a prerequisite is
 23337 newer than the target. There are two types of rule:

23338 1. *Inference rules*, which have one target name with at least one period ('.') and no slash
 23339 ('/')

23340 2. *Target rules*, which can have more than one target name

23341 In addition, *make* shall have a collection of built-in macros and inference rules that infer
 23342 prerequisite relationships to simplify maintenance of programs.

23343 To receive exactly the behavior described in this section, the user shall ensure that a portable
 23344 makefile:

- 23345 • Includes the special target **.POSIX**
- 23346 • Omits any special target reserved for implementations (a leading period followed by
 23347 uppercase letters) that has not been specified by this section

23348 The behavior of *make* is unspecified if either or both of these conditions are not met.

23349 OPTIONS

23350 The *make* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 23351 12.2, Utility Syntax Guidelines.

23352 The following options shall be supported:

23353 **-e** Cause environment variables, including those with null values, to override macro
 23354 assignments within makefiles.

23355 **-f *makefile*** Specify a different makefile. The argument *makefile* is a path name of a description
 23356 file, which is also referred to as the *makefile*. A path name of '-' shall denote the
 23357 standard input. There can be multiple instances of this option, and they shall be
 23358 processed in the order specified. The effect of specifying the same option-
 23359 argument more than once is unspecified.

23360 **-i** Ignore error codes returned by invoked commands. This mode is the same as if the
 23361 special target **.IGNORE** were specified without prerequisites.

23362 **-k** Continue to update other targets that do not depend on the current target if a non-
 23363 ignored error occurs while executing the commands to bring a target up-to-date.

23364 **-n** Write commands that would be executed on standard output, but do not execute
 23365 them. However, lines with a plus sign ('+') prefix shall be executed. In this mode,

- 23366 lines with an at sign ('@') character prefix shall be written to standard output.
- 23367 **-p** Write to standard output the complete set of macro definitions and target
23368 descriptions. The output format is unspecified.
- 23369 **-q** Return a zero exit value if the target file is up-to-date; otherwise, return an exit
23370 value of 1. Targets shall not be updated if this option is specified. However, a
23371 makefile command line (associated with the targets) with a plus sign ('+') prefix
23372 shall be executed.
- 23373 **-r** Clear the suffix list and does not use the built-in rules.
- 23374 **-S** Terminate *make* if an error occurs while executing the commands to bring a target
23375 up-to-date. This shall be the default and the opposite of **-k**.
- 23376 **-s** Do not write makefile command lines or touch messages (see **-t**) to standard
23377 output before executing. This mode shall be the same as if the special target
23378 **.SILENT** were specified without prerequisites.
- 23379 **-t** Update the modification time of each target as though a *touch target* had been
23380 executed. Targets that have prerequisites but no commands (see **Target Rules** (on
23381 page 2822)), or that are already up-to-date, shall not be touched in this manner.
23382 Write messages to standard output for each target file indicating the name of the
23383 file and that it was touched. Normally, the makefile command lines associated
23384 with each target are not executed. However, a command line with a plus sign
23385 ('+') prefix shall be executed.
- 23386 Any options specified in the *MAKEFLAGS* environment variable shall be evaluated before any
23387 options specified on the *make* utility command line. If the **-k** and **-S** options are both specified
23388 on the *make* utility command line or by the *MAKEFLAGS* environment variable, the last option
23389 specified shall take precedence. If the **-f** or **-p** options appear in the *MAKEFLAGS* environment
23390 variable, the result is undefined.
- 23391 **OPERANDS**
- 23392 The following operands shall be supported:
- 23393 *target_name* Target names, as defined in the EXTENDED DESCRIPTION section. If no target is
23394 specified, while *make* is processing the makefiles, the first target that *make*
23395 encounters that is not a special target or an inference rule shall be used.
- 23396 *macro=value* Macro definitions, as defined in **Macros** (on page 2824).
- 23397 If the *target_name* and *macro=value* operands are intermixed on the *make* utility command line,
23398 the results are unspecified.
- 23399 **STDIN**
- 23400 The standard input shall be used only if the *makefile* option-argument is '-'. See the INPUT
23401 FILES section.
- 23402 **INPUT FILES**
- 23403 The input file, otherwise known as the makefile, is a text file containing rules, macro definitions,
23404 and comments.
- 23405 **ENVIRONMENT VARIABLES**
- 23406 The following environment variables shall affect the execution of *make*:
- 23407 *LANG* Provide a default value for the internationalization variables that are unset or null.
23408 If *LANG* is unset or null, the corresponding value from the implementation-
23409 defined default locale shall be used. If any of the internationalization variables
23410 contains an invalid setting, the utility shall behave as if none of the variables had

- 23411 been defined.
- 23412 *LC_ALL* If set to a non-empty string value, override the values of all the other
23413 internationalization variables.
- 23414 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
23415 characters (for example, single-byte as opposed to multi-byte characters in
23416 arguments and input files).
- 23417 *LC_MESSAGES*
23418 Determine the locale that should be used to affect the format and contents of
23419 diagnostic messages written to standard error.
- 23420 *MAKEFLAGS*
23421 This variable shall be interpreted as a character string representing a series of
23422 option characters to be used as the default options. The implementation shall
23423 accept both of the following formats (but need not accept them when intermixed):
- 23424 • The characters are option letters without the leading hyphens or <blank>
23425 character separation used on a *make* utility command line.
 - 23426 • The characters are formatted in a manner similar to a portion of the *make* utility
23427 command line: options are preceded by hyphens and <blank> character-
23428 separated as described in the Base Definitions volume of IEEE Std. 1003.1-200x,
23429 Section 12.2, Utility Syntax Guidelines. The *macro=value* macro definition
23430 operands can also be included. The difference between the contents of
23431 *MAKEFLAGS* and the *make* utility command line is that the contents of the
23432 variable shall not be subjected to the word expansions (see Section 2.6 (on page
23433 2244)) associated with parsing the command line values.
- 23434 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 23435 XSI *PROJECTDIR*
23436 Provide a directory to be used to search for SCCS files not found in the current
23437 directory. In all of the following cases, the search for SCCS files is made in the
23438 directory *SCCS* in the identified directory. If the value of *PROJECTDIR* begins
23439 with a slash, it shall be considered an absolute path name; otherwise, the value of
23440 *PROJECTDIR* is treated as a user name and that user's initial working directory
23441 shall be examined for a subdirectory *src* or *source*. If such a directory is found, it
23442 shall be used. Otherwise, the value is used as a relative path name.
- 23443 If *PROJECTDIR* is not set or has a null value, the search for SCCS files shall be
23444 made in the directory *SCCS* in the current directory.
- 23445 The setting of *PROJECTDIR* affects all files listed in the remainder of this utility
23446 description for files with a component named *SCCS*.
- 23447 The value of the *SHELL* environment variable shall not be used as a macro and shall not be
23448 modified by defining the *SHELL* macro in a makefile or on the command line. All other
23449 environment variables, including those with null values, shall be used as macros, as defined in
23450 **Macros** (on page 2824).
- 23451 **ASYNCHRONOUS EVENTS**
23452 If not already ignored, *make* shall trap SIGHUP, SIGTERM, SIGINT, and SIGQUIT and remove
23453 the current target unless the target is a directory or the target is a prerequisite of the special
23454 target *.PRECIOUS* or unless one of the *-n*, *-p*, or *-q* options was specified. Any targets removed
23455 in this manner shall be reported in diagnostic messages of unspecified format, written to
23456 standard error. After this cleanup process, if any, *make* shall take the standard action for all other

23457 signals.

23458 **STDOUT**

23459 The *make* utility shall write all commands to be executed to standard output unless the `-s` option
 23460 was specified, the command is prefixed with an at sign, or the special target `.SILENT` has either
 23461 the current target as a prerequisite or has no prerequisites. If *make* is invoked without any work
 23462 needing to be done, it shall write a message to standard output indicating that no action was
 23463 taken. If the `-t` option is present and a file is touched, *make* shall write to standard output a
 23464 message of unspecified format indicating that the file was touched, including the file name of
 23465 the file.

23466 **STDERR**

23467 Used only for diagnostic messages.

23468 **OUTPUT FILES**

23469 Files can be created when the `-t` option is present. Additional files can also be created by the
 23470 utilities invoked by *make*.

23471 **EXTENDED DESCRIPTION**

23472 The *make* utility attempts to perform the actions required to ensure that the specified targets are
 23473 up-to-date. A target is considered out-of-date if it is older than any of its prerequisites or if it
 23474 does not exist. The *make* utility shall treat all prerequisites as targets themselves and recursively
 23475 ensure that they are up-to-date, processing them in the order in which they appear in the rule.
 23476 The *make* utility shall use the modification times of files to determine whether the corresponding
 23477 targets are out-of-date.

23478 After *make* has ensured that all of the prerequisites of a target are up-to-date and if the target is
 23479 out-of-date, the commands associated with the target entry shall be executed. If there are no
 23480 commands listed for the target, the target shall be treated as up-to-date.

23481 **Makefile Syntax**

23482 A makefile can contain rules, macro definitions (see **Macros** (on page 2824)), and comments.
 23483 There are two kinds of rules: *inference rules* and *target rules*. The *make* utility shall contain a set of
 23484 built-in inference rules. If the `-r` option is present, the built-in rules shall not be used and the
 23485 suffix list shall be cleared. Additional rules of both types can be specified in a makefile. If a rule
 23486 is defined more than once, the value of the rule shall be that of the last one specified. Macros can
 23487 also be defined more than once, and the value of the macro is specified in **Macros** (on page
 23488 2824). Comments start with a number sign (`'#'`) and continue until an unescaped `<newline>`
 23489 character is reached.

23490 By default, the following files shall be tried in sequence: `./makefile` and `./Makefile`. If neither
 23491 `./makefile` or `./Makefile` are found, other implementation-defined files may also be tried. On
 23492 XSI-conformant systems, the additional files `./s.makefile`, `SCCS/s.makefile`, `./s.Makefile`, and
 23493 `SCCS/s.Makefile` shall also be tried.

23494 The `-f` option shall direct *make* to ignore any of these default files and use the specified argument
 23495 as a makefile instead. If the `'-'` argument is specified, standard input shall be used.

23496 The term *makefile* is used to refer to any rules provided by the user, whether in `./makefile` or its
 23497 variants, or specified by the `-f` option.

23498 The rules in makefiles shall consist of the following types of lines: target rules, including special
 23499 targets (see **Target Rules** (on page 2822)), inference rules (see **Inference Rules** (on page 2825)),
 23500 macro definitions (see **Macros** (on page 2824)), empty lines, and comments.

23501 When an escaped `<newline>` (one preceded by a backslash) is found anywhere in the makefile
 23502 except in a command line, it shall be replaced, along with any leading white space on the

23503 following line, with a single <space>. When an escaped <newline> is found in a command line
 23504 in a makefile, the command line shall contain the backslash, the <newline>, and the next line,
 23505 except that the first character of the next line shall not be included if it is a <tab>.

23506 **Makefile Execution**

23507 Makefile command lines shall be processed one at a time by writing the makefile command line
 23508 to the standard output (unless one of the conditions listed under '@' suppresses the writing)
 23509 and executing the command(s) in the line. A <tab> character may precede the command to
 23510 standard output. Command execution shall be as if the makefile command line were the
 23511 argument to the *system()* function. The environment for the command being executed shall
 23512 contain all of the variables in the environment of *make*.

23513 By default, when *make* receives a non-zero status from the execution of a command, it terminates
 23514 with an error message to standard error.

23515 Makefile command lines can have one or more of the following prefixes: a hyphen ('-'), an at
 23516 sign ('@'), or a plus sign ('+'). These modify the way in which *make* processes the command.
 23517 When a command is written to standard output, the prefix shall not be included in the output.

23518 – If the command prefix contains a hyphen, or the **-i** option is present, or the special target
 23519 **.IGNORE** has either the current target as a prerequisite or has no prerequisites, any error
 23520 found while executing the command shall be ignored.

23521 @ If the command prefix contains an at sign and the *make* utility command line **-n** option is
 23522 not specified, or the **-s** option is present, or the special target **.SILENT** has either the current
 23523 target as a prerequisite or has no prerequisites, the command shall not be written to
 23524 standard output before it is executed.

23525 + If the command prefix contains a plus sign, this indicates a makefile command line that
 23526 shall be executed even if **-n**, **-q**, or **-t** is specified.

23527 **Target Rules**

23528 Target rules are formatted as follows:

```
23529 target [target...]: [prerequisite...][;command]
23530 [<tab>command
23531 <tab>command
23532 ...]
```

23533 *line that does not begin with <tab>*

23534 Target entries are specified by a <blank> character-separated, non-null list of targets, then a
 23535 colon, then a <blank> character-separated, possibly empty list of prerequisites. Text following a
 23536 semicolon, if any, and all following lines that begin with a <tab> character, are makefile
 23537 command lines to be executed to update the target. The first non-empty line that does not begin
 23538 with a <tab> character or '#' shall begin a new entry. An empty or blank line, or a line
 23539 beginning with '#', may begin a new entry.

23540 Applications shall select target names from the set of characters consisting solely of periods,
 23541 underscores, digits, and alphabetic characters from the portable character set (see the Base Definitions
 23542 volume of IEEE Std. 1003.1-200x, Section 6.1, Portable Character Set). Implementations may
 23543 allow other characters in target names as extensions. The interpretation of targets containing the
 23544 characters '%' and '"' is implementation-defined.

23545 A target that has prerequisites, but does not have any commands, can be used to add to the
 23546 prerequisite list for that target. Only one target rule for any given target can contain commands.

23547 Lines that begin with one of the following are called *special targets* and control the operation of
23548 *make*:

23549 **.DEFAULT** If the makefile uses this special target, the application shall ensure that it is
23550 specified with commands, but without prerequisites. The commands shall be used
23551 by *make* if there are no other rules available to build a target.

23552 **.IGNORE** Prerequisites of this special target are targets themselves; this shall cause errors
23553 from commands associated with them to be ignored in the same manner as
23554 specified by the `-i` option. Subsequent occurrences of **.IGNORE** shall add to the
23555 list of targets ignoring command errors. If no prerequisites are specified, *make* shall
23556 behave as if the `-i` option had been specified and errors from all commands
23557 associated with all targets shall be ignored.

23558 **.POSIX** The application shall ensure that this special target is specified without
23559 prerequisites or commands. If it appears as the first non-comment line in the
23560 makefile, *make* shall process the makefile as specified by this section; otherwise, the
23561 behavior of *make* is unspecified.

23562 **.PRECIOUS** Prerequisites of this special target shall not be removed if *make* receives one of the
23563 asynchronous events explicitly described in the ASYNCHRONOUS EVENTS
23564 section. Subsequent occurrences of **.PRECIOUS** shall add to the list of precious
23565 files. If no prerequisites are specified, all targets in the makefile shall be treated as
23566 if specified with **.PRECIOUS**.

23567 XSI **.SCCS_GET** The application shall ensure that this special target is specified without
23568 prerequisites. If this special target is included in a makefile, the commands
23569 specified with this target shall replace the default commands associated with this
23570 special target (see **Default Rules** (on page 2828)). The commands specified with
23571 this target are used to get all SCCS files that are not found in the current directory.

23572 When source files are named in a dependency list, *make* treats them just like any
23573 other target. Because the source file is presumed to be present in the directory,
23574 there is no need to add an entry for it to the makefile. When a target has no
23575 dependencies, but is present in the directory, *make* assumes that that file is up-to-
23576 date. If, however, an SCCS file named `SCCS/s.source_file` is found for a target
23577 `source_file`, *make* does some additional checking to assure that the target is up-to-
23578 date. If the target is missing, or if the SCCS file is newer, *make* automatically issues
23579 the commands specified for the **.SCCS_GET** special target to retrieve the most
23580 recent version. However, if the target is writable by anyone, *make* does not retrieve
23581 a new version.

23582 **.SILENT** Prerequisites of this special target are targets themselves; this shall cause
23583 commands associated with them to not be written to the standard output before
23584 they are executed. Subsequent occurrences of **.SILENT** shall add to the list of
23585 targets with silent commands. If no prerequisites are specified, *make* shall behave
23586 as if the `-s` option had been specified and no commands or touch messages
23587 associated with any target shall be written to standard output.

23588 **.SUFFIXES** Prerequisites of **.SUFFIXES** shall be appended to the list of known suffixes and are
23589 used in conjunction with the inference rules (see **Inference Rules** (on page 2825)).
23590 If **.SUFFIXES** does not have any prerequisites, the list of known suffixes shall be
23591 cleared.

23592 The special targets **.IGNORE**, **.POSIX**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES** shall be specified
23593 without commands.

23594 Targets with names consisting of a leading period followed by the uppercase letters "POSIX"
 23595 and then any other characters are reserved for future standardization. Targets with names
 23596 consisting of a leading period followed by one or more uppercase letters are reserved for
 23597 implementation extensions.

23598 **Macros**

23599 Macro definitions are in the form:

```
23600 string1 = [string2]
```

23601 The macro named *string1* is defined as having the value of *string2*, where *string2* is defined as all
 23602 characters, if any, after the equal sign, up to a comment character ('#') or an unescaped
 23603 <newline> character. Any <blank> characters immediately before or after the equal sign shall be
 23604 ignored.

23605 Applications shall select macro names from the set of characters consisting solely of periods,
 23606 underscores, digits, and alphabets from the portable character set (see the Base Definitions
 23607 volume of IEEE Std. 1003.1-200x, Section 6.1, Portable Character Set). A macro name shall not
 23608 contain an equals sign. Implementations may allow other characters in macro names as
 23609 extensions.

23610 Macros can appear anywhere in the makefile. $\$(string1)$ or $\${string1}$ shall be replaced by
 23611 *string2*, as follows:

- 23612 • Macros in target lines shall be evaluated when the target line is read.
- 23613 • Macros in makefile command lines shall be evaluated when the command is executed.
- 23614 • Macros in the string before the equals sign in a macro definition shall be evaluated when the
 23615 macro assignment is made.
- 23616 • Macros after the equals sign in a macro definition shall not be evaluated until the defined
 23617 macro is used in a rule or command, or before the equals sign in a macro definition.

23618 The parentheses or braces are optional if *string1* is a single character. The macro $\$\$$ shall be
 23619 replaced by the single character '\$'.

23620 The forms $\$(string1[:subst1=[subst2]])$ or $\${string1[:subst1=[subst2]]}$ can be used to replace all
 23621 occurrences of *subst1* with *subst2* when the macro substitution is performed. The *subst1* to be
 23622 replaced shall be recognized when it is a suffix at the end of a word in *string1* (where a *word*, in
 23623 this context, is defined to be a string delimited by the beginning of the line, a <blank> or
 23624 <newline> character).

23625 Macro definitions shall be taken from the following sources, in the following logical order,
 23626 before the makefile(s) are read.

- 23627 1. Macros specified on the *make* utility command line, in the order specified on the command
 23628 line. It is unspecified whether the internal macros defined in **Internal Macros** (on page
 23629 2826) are accepted from this source.
- 23630 2. Macros defined by the *MAKEFLAGS* environment variable, in the order specified in the
 23631 environment variable. It is unspecified whether the internal macros defined in **Internal**
 23632 **Macros** (on page 2826) are accepted from this source.
- 23633 3. The contents of the environment, excluding the *MAKEFLAGS* and *SHELL* variables and
 23634 including the variables with null values.
- 23635 4. Macros defined in the inference rules built into *make*.

23636 Macro definitions from these sources shall not override macro definitions from a lower-
 23637 numbered source. Macro definitions from a single source (for example, the *make* utility
 23638 command line, the *MAKEFLAGS* environment variable, or the other environment variables) shall
 23639 override previous macro definitions from the same source.

23640 Macros defined in the makefile(s) shall override macro definitions that occur before them in the
 23641 makefile(s) and macro definitions from source 4. If the *-e* option is not specified, macros defined
 23642 in the makefile(s) shall override macro definitions from source 3. Macros defined in the
 23643 makefile(s) shall not override macro definitions from source 1 or source 2.

23644 Before the makefile(s) are read, all of the *make* utility command line options (except *-f* and *-p*)
 23645 and *make* utility command line macro definitions (except any for the *MAKEFLAGS* macro), not
 23646 already included in the *MAKEFLAGS* macro, shall be added to the *MAKEFLAGS* macro. Other
 23647 implementation-defined options and macros may also be added to the *MAKEFLAGS* macro. If
 23648 this modifies the value of the *MAKEFLAGS* macro, or, if the *MAKEFLAGS* macro is modified at
 23649 any subsequent time, the *MAKEFLAGS* environment variable shall be modified to match the
 23650 new value of the *MAKEFLAGS* macro.

23651 Before the makefile(s) are read, all of the *make* utility command line macro definitions (except the
 23652 *MAKEFLAGS* macro or the *SHELL* macro) shall be added to the environment of *make*. Other
 23653 implementation-defined variables may also be added to the environment of *make*.

23654 The *SHELL* macro shall be treated specially. It shall be provided by *make* and set to the path
 23655 name of the shell command language interpreter (see *sh* (on page 3060)). The *SHELL*
 23656 environment variable shall not affect the value of the *SHELL* macro. If *SHELL* is defined in
 23657 the makefile or is specified on the command line, it shall replace the original value of the *SHELL*
 23658 macro, but shall not affect the *SHELL* environment variable. Other effects of defining *SHELL* in
 23659 the makefile or on the command line are implementation-defined.

23660 Inference Rules

23661 Inference rules are formatted as follows:

```
23662 target:
23663 <tab>command
23664 [ <tab>command]
23665 ...
23666 line that does not begin with <tab> or #
```

23667 The application shall ensure that the *target* portion is a valid target name (see **Target Rules** (on
 23668 page 2822)) of the form *.s2* or *.s1.s2* (where *.s1* and *.s2* are suffixes that have been given as
 23669 prerequisites of the *.SUFFIXES* special target and *s1* and *s2* do not contain any slashes or
 23670 periods.) If there is only one period in the target, it is a single-suffix inference rule. Targets with
 23671 two periods are double-suffix inference rules. Inference rules can have only one target before the
 23672 colon.

23673 The application shall ensure that the makefile does not specify prerequisites for inference rules;
 23674 no characters other than white space shall follow the colon in the first line, except when creating
 23675 the *empty rule*, described below. Prerequisites are inferred, as described below.

23676 Inference rules can be redefined. A target that matches an existing inference rule shall overwrite
 23677 the old inference rule. An empty rule can be created with a command consisting of simply a
 23678 semicolon (that is, the rule still exists and is found during inference rule search, but since it is
 23679 empty, execution has no effect). The empty rule also can be formatted as follows:

```
23680 rule: ;
```

- 23681 where zero or more <blank> characters separate the colon and semicolon.
- 23682 The *make* utility uses the suffixes of targets and their prerequisites to infer how a target can be
 23683 made up-to-date. A list of inference rules defines the commands to be executed. By default, *make*
 23684 contains a built-in set of inference rules. Additional rules can be specified in the makefile.
- 23685 The special target **.SUFFIXES** contains as its prerequisites a list of suffixes that shall be used by
 23686 the inference rules. The order in which the suffixes are specified defines the order in which the
 23687 inference rules for the suffixes are used. New suffixes shall be appended to the current list by
 23688 specifying a **.SUFFIXES** special target in the makefile. A **.SUFFIXES** target with no prerequisites
 23689 shall clear the list of suffixes. An empty **.SUFFIXES** target followed by a new **.SUFFIXES** list is
 23690 required to change the order of the suffixes.
- 23691 Normally, the user would provide an inference rule for each suffix. The inference rule to update
 23692 a target with a suffix **.s1** from a prerequisite with a suffix **.s2** is specified as a target **.s2.s1**. The
 23693 internal macros provide the means to specify general inference rules (see **Internal Macros**).
- 23694 When no target rule is found to update a target, the inference rules shall be checked. The suffix
 23695 of the target (**.s1**) to be built is compared to the list of suffixes specified by the **.SUFFIXES** special
 23696 targets. If the **.s1** suffix is found in **.SUFFIXES**, the inference rules shall be searched in the order
 23697 defined for the first **.s2.s1** rule whose prerequisite file (**\$*.s2**) exists. If the target is out-of-date
 23698 with respect to this prerequisite, the commands for that inference rule shall be executed.
- 23699 If the target to be built does not contain a suffix and there is no rule for the target, the single
 23700 suffix inference rules shall be checked. The single-suffix inference rules define how to build a
 23701 target if a file is found with a name that matches the target name with one of the single suffixes
 23702 appended. A rule with one suffix **.s2** is the definition of how to build *target* from **target.s2**. The
 23703 other suffix (**.s1**) is treated as null.
- 23704 XSI A tilde ('~') in the above rules refers to an SCCS file in the current directory. Thus, the rule **.c~.o**
 23705 would transform an SCCS C-language source file into an object file (**.o**). Because the **s.** of the
 23706 SCCS files is a prefix, it is incompatible with *make's* suffix point of view. Hence, the '**~**' is a way
 23707 of changing any file reference into an SCCS file reference.
- 23708 **Libraries**
- 23709 If a target or prerequisite contains parentheses, it shall be treated as a member of an archive
 23710 library. For the *lib(member.o)* expression *lib* refers to the name of the archive library and *member.o*
 23711 to the member name. The application shall ensure that the member is an object file with the **.o**
 23712 suffix. The modification time of the expression is the modification time for the member as kept
 23713 in the archive library; see *ar* (on page 2348). The **.a** suffix refers to an archive library. The **.s2.a**
 23714 rule is used to update a member in the library from a file with a suffix **.s2**.
- 23715 **Internal Macros**
- 23716 The *make* utility shall maintain five internal macros that can be used in target and inference rules.
 23717 In order to clearly define the meaning of these macros, some clarification of the terms *target rule*,
 23718 *inference rule*, *target*, and *prerequisite* is necessary.
- 23719 Target rules are specified by the user in a makefile for a particular target. Inference rules are
 23720 user-specified or *make*-specified rules for a particular class of target name. Explicit prerequisites
 23721 are those prerequisites specified in a makefile on target lines. Implicit prerequisites are those
 23722 prerequisites that are generated when inference rules are used. Inference rules are applied to
 23723 implicit prerequisites or to explicit prerequisites that do not have target rules defined for them in
 23724 the makefile. Target rules are applied to targets specified in the makefile.

23725 Before any target in the makefile is updated, each of its prerequisites (both explicit and implicit)
 23726 shall be updated. This shall be accomplished by recursively processing each prerequisite. Upon
 23727 recursion, each prerequisite shall become a target itself. Its prerequisites in turn shall be
 23728 processed recursively until a target is found that has no prerequisites, at which point the
 23729 recursion stops. The recursion then shall back up, updating each target as it goes.

23730 In the definitions that follow, the word *target* refers to one of:

- 23731 • A target specified in the makefile
- 23732 • An explicit prerequisite specified in the makefile that becomes the target when *make*
 23733 processes it during recursion
- 23734 • An implicit prerequisite that becomes a target when *make* processes it during recursion

23735 In the definitions that follow, the word *prerequisite* refers to one of the following:

- 23736 • An explicit prerequisite specified in the makefile for a particular target
- 23737 • An implicit prerequisite generated as a result of locating an appropriate inference rule and
 23738 corresponding file that matches the suffix of the target

23739 The five internal macros are:

23740 **\$@** The **\$@** shall evaluate to the full target name of the current target, or the archive file
 23741 name part of a library archive target. It shall be evaluated for both target and inference
 23742 rules.

23743 For example, in the **.c.a** inference rule, **\$@** represents the out-of-date **.a** file to be built.
 23744 Similarly, in a makefile target rule to build **lib.a** from **file.c**, **\$@** represents the out-of-
 23745 date **lib.a**.

23746 **\$%** The **\$%** macro shall be evaluated only when the current target is an archive library
 23747 member of the form *libname(member.o)*. In these cases, **\$@** shall evaluate to *libname* and
 23748 **\$%** shall evaluates to *member.o*. The **\$%** macro shall be evaluated for both target and
 23749 inference rules.

23750 For example, in a makefile target rule to build **lib.a(file.o)**, **\$%** represents **file.o**, as
 23751 opposed to **\$@**, which represents **lib.a**.

23752 **\$?** The **\$?** macro shall evaluate to the list of prerequisites that are newer than the current
 23753 target. It shall be evaluated for both target and inference rules.

23754 For example, in a makefile target rule to build *prog* from **file1.o**, **file2.o**, and **file3.o**, and
 23755 where *prog* is not out of date with respect to **file1.o**, but is out of date with respect to
 23756 **file2.o** and **file3.o**, **\$?** represents **file2.o** and **file3.o**.

23757 **\$<** In an inference rule, the **\$<** macro shall evaluate to the file name whose existence
 23758 allowed the inference rule to be chosen for the target. In the **.DEFAULT** rule, the **\$<**
 23759 macro shall evaluate to the current target name. The meaning of the **\$<** macro is
 23760 otherwise unspecified.

23761 For example, in the **.c.a** inference rule, **\$<** represents the prerequisite **.c** file.

23762 **\$*** The **\$*** macro shall evaluate to the current target name with its suffix deleted. It shall be
 23763 evaluated at least for inference rules.

23764 For example, in the **.c.a** inference rule, **\$*.o** represents the out-of-date **.o** file that
 23765 corresponds to the prerequisite **.c** file.

23766 Each of the internal macros has an alternative form. When an uppercase 'D' or 'F' is appended
 23767 to any of the macros, the meaning is changed to the *directory part* for 'D' and *file name part* for

23768 'F'. The directory part is the path prefix of the file without a trailing slash; for the current
 23769 directory, the directory part is '.'. When the \$? macro contains more than one prerequisite file
 23770 name, the \$(?D) and \$(?F) (or \${?D} and \${?F}) macros expand to a list of directory name parts
 23771 and file name parts respectively.

23772 For the target *lib(member.o)* and the *s2.a* rule, the internal macros are defined as:

23773 \$< *member.s2*

23774 \$* *member*

23775 \$@ *lib*

23776 \$? *member.s2*

23777 \$% *member.o*

23778 **Default Rules**

23779 The default rules for *make* shall achieve results that are the same as if the following were used.
 23780 Implementations that do not support the C-Language Development Utilities option may omit
 23781 **CC**, **CFLAGS**, **YACC**, **YFLAGS**, **LEX**, **LFLAGS**, **LDFLAGS**, and the *.c*, *.y*, and *.l* inference rules.
 23782 Implementations that do not support FORTRAN may omit **FC**, **FFLAGS**, and the *.f* inference
 23783 rules. Implementations may provide additional macros and rules.

23784 *SPECIAL TARGETS*

23785 XSI .SCCS_GET: sccs \$(SCCSFLAGS) get \$(SCCSGETFLAGS) \$@

23786

23787 XSI .SUFFIXES: .o .c .y .l .a .sh .f .c~ .y~ .l~ .sh~ .f~

23788 **MACROS**

23789 MAKE=make

23790 AR=ar

23791 ARFLAGS=-rv

23792 YACC=yacc

23793 YFLAGS=

23794 LEX=lex

23795 LFLAGS=

23796 LDFLAGS=

23797 CC=c99

23798 CFLAGS=-O

23799 FC=fort77

23800 FFLAGS=-O 1

23801 XSI GET=get

23802 GFLAGS=

23803 SCCSFLAGS=

23804 SCCSGETFLAGS=-s

23805

23806 *SINGLE SUFFIX RULES*

23807 .c:

23808 \$(CC) \$(CFLAGS) \$(LDFLAGS) -o \$@ \$<

23809 .f:

23810 \$(FC) \$(FFLAGS) \$(LDFLAGS) -o \$@ \$<


```

23811     .sh:
23812         cp $< $@
23813         chmod a+x $@

23814 XSI   .c~:
23815         $(GET) $(GFLAGS) -p $< > $*.c
23816         $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $*.c

23817     .f~:
23818         $(GET) $(GFLAGS) -p $< > $*.f
23819         $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.f

23820     .sh~:
23821         $(GET) $(GFLAGS) -p $< > $*.sh
23822         cp $*.sh $@
23823         chmod a+x $@
23824

23825     DOUBLE SUFFIX RULES

23826     .c.o:
23827         $(CC) $(CFLAGS) -c $<

23828     .f.o:
23829         $(FC) $(FFLAGS) -c $<

23830     .y.o:
23831         $(YACC) $(YFLAGS) $<
23832         $(CC) $(CFLAGS) -c y.tab.c
23833         rm -f y.tab.c
23834         mv y.tab.o $@

23835     .l.o:
23836         $(LEX) $(LFLAGS) $<
23837         $(CC) $(CFLAGS) -c lex.yy.c
23838         rm -f lex.yy.c
23839         mv lex.yy.o $@

23840     .y.c:
23841         $(YACC) $(YFLAGS) $<
23842         mv y.tab.c $@

23843     .l.c:
23844         $(LEX) $(LFLAGS) $<
23845         mv lex.yy.c $@

23846 XSI   .c~.o:
23847         $(GET) $(GFLAGS) -p $< > $*.c
23848         $(CC) $(CFLAGS) -c $*.c

23849     .f~.o:
23850         $(GET) $(GFLAGS) -p $< > $*.f
23851         $(FC) $(FFLAGS) -c $*.f

23852     .y~.o:
23853         $(GET) $(GFLAGS) -p $< > $*.y
23854         $(YACC) $(YFLAGS) $*.y
23855         $(CC) $(CFLAGS) -c y.tab.c
23856         rm -f y.tab.c

```

```

23857 mv y.tab.o $@
23858 .l~.o:
23859 $(GET) $(GFLAGS) -p $< > $*.l
23860 $(LEX) $(LFLAGS) $*.l
23861 $(CC) $(CFLAGS) -c lex.yy.c
23862 rm -f lex.yy.c
23863 mv lex.yy.o $@
23864 .y~.c:
23865 $(GET) $(GFLAGS) -p $< > $*.y
23866 $(YACC) $(YFLAGS) $*.y
23867 mv y.tab.c $@
23868 .l~.c:
23869 $(GET) $(GFLAGS) -p $< > $*.l
23870 $(LEX) $(LFLAGS) $*.l
23871 mv lex.yy.c $@
23872
23873 .c.a:
23874 $(CC) -c $(CFLAGS) $<
23875 $(AR) $(ARFLAGS) $@ $*.o
23876 rm -f $*.o
23877 .f.a:
23878 $(FC) -c $(FFLAGS) $<
23879 $(AR) $(ARFLAGS) $@ $*.o
23880 rm -f $*.o

```

23881 EXIT STATUS

23882 When the `-q` option is specified, the *make* utility shall exit with one of the following values:

- 23883 0 Successful completion.
- 23884 1 The target was not up-to-date.
- 23885 >1 An error occurred.

23886 When the `-q` option is not specified, the *make* utility shall exit with one of the following values:

- 23887 0 Successful completion.
- 23888 >0 An error occurred.

23889 CONSEQUENCES OF ERRORS

23890 Default.

23891 APPLICATION USAGE

23892 If there is a source file (such as `./source.c`) and there are two SCCS files corresponding to it
 23893 (`./s.source.c` and `./SCCS/s.source.c`), on XSI-conformant systems *make* uses the SCCS file in the
 23894 current directory. However, users are advised to use the underlying SCCS utilities (*admin*, *delta*,
 23895 *get*, and so on) or the *sccs* utility for all source files in a given directory. If both forms are used for
 23896 a given source file, future developers are very likely to be confused.

23897 It is incumbent upon portable makefiles to specify the `.POSIX` special target in order to
 23898 guarantee that they are not affected by local extensions.

23899 The `-k` and `-S` options are both present so that the relationship between the command line, the
 23900 `MAKEFLAGS` variable, and the makefile can be controlled precisely. If the `k` flag is passed in

23901 *MAKEFLAGS* and a command is of the form:

23902 `$(MAKE) -S foo`

23903 then the default behavior is restored for the child *make*.

23904 When the `-n` option is specified, it is always added to *MAKEFLAGS*. This allows a recursive
23905 *make -n target* to be used to see all of the action that would be taken to update *target*.

23906 Because of widespread historical practice, interpreting a '#' number sign inside a variable as
23907 the start of a comment has the unfortunate side effect of making it impossible to place a number
23908 sign in a variable, thus forbidding something like:

23909 `CFLAGS = "-D COMMENT_CHAR='#'"`

23910 Many historical *make* utilities stop chaining together inference rules when an intermediate target
23911 is nonexistent. For example, it might be possible for a *make* to determine that both *.y.c* and *.c.o*
23912 could be used to convert a *.y* to a *.o*. Instead, in this case, *make* requires the use of a *.y.o* rule.

23913 The best way to provide portable makefiles is to include all of the rules needed in the makefile
23914 itself. The rules provided use only features provided by other parts of this volume of
23915 IEEE Std. 1003.1-200x. The default rules include rules for optional commands in this volume of
23916 IEEE Std. 1003.1-200x. Only rules pertaining to commands that are provided are needed in an
23917 implementation's default set.

23918 Macros used within other macros are evaluated when the new macro is used rather than when
23919 the new macro is defined. Therefore:

23920 `MACRO = value1`
23921 `NEW = $(MACRO)`
23922 `MACRO = value2`

23923 `target:`
23924 `echo $(NEW)`

23925 would produce *value2* and not *value1* since **NEW** was not expanded until it was needed in the
23926 *echo* command line.

23927 Some historical applications have been known to intermix *target_name* and *macro=name* operands
23928 on the command line, expecting that all of the macros are processed before any of the targets are
23929 dealt with. Portable applications do not do this, although some backward compatibility support
23930 may be included in some implementations.

23931 The following characters in file names may give trouble: '=', ':', '\', '\'', and '@'. For
23932 inference rules, the description of `$<` and `$?` seem similar. However, an example shows the
23933 minor difference. In a makefile containing:

23934 `foo.o: foo.h`

23935 if **foo.h** is newer than **foo.o**, yet **foo.c** is older than **foo.o**, the built-in rule to make **foo.o** from
23936 **foo.c** is used, with `$<` equal to **foo.c** and `$?` equal to **foo.h**. If **foo.c** is also newer than **foo.o**, `$<` is
23937 equal to **foo.c** and `$?` is equal to **foo.h foo.c**.

23938 EXAMPLES

23939 1. The following command:

23940 `make`

23941 makes the first target found in the makefile.

- 23942 2. The following command:
- 23943 make junk
- 23944 makes the target **junk**.
- 23945 3. The following makefile says that **pgm** depends on two files, **a.o** and **b.o**, and that they in
- 23946 turn depend on their corresponding source files (**a.c** and **b.c**), and a common file **incl.h**:
- 23947 pgm: a.o b.o
- 23948 c99 a.o b.o -o pgm
- 23949 a.o: incl.h a.c
- 23950 c99 -c a.c
- 23951 b.o: incl.h b.c
- 23952 c99 -c b.c
- 23953 4. An example for making optimized **.o** files from **.c** files is:
- 23954 .c.o:
- 23955 c99 -c -O \$*.c
- 23956 or:
- 23957 .c.o:
- 23958 c99 -c -O \$<
- 23959 5. The most common use of the archive interface follows. Here, it is assumed that the source
- 23960 files are all C-language source:
- 23961 lib: lib(file1.o) lib(file2.o) lib(file3.o)
- 23962 @echo lib is now up-to-date
- 23963 The **.c.a** rule is used to make **file1.o**, **file2.o**, and **file3.o** and insert them into **lib**.
- 23964 The treatment of escaped <newline> characters throughout the makefile is historical
- 23965 practice. For example, the inference rule:
- 23966 .c.o\
23967 :
- 23968 works, and the macro:
- 23969 f= bar baz\
23970 biz
- 23971 a:
23972 echo ==\$f==
- 23973 echoes "==bar baz biz==".
- 23974 If **\$?** were:
- 23975 /usr/include/stdio.h /usr/include/unistd.h foo.h
- 23976 then **\$(?D)** would be:
- 23977 /usr/include /usr/include .
- 23978 and **\$(?F)** would be:
- 23979 stdio.h unistd.h foo.h
- 23980 6. The contents of the built-in rules can be viewed by running:

23981 `make -p -f /dev/null 2>/dev/null`

23982 RATIONALE

23983 The *make* utility described in this volume of IEEE Std. 1003.1-200x is intended to provide the
 23984 means for changing portable source code into executables that can be run on a
 23985 IEEE Std. 1003.1-200x-conforming system. It reflects the most common features present in
 23986 System V and BSD *makes*.

23987 Historically, the *make* utility has been an especially fertile ground for vendor and research
 23988 organization-specific syntax modifications and extensions. Examples include:

- 23989 • Syntax supporting parallel execution (such as from various multiprocessor vendors, GNU,
 23990 and others)
- 23991 • Additional “operators” separating targets and their prerequisites (System V, BSD, and
 23992 others)
- 23993 • Specifying that command lines containing the strings `$(MAKE)` and `$(MAKE)` are executed
 23994 when the `-n` option is specified (GNU and System V)
- 23995 • Modifications of the meaning of internal macros when referencing libraries (BSD and others)
- 23996 • Using a single instance of the shell for all of the command lines of the target (BSD and others)
- 23997 • Allowing spaces as well as tabs to delimit command lines (BSD)
- 23998 • Adding C preprocessor-style “include” and “ifdef” constructs (System V, GNU, BSD, and
 23999 others)
- 24000 • Remote execution of command lines (Sprite and others)
- 24001 • Specifying additional special targets (BSD, System V, and most others)

24002 Additionally, many vendors and research organizations have rethought the basic concepts of
 24003 *make*, creating vastly extended, as well as completely new, syntaxes. Each of these versions of
 24004 *make* fulfills the needs of a different community of users; it is unreasonable for this volume of
 24005 IEEE Std. 1003.1-200x to require behavior that would be incompatible (and probably inferior) to
 24006 historical practice for such a community.

24007 In similar circumstances, when the industry has enough sufficiently incompatible formats as to
 24008 make them irreconcilable, this volume of IEEE Std. 1003.1-200x has followed one or both of two
 24009 courses of action. Commands have been renamed (*cksum*, *echo*, and *pax*) and/or command line
 24010 options have been provided to select the desired behavior (*grep*, *od*, and *pax*).

24011 Because the syntax specified for the *make* utility is, by and large, a subset of the syntaxes
 24012 accepted by almost all versions of *make*, it was decided that it would be counter-productive to
 24013 change the name. And since the makefile itself is a basic unit of portability, it would not be
 24014 completely effective to reserve a new option letter, such as *make -P*, to achieve the portable
 24015 behavior. Therefore, the special target `.POSIX` was added to the makefile, allowing users to
 24016 specify “standard” behavior. This special target does not preclude extensions in the *make* utility,
 24017 nor does it preclude such extensions being used by the makefile specifying the target; it does,
 24018 however, preclude any extensions from being applied that could alter the behavior of previously
 24019 valid syntax; such extensions must be controlled via command line options or new special
 24020 targets. It is incumbent upon portable makefiles to specify the `.POSIX` special target in order to
 24021 guarantee that they are not affected by local extensions.

24022 The portable version of *make* described in this reference page is not intended to be the state-of-
 24023 the-art software generation tool and, as such, some newer and more leading-edge features have
 24024 not been included. An attempt has been made to describe the portable makefile in a manner that
 24025 does not preclude such extensions as long as they do not disturb the portable behavior described

24026 here.

24027 When the `-n` option is specified, it is always added to `MAKEFLAGS`. This allows a recursive
24028 `make -n target` to be used to see all of the action that would be taken to update `target`.

24029 The definition of `MAKEFLAGS` allows both the System V letter string and the BSD command line
24030 formats. The two formats are sufficiently different to allow implementations to support both
24031 without ambiguity.

24032 Early proposals stated that an “unquoted” number sign was treated as the start of a comment.
24033 The `make` utility does not pay any attention to quotes. A number sign starts a comment
24034 regardless of its surroundings.

24035 The text about “other implementation-defined path names may also be tried” in addition to
24036 `./makefile` and `./Makefile` is to allow such extensions as `SCCS/s.Makefile` and other variations.
24037 It was made an implementation-defined requirement (as opposed to unspecified behavior) to
24038 highlight surprising implementations that might select something unexpected like
24039 `/etc/Makefile`. XSI-conformant systems also try `./s.makefile`, `SCCS/s.makefile`, `./s.Makefile`,
24040 and `SCCS/s.Makefile`.

24041 Early proposals contained the macro `NPROC` as a means of specifying that `make` should use `n`
24042 processes to do the work required. While this feature is a valuable extension for many systems, it
24043 is not common usage and could require other non-trivial extensions to makefile syntax. This
24044 extension is not required by this volume of IEEE Std. 1003.1-200x, but could be provided as a
24045 compatible extension. The macro `PARALLEL` is used by some historical systems with essentially
24046 the same meaning (but without using a name that is a common system limit value). It is
24047 suggested that implementors recognize the existing use of `NPROC` and/or `PARALLEL` as
24048 extensions to `make`.

24049 The default rules are based on System V. The default `CC=` value is `c99` instead of `cc` because this
24050 volume of IEEE Std. 1003.1-200x does not standardize the utility named `cc`. Thus, every
24051 conforming application would be required to define `CC=c99` to expect to run. There is no
24052 advantage conferred by the hope that the makefile might hit the “preferred” compiler because
24053 this cannot be guaranteed to work. Also, since the portable makescript can only use the `c99`
24054 options, no advantage is conferred in terms of what the script can do. It is a quality-of-
24055 implementation issue as to whether `c99` is as valuable as `cc`.

24056 The `-d` option to `make` is frequently used to produce debugging information, but is too
24057 implementation-defined to add to this volume of IEEE Std. 1003.1-200x.

24058 The `-p` option is not passed in `MAKEFLAGS` on most historical implementations and to change
24059 this would cause many implementations to break without sufficiently increased portability.

24060 Commands that begin with a plus sign (`' + '`) are executed even if the `-n` option is present. Based
24061 on the GNU version of `make`, the behavior of `-n` when the plus-sign prefix is encountered has
24062 been extended to apply to `-q` and `-t` as well. However, the System V convention of forcing
24063 command execution with `-n` when the command line of a target contains either of the strings
24064 `$(MAKE)` or `${MAKE}` has not been adopted. This functionality appeared in early proposals, but
24065 the danger of this approach was pointed out with the following example of a portion of a
24066 makefile:

```
24067 subdir:  
24068     cd subdir; rm all_the_files; $(MAKE)
```

24069 The loss of the System V behavior in this case is well-balanced by the safety afforded to other
24070 makefiles that were not aware of this situation. In any event, the command line plus-sign prefix
24071 can provide the desired functionality.

24072 The double colon in the target rule format is supported in BSD systems to allow more than one
 24073 target line containing the same target name to have commands associated with it. Since this is
 24074 not functionality described in the SVID or XPG3 it has been allowed as an extension, but not
 24075 mandated.

24076 The default rules are provided with text specifying that the built-in rules shall be the same *as if*
 24077 the listed set were used. The intent is that implementations should be able to use the rules
 24078 without change, but will be allowed to alter them in ways that do not affect the primary
 24079 behavior.

24080 The best way to provide portable makefiles is to include all of the rules needed in the makefile
 24081 itself. The rules provided use only features provided by other portions of this volume of
 24082 IEEE Std. 1003.1-200x. The default rules include rules for optional commands in this volume of
 24083 IEEE Std. 1003.1-200x. Only rules pertaining to commands that are provided are needed in the
 24084 default set of an implementation.

24085 One point of discussion was whether to drop the default rules list from this volume of
 24086 IEEE Std. 1003.1-200x. They provide convenience, but do not enhance portability of applications.
 24087 The prime benefit is in portability of users who wish to type *make command* and have the
 24088 command build from a **command.c** file.

24089 The historical *MAKESHELL* feature was omitted. In some implementations it is used to let a user
 24090 override the shell to be used to run *make* commands. This was confusing; for a portable *make*, the
 24091 shell should be chosen by the makefile writer or specified on the *make* command line and not by
 24092 a user running *make*.

24093 The *make* utilities in most historical implementations process the prerequisites of a target in left-
 24094 to-right order, and the makefile format requires this. It supports the standard idiom used in
 24095 many makefiles that produce yacc programs; for example:

```
24096 foo: y.tab.o lex.o main.o
24097      $(CC) $(CFLAGS) -o $@ t.tab.o lex.o main.o
```

24098 In this example, if *make* chose any arbitrary order, the **lex.o** might not be made with the correct
 24099 **y.tab.h**. Although there may be better ways to express this relationship, it is widely used
 24100 historically. Implementations that desire to update prerequisites in parallel should require an
 24101 explicit extension to *make* or the makefile format to accomplish it, as described previously.

24102 The algorithm for determining a new entry for target rules is partially unspecified. Some
 24103 historical *makes* allow blank, empty, or comment lines within the collection of commands
 24104 marked by leading <tab>s. A conforming makefile must ensure that each command starts with
 24105 a <tab>, but implementations are free to ignore blank, empty, and comment lines without
 24106 triggering the start of a new entry.

24107 The ASYNCHRONOUS EVENTS section includes having SIGTERM and SIGHUP, along with
 24108 the more traditional SIGINT and SIGQUIT, remove the current target unless directed not to do
 24109 so. SIGTERM and SIGHUP were added to parallel other utilities that have historically cleaned
 24110 up their work as a result of these signals. When *make* receives any signal other than SIGQUIT, it
 24111 is required to resend itself the signal it received so that it exits with a status that reflects the
 24112 signal. The results from SIGQUIT are partially unspecified because, on systems that create **core**
 24113 files upon receipt of SIGQUIT, the **core** from *make* would conflict with a core file from the
 24114 command that was running when the SIGQUIT arrived. The main concern was to prevent
 24115 damaged files from appearing up-to-date when *make* is rerun.

24116 The **.PRECIOUS** special target was extended to affect all targets globally (by specifying no
 24117 prerequisites). The **.IGNORE** and **.SILENT** special targets were extended to allow prerequisites;
 24118 it was judged to be more useful in some cases to be able to turn off errors or echoing for a list of

24119 targets than for the entire makefile. These extensions to the *make* in System V were made to
24120 match historical practice from the BSD *make*.

24121 Macros are not exported to the environment of commands to be run. This was never the case in
24122 any historical *make* and would have serious consequences. The environment is the same as the
24123 environment to *make* except that *MAKEFLAGS* and macros defined on the *make* command line
24124 are added.

24125 Some implementations do not use *system()* for all command lines, as required by the portable
24126 makefile format; as a performance enhancement, they select lines without shell metacharacters
24127 for direct execution by *execve()*. There is no requirement that *system()* be used specifically, but
24128 merely that the same results be achieved. The metacharacters typically used to bypass the direct
24129 *execve()* execution have been any of:

24130 = | ^ () ; & < > * ? [] : \$ \ ' " \ \n

24131 The default in some advanced versions of *make* is to group all the command lines for a target and
24132 execute them using a single shell invocation; the System V method is to pass each line
24133 individually to a separate shell. The single-shell method has the advantages in performance and
24134 the lack of a requirement for many continued lines. However, converting to this newer method
24135 has caused portability problems with many historical makefiles, so the behavior with the POSIX
24136 makefile is specified to be the same as that of System V. It is suggested that the special target
24137 *.ONESHELL* be used as an implementation extension to achieve the single-shell grouping for a
24138 target or group of targets.

24139 Novice users of *make* have had difficulty with the historical need to start commands with a
24140 <tab> character. Since it is often difficult to discern differences between <tab> and <space>
24141 characters on terminals or printed listings, confusing bugs can arise. In early proposals, an
24142 attempt was made to correct this problem by allowing leading <blank>s instead of <tab>s.
24143 However, implementors reported many makefiles that failed in subtle ways following this
24144 change, and it is difficult to implement a *make* that unambiguously can differentiate between
24145 macro and command lines. There is extensive historical practice of allowing leading spaces
24146 before macro definitions. Forcing macro lines into column 1 would be a significant backwards-
24147 compatibility problem for some makefiles. Therefore, historical practice was restored.

24148 The System V INCLUDE feature was considered, but not included. This would treat a line that
24149 began in the first column and contained INCLUDE <filename> as an indication to read <filename>
24150 at that point in the makefile. This is difficult to use in a portable way, and it raises concerns
24151 about nesting levels and diagnostics. System V, BSD, GNU, and others have used different
24152 methods for including files.

24153 The System V dynamic dependency feature was not included. It would support:

24154 cat: \$\$@.c

24155 that would expand to;

24156 cat: cat.c

24157 This feature exists only in the new version of System V *make* and, while useful, is not in wide
24158 usage. This means that macros are expanded twice for prerequisites: once at makefile parse time
24159 and once at target update time.

24160 Consideration was given to adding metarules to the POSIX *make*. This would make *%o: %c* the
24161 same as *.c.o:*. This is quite useful and available from some vendors, but it would cause too many
24162 changes to this *make* to support. It would have introduced rule chaining and new substitution
24163 rules. However, the rules for target names have been set to reserve the '%' and '"' characters.
24164 These are traditionally used to implement metarules and quoting of target names, respectively.

- 24165 Implementors are strongly encouraged to use these characters only for these purposes.
- 24166 A request was made to extend the suffix delimiter character from a period to any character. The
24167 metarules feature in newer *makes* solves this problem in a more general way. This volume of
24168 IEEE Std. 1003.1-200x is staying with the more conservative historical definition.
- 24169 The standard output format for the `-p` option is not described because it is primarily a
24170 debugging option and because the format is not generally useful to programs. In historical
24171 implementations the output is not suitable for use in generating makefiles. The `-p` format has
24172 been variable across historical implementations. Therefore, the definition of `-p` was only to
24173 provide a consistently named option for obtaining *make* script debugging information.
- 24174 Some historical implementations have not cleared the suffix list with `-r`.
- 24175 Implementations should be aware that some historical applications have intermixed *target_name*
24176 and *macro=value* operands on the command line, expecting that all of the macros are processed
24177 before any of the targets are dealt with. Portable applications do not do this, but some
24178 backwards-compatibility support may be warranted.
- 24179 Empty inference rules are specified with a semicolon command rather than omitting all
24180 commands, as described in an early proposal. The latter case has no traditional meaning and is
24181 reserved for implementation extensions, such as in GNU *make*.
- 24182 **FUTURE DIRECTIONS**
- 24183 None.
- 24184 **SEE ALSO**
- 24185 *ar*, *c99*, *get*, *lex*, *sh*, *yacc*, the System Interfaces volume of IEEE Std. 1003.1-200x, *system()*
- 24186 **CHANGE HISTORY**
- 24187 First released in Issue 2.
- 24188 **Issue 4**
- 24189 Aligned with the ISO/IEC 9945-2: 1993 standard.
- 24190 **Issue 4, Version 2**
- 24191 Under **Default Rules**, the string `"-G$@"` is deleted from the line referencing *sccs*.
- 24192 **Issue 5**
- 24193 **FUTURE DIRECTIONS** section added.
- 24194 **Issue 6**
- 24195 This utility is now marked as part of the Software Development Utilities option.
- 24196 The Open Group corrigenda item U029/1 has been applied, correcting a typographical error in
24197 the SPECIAL TARGETS section.
- 24198 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from
24199 “otherwise, the home directory of a user of that name is examined” to “otherwise, the value of
24200 *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined”.
- 24201 It is specified whether the command line is related to the makefile or to the *make* command, and
24202 the macro processing rules are updated to align with the IEEE P1003.2b draft standard.
- 24203 The normative text is reworded to avoid use of the term “must” for application requirements.

24204 **NAME**

24205 man — display system documentation

24206 **SYNOPSIS**24207 man [-k] *name*...24208 **DESCRIPTION**

24209 The *man* utility shall write information about each of the *name* operands. If *name* is the name of a
 24210 standard utility, *man* at a minimum shall write a message describing the syntax used by the
 24211 standard utility, its options, and operands. If more information is available, the *man* utility shall
 24212 provide it in an implementation-defined manner.

24213 An implementation may provide information for values of *name* other than the standard utilities.
 24214 Standard utilities that are listed as optional and that are not supported by the implementation
 24215 either shall cause a brief message indicating that fact to be displayed or shall cause a full display
 24216 of information as described previously.

24217 **OPTIONS**

24218 The *man* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 24219 12.2, Utility Syntax Guidelines.

24220 The following option shall be supported:

24221 **-k** Interpret *name* operands as keywords to be used in searching a utilities summary
 24222 database that contains a brief purpose entry for each standard utility and write lines
 24223 from the summary database that match any of the keywords. The keyword search shall
 24224 produce results that are the equivalent of the output of the following command:

```
24225 grep -Ei '  
24226 name  
24227 name  
24228 ...  
24229 ' summary-database
```

24230 This assumes that the *summary-database* is a text file with a single entry per line; this
 24231 organization is not required and the example using *grep -Ei* is merely illustrative of the
 24232 type of search intended. The purpose entry to be included in the database shall consist
 24233 of a terse description of the purpose of the utility.

24234 **OPERANDS**

24235 The following operand shall be supported:

24236 *name* A keyword or the name of a standard utility. When **-k** is not specified and *name*
 24237 does not represent one of the standard utilities, the results are unspecified.

24238 **STDIN**

24239 Not used.

24240 **INPUT FILES**

24241 None.

24242 **ENVIRONMENT VARIABLES**24243 The following environment variables shall affect the execution of *man*:

24244 *LANG* Provide a default value for the internationalization variables that are unset or null.
 24245 If *LANG* is unset or null, the corresponding value from the implementation-
 24246 defined default locale shall be used. If any of the internationalization variables
 24247 contains an invalid setting, the utility shall behave as if none of the variables had
 24248 been defined.

- 24249 **LC_ALL** If set to a non-empty string value, override the values of all the other
24250 internationalization variables.
- 24251 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
24252 characters (for example, single-byte as opposed to multi-byte characters in
24253 arguments and in the summary database). The value of *LC_CTYPE* need not affect
24254 the format of the information written about the name operands.
- 24255 **LC_MESSAGES**
24256 Determine the locale that should be used to affect the format and contents of
24257 diagnostic messages written to standard error and informative messages written to
24258 standard output.
- 24259 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 24260 **PAGER** Determine an output filtering command for writing the output to a terminal. Any
24261 string acceptable as a *command_string* operand to the *sh -c* command shall be valid.
24262 When standard output is a terminal device, the reference page output shall be
24263 piped through the command. If the *PAGER* variable is null or not set, the
24264 command shall be either *more* or another paginator utility documented in the
24265 system documentation.
- 24266 **ASYNCHRONOUS EVENTS**
24267 Default.
- 24268 **STDOUT**
24269 The *man* utility shall write text describing the syntax of the utility *name*, its options and its
24270 operands, or, when *-k* is specified, lines from the summary database. The format of this text is
24271 implementation-defined.
- 24272 **STDERR**
24273 Used only for diagnostic messages.
- 24274 **OUTPUT FILES**
24275 None.
- 24276 **EXTENDED DESCRIPTION**
24277 None.
- 24278 **EXIT STATUS**
24279 The following exit values shall be returned:
24280 0 Successful completion.
24281 >0 An error occurred.
- 24282 **CONSEQUENCES OF ERRORS**
24283 Default.
- 24284 **APPLICATION USAGE**
24285 None.
- 24286 **EXAMPLES**
24287 None.
- 24288 **RATIONALE**
24289 It is recognized that the *man* utility is only of minimal usefulness as specified. The opinion of the
24290 standard developers was strongly divided as to how much or how little information *man* should
24291 be required to provide. They considered, however, that the provision of some portable way of
24292 accessing documentation would aid user portability. The arguments against a fuller

- 24293 specification were:
- 24294 • Large quantities of documentation should not be required on a system that does not have
 - 24295 excess disk space.
 - 24296 • The current manual system does not present information in a manner that greatly aids user
 - 24297 portability.
 - 24298 • A “better help system” is currently an area in which vendors feel that they can add value to
 - 24299 their POSIX implementations.
- 24300 The `-f` option was considered, but due to implementation differences, it was not included in this
- 24301 volume of IEEE Std. 1003.1-200x.
- 24302 The description was changed to be more specific about what has to be displayed for a utility.
- 24303 The standard developers considered it insufficient to allow a display of only the synopsis
- 24304 without giving a short description of what each option and operand does.
- 24305 The “purpose” entry to be included in the database can be similar to the section title (less the
- 24306 numeric prefix) from this volume of IEEE Std. 1003.1-200x for each utility. These titles are
- 24307 similar to those used in historical systems for this purpose.
- 24308 See *mailx* for rationale concerning the default paginator.
- 24309 The caveat in the *LC_CTYPE* description was added because it is not a requirement that an
- 24310 implementation provide reference pages for all of its supported locales on each system;
- 24311 changing *LC_CTYPE* does not necessarily translate the reference page into another language.
- 24312 This is equivalent to the current state of *LC_MESSAGES* in IEEE Std. 1003.1-200x—locale-specific
- 24313 messages are not yet a requirement.
- 24314 The historical *MANPATH* variable is not included in POSIX because no attempt is made to
- 24315 specify naming conventions for reference page files, nor even to mandate that they are files at
- 24316 all. In some systems they could be a true database, a hypertext file, or even fixed strings within
- 24317 the *man* executable. The standard developers considered the portability of reference pages to be
- 24318 outside their scope of work (and more appropriate to the POSIX.7 working group developing
- 24319 application-installation tools). However, users should be aware that *MANPATH* is implemented
- 24320 on a number of historical systems and that it can be used to tailor the search pattern for reference
- 24321 pages from the various categories (utilities, functions, file formats, and so on) when the system
- 24322 administrator reveals the location and conventions for reference pages on the system.
- 24323 The keyword search can rely on at least the text of the section titles from these utility
- 24324 descriptions, and the implementation may add more keywords. The term “section titles” refers
- 24325 to the strings such as:
- 24326 `man` – Display system documentation
- 24327 `ps` – Report process status
- 24328 **FUTURE DIRECTIONS**
- 24329 None.
- 24330 **SEE ALSO**
- 24331 *more*
- 24332 **CHANGE HISTORY**
- 24333 First released in Issue 4.

24334 **Issue 5**

24335 FUTURE DIRECTIONS section added.

24336 **NAME**

24337 mesg — permit or deny messages

24338 **SYNOPSIS**

24339 UP mesg [y|n]

24340

24341 **DESCRIPTION**

24342 The *mesg* utility shall control whether other users are allowed to send messages via *write*, *talk*, or
 24343 other utilities to a terminal device. The terminal device affected shall be determined by searching
 24344 for the first terminal in the sequence of devices associated with standard input, standard output,
 24345 and standard error, respectively. With no arguments, *mesg* shall report the current state without
 24346 changing it. Processes with appropriate privileges may be able to send messages to the terminal
 24347 independent of the current state.

24348 **OPTIONS**

24349 None.

24350 **OPERANDS**

24351 The following operands shall be supported in the POSIX locale:

24352 *y* Grant permission to other users to send messages to the terminal device.24353 *n* Deny permission to other users to send messages to the terminal device.24354 **STDIN**

24355 Not used.

24356 **INPUT FILES**

24357 None.

24358 **ENVIRONMENT VARIABLES**24359 The following environment variables shall affect the execution of *mesg*:

24360 *LANG* Provide a default value for the internationalization variables that are unset or null.
 24361 If *LANG* is unset or null, the corresponding value from the implementation-
 24362 defined default locale shall be used. If any of the internationalization variables
 24363 contains an invalid setting, the utility shall behave as if none of the variables had
 24364 been defined.

24365 *LC_ALL* If set to a non-empty string value, override the values of all the other
 24366 internationalization variables.

24367 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 24368 characters (for example, single-byte as opposed to multi-byte characters in
 24369 arguments).

24370 *LC_MESSAGES*

24371 Determine the locale that should be used to affect the format and contents of
 24372 diagnostic messages written (by *mesg*) to standard error.

24373 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.24374 **ASYNCHRONOUS EVENTS**

24375 Default.

24376 **STDOUT**24377 If no operand is specified, *mesg* shall display the current terminal state in an unspecified format.

24378 **STDERR**

24379 Used only for diagnostic messages.

24380 **OUTPUT FILES**

24381 None.

24382 **EXTENDED DESCRIPTION**

24383 None.

24384 **EXIT STATUS**

24385 The following exit values shall be returned:

24386 0 Receiving messages is allowed.

24387 1 Receiving messages is not allowed.

24388 >1 An error occurred.

24389 **CONSEQUENCES OF ERRORS**

24390 Default.

24391 **APPLICATION USAGE**

24392 The mechanism by which the message status of the terminal is changed is unspecified.
24393 Therefore, unspecified actions may cause the status of the terminal to change after *mesg* has
24394 successfully completed. These actions may include, but are not limited to: another invocation of
24395 the *mesg* utility, login procedures; invocation of the *stty* utility, invocation of the *chmod* utility or
24396 *chmod()* function, and so on.

24397 **EXAMPLES**

24398 None.

24399 **RATIONALE**

24400 The terminal changed by *mesg* is that associated with the standard input, output, or error, rather
24401 than the controlling terminal for the session. This is because users logged in more than once
24402 should be able to change any of their login terminals without having to stop the job running in
24403 those sessions. This is not a security problem involving the terminals of other users because
24404 appropriate privileges would be required to affect the terminal of another user.

24405 The method of checking each of the first three file descriptors in sequence until a terminal is
24406 found was adopted from System V.

24407 The file */dev/tty* is not specified for the terminal device because it was thought to be too
24408 restrictive. Typical environment changes for the *n* operand are that write permissions are
24409 removed for *others* and *group* from the appropriate device. It was decided to leave the actual
24410 description of what is done as unspecified because of potential differences between
24411 implementations.

24412 The format for standard output is unspecified because of differences between historical
24413 implementations. This output is generally not useful to shell scripts (they can use the exit
24414 status), so exact parsing of the output is unnecessary.

24415 **FUTURE DIRECTIONS**

24416 None.

24417 **SEE ALSO**

24418 *talk*, *write*

24419 **CHANGE HISTORY**

24420 First released in Issue 2.

24421 **Issue 4**

24422 Aligned with the ISO/IEC 9945-2: 1993 standard.

24423 **Issue 6**

24424 This utility is now marked as part of the User Portability Utilities option.

24425 **NAME**

24426 mkdir — make directories

24427 **SYNOPSIS**24428 mkdir [-p][-m *mode*] *dir*...24429 **DESCRIPTION**24430 The *mkdir* utility shall create the directories specified by the operands, in the order specified.24431 For each *dir* operand, the *mkdir* utility shall perform actions equivalent to the *mkdir()* function
24432 defined in the System Interfaces volume of IEEE Std. 1003.1-200x, called with the following
24433 arguments:

- 24434 1. The *dir* operand is used as the *path* argument.
- 24435 2. The value of the bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO is used as
24436 the *mode* argument. (If the **-m** option is specified, the *mode* option-argument overrides this
24437 default.)

24438 **OPTIONS**24439 The *mkdir* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
24440 12.2, Utility Syntax Guidelines.

24441 The following options shall be supported:

24442 **-m** *mode* Set the file permission bits of the newly-created directory to the specified *mode*
24443 value. The *mode* option-argument shall be the same as the *mode* operand defined
24444 for the *chmod* utility. In the *symbolic_mode* strings, the *op* characters '+' and '-'
24445 shall be interpreted relative to an assumed initial mode of *a=rwx*; '+' shall add
24446 permissions to the default mode, '-' shall delete permissions from the default
24447 mode.

24448 **-p** Create any missing intermediate path name components.24449 For each *dir* operand that does not name an existing directory, effects equivalent to
24450 those caused by the following command shall occur:

```
24451           mkdir -p -m $(umask -S),u+wX $(dirname dir) &&
24452           mkdir [-m mode] dir
```

24453 where the **-m** *mode* option represents that option supplied to the original
24454 invocation of *mkdir*, if any.24455 Each *dir* operand that names an existing directory shall be ignored without error.24456 **OPERANDS**

24457 The following operand shall be supported:

24458 *dir* A path name of a directory to be created.24459 **STDIN**

24460 Not used.

24461 **INPUT FILES**

24462 None.

24463 **ENVIRONMENT VARIABLES**24464 The following environment variables shall affect the execution of *mkdir*:

24465 *LANG* Provide a default value for the internationalization variables that are unset or null.
24466 If *LANG* is unset or null, the corresponding value from the implementation-
24467 defined default locale shall be used. If any of the internationalization variables

- 24468 contains an invalid setting, the utility shall behave as if none of the variables had
24469 been defined.
- 24470 **LC_ALL** If set to a non-empty string value, override the values of all the other
24471 internationalization variables.
- 24472 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
24473 characters (for example, single-byte as opposed to multi-byte characters in
24474 arguments).
- 24475 **LC_MESSAGES**
24476 Determine the locale that should be used to affect the format and contents of
24477 diagnostic messages written to standard error.
- 24478 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 24479 **ASYNCHRONOUS EVENTS**
- 24480 Default.
- 24481 **STDOUT**
- 24482 Not used.
- 24483 **STDERR**
- 24484 Used only for diagnostic messages.
- 24485 **OUTPUT FILES**
- 24486 None.
- 24487 **EXTENDED DESCRIPTION**
- 24488 None.
- 24489 **EXIT STATUS**
- 24490 The following exit values shall be returned:
- 24491 0 All the specified directories were created successfully or the **-p** option was specified and all
24492 the specified directories now exist.
- 24493 >0 An error occurred.
- 24494 **CONSEQUENCES OF ERRORS**
- 24495 Default.
- 24496 **APPLICATION USAGE**
- 24497 The default file mode for directories is *a=rwx* (777 on most systems) with selected permissions
24498 removed in accordance with the file mode creation mask. For intermediate path name
24499 components created by *mkdir*, the mode is the default modified by *u+wx* so that the
24500 subdirectories can always be created regardless of the file mode creation mask; if different
24501 ultimate permissions are desired for the intermediate directories, they can be changed
24502 afterwards with *chmod*.
- 24503 Note that some of the requested directories may have been created even if an error occurs.
- 24504 **EXAMPLES**
- 24505 None.
- 24506 **RATIONALE**
- 24507 The System V **-m** option was included to control the file mode.
- 24508 The System V **-p** option was included to create any needed intermediate directories and to
24509 complement the functionality provided by *rmdir* for removing directories in the path prefix as
24510 they become empty. Because no error is produced if any path component already exists, the **-p**

24511 option is also useful to ensure that a particular directory exists.

24512 The functionality of *mkdir* is described substantially through a reference to the *mkdir()* function
24513 in the System Interfaces volume of IEEE Std. 1003.1-200x. For example, by default, the mode of
24514 the directory is affected by the file mode creation mask in accordance with the specified
24515 behavior of the *mkdir()* function. In this way, there is less duplication of effort required for
24516 describing details of the directory creation.

24517 **FUTURE DIRECTIONS**

24518 None.

24519 **SEE ALSO**

24520 *rm*, *rmdir*, *umask*, the System Interfaces volume of IEEE Std. 1003.1-200x, *mkdir()*

24521 **CHANGE HISTORY**

24522 First released in Issue 2.

24523 **Issue 4**

24524 Aligned with the ISO/IEC 9945-2:1993 standard.

24525 **Issue 5**

24526 FUTURE DIRECTIONS section added.

24527 **NAME**

24528 mkfifo — make FIFO special files

24529 **SYNOPSIS**24530 mkfifo [-m *mode*] *file*...24531 **DESCRIPTION**24532 The *mkfifo* utility shall create the FIFO special files specified by the operands, in the order
24533 specified.24534 For each *file* operand, the *mkfifo* utility shall perform actions equivalent to the *mkfifo*() function
24535 defined in the System Interfaces volume of IEEE Std. 1003.1-200x, called with the following
24536 arguments:

- 24537 1. The *file* operand is used as the *path* argument.
- 24538 2. The value of the bitwise-inclusive OR of S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP,
24539 S_IROTH, and S_IWOTH is used as the *mode* argument. (If the **-m** option is specified, the
24540 value of the *mkfifo*() *mode* argument is unspecified, but the FIFO shall at no time have
24541 permissions less restrictive than the **-m mode** option-argument.)

24542 **OPTIONS**24543 The *mkfifo* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
24544 12.2, Utility Syntax Guidelines.

24545 The following option shall be supported:

24546 **-m mode** Set the file permission bits of the newly-created FIFO to the specified *mode* value.
24547 The *mode* option-argument shall be the same as the *mode* operand defined for the
24548 *chmod* utility. In the *symbolic_mode* strings, the *op* characters '+' and '-' shall be
24549 interpreted relative to an assumed initial mode of *a=rw*.

24550 **OPERANDS**

24551 The following operand shall be supported:

24552 *file* A path name of the FIFO special file to be created.24553 **STDIN**

24554 Not used.

24555 **INPUT FILES**

24556 None.

24557 **ENVIRONMENT VARIABLES**24558 The following environment variables shall affect the execution of *mkfifo*:

24559 **LANG** Provide a default value for the internationalization variables that are unset or null.
24560 If *LANG* is unset or null, the corresponding value from the implementation-
24561 defined default locale shall be used. If any of the internationalization variables
24562 contains an invalid setting, the utility shall behave as if none of the variables had
24563 been defined.

24564 **LC_ALL** If set to a non-empty string value, override the values of all the other
24565 internationalization variables.

24566 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
24567 characters (for example, single-byte as opposed to multi-byte characters in
24568 arguments).

24569 **LC_MESSAGES**

24570 Determine the locale that should be used to affect the format and contents of

- 24571 diagnostic messages written to standard error.
- 24572 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 24573 **ASYNCHRONOUS EVENTS**
- 24574 Default.
- 24575 **STDOUT**
- 24576 Not used.
- 24577 **STDERR**
- 24578 Used only for diagnostic messages.
- 24579 **OUTPUT FILES**
- 24580 None.
- 24581 **EXTENDED DESCRIPTION**
- 24582 None.
- 24583 **EXIT STATUS**
- 24584 The following exit values shall be returned:
- 24585 0 All the specified FIFO special files were created successfully.
- 24586 >0 An error occurred.
- 24587 **CONSEQUENCES OF ERRORS**
- 24588 Default.
- 24589 **APPLICATION USAGE**
- 24590 None.
- 24591 **EXAMPLES**
- 24592 None.
- 24593 **RATIONALE**
- 24594 This new utility was added to permit shell applications to create FIFO special files.
- 24595 The **-m** option was added to control the file mode, for consistency with the similar functionality provided the *mkdir* utility.
- 24596
- 24597 Early proposals included a **-p** option similar to the *mkdir -p* option that created intermediate directories leading up to the FIFO specified by the final component. This was removed because it is not commonly needed and is not common practice with similar utilities.
- 24598
- 24599
- 24600 The functionality of *mkfifo* is described substantially through a reference to the *mkfifo()* function in the System Interfaces volume of IEEE Std. 1003.1-200x. For example, by default, the mode of the FIFO file is affected by the file mode creation mask in accordance with the specified behavior of the *mkfifo()* function. In this way, there is less duplication of effort required for describing details of the file creation.
- 24601
- 24602
- 24603
- 24604
- 24605 **FUTURE DIRECTIONS**
- 24606 None.
- 24607 **SEE ALSO**
- 24608 *umask*, the System Interfaces volume of IEEE Std. 1003.1-200x, *mkfifo()*
- 24609 **CHANGE HISTORY**
- 24610 First released in Issue 3.

24611 **Issue 4**

24612 Aligned with the ISO/IEC 9945-2: 1993 standard.

24613 NAME

24614 more — display files on a page-by-page basis

24615 SYNOPSIS

24616 UP more [-ceisu][-n *number*][-p *command*][-t *tagstring*][*file* ...]

24617

24618 DESCRIPTION

24619 The *more* utility shall read files and either write them to the terminal on a page-by-page basis or
 24620 filter them to standard output. If standard output is not a terminal device, all input files shall be
 24621 copied to standard output in their entirety, without modification, except as specified for the *-s*
 24622 option. If standard output is a terminal device, the files shall be written a number of lines (one
 24623 screenful) at a time under the control of user commands. See the EXTENDED DESCRIPTION
 24624 section.

24625 Certain block-mode terminals do not have all the capabilities necessary to support the complete
 24626 *more* definition; they are incapable of accepting commands that are not terminated with a
 24627 <newline> character. Implementations that support such terminals shall provide an operating
 24628 mode to *more* in which all commands can be terminated with a <newline> character on those
 24629 terminals. This mode:

- 24630 • Shall be documented in the system documentation
- 24631 • Shall, at invocation, inform the user of the terminal deficiency that requires the <newline>
 24632 character usage and provide instructions on how this warning can be suppressed in future
 24633 invocations
- 24634 • Shall not be required for implementations supporting only fully capable terminals
- 24635 • Shall not affect commands already requiring <newline> characters
- 24636 • Shall not affect users on the capable terminals from using *more* as described in this volume of
 24637 IEEE Std. 1003.1-200x

24638 OPTIONS

24639 The *more* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 24640 12.2, Utility Syntax Guidelines.

24641 The following options shall be supported:

- 24642 **-c** If a screen is to be written that has no lines in common with the current screen, or
 24643 *more* is writing its first screen, *more* does not scroll the screen, but instead redraws
 24644 each line of the screen in turn, from the top of the screen to the bottom. In addition,
 24645 if *more* is writing its first screen, the screen is cleared. This option may be silently
 24646 ignored on devices with insufficient terminal capabilities.
- 24647 **-e** By default, *more* shall exit immediately after writing the last line of the last file in
 24648 the argument list. If the *-e* option is specified:
 - 24649 1. If there is only a single file in the argument list and that file was completely
 24650 displayed on a single screen, *more* shall exit immediately after writing the last
 24651 line of that file.
 - 24652 2. Otherwise, *more* shall exit only after reaching end-of-file on the last file in the
 24653 argument list twice without an intervening operation. See the EXTENDED
 24654 DESCRIPTION section.
- 24655 **-i** Perform pattern matching in searches without regard to case; see the Base
 24656 Definitions volume of IEEE Std. 1003.1-200x, Section 9.2, Regular Expression
 24657 General Requirements .

- 24658 **-n number** Specify the number of lines per screenful. The *number* argument is a positive
24659 decimal integer. The **-n** option shall override any values obtained from any other
24660 source.
- 24661 **-p command** Each time a screen from a new file is displayed or redisplayed (including as a
24662 result of *more* commands; for example, **:p**), execute the *more* command(s) in the
24663 command arguments in the order specified, as if entered by the user after the first
24664 screen has been displayed. No intermediate results shall be displayed (that is, if the
24665 command is a movement to a screen different than the normal first screen, only the
24666 screen resulting from the command shall be displayed.) If any of the commands
24667 fail for any reason, an informational message to this effect shall be written, and no
24668 further commands specified using the **-p** option shall be executed for this file.
- 24669 **-s** Behave as if consecutive empty lines were a single empty line.
- 24670 **-t tagstring** Write the screenful of the file containing the tag named by the *tagstring* argument.
24671 See the *ctags* utility. The tags feature represented by **-t tagstring** and the **:t**
24672 command is optional. It shall be provided on any system that also provides a
24673 conforming implementation of *ctags*; otherwise, the use of **-t** produces undefined
24674 results.
- 24675 The file name resulting from the **-t** option shall be logically added as a prefix to the
24676 list of command line files, as if specified by the user. If the tag named by the
24677 *tagstring* argument is not found, it shall be an error, and *more* shall take no further
24678 action.
- 24679 If the tag specifies a line number, the first line of the display shall contain the
24680 beginning of that line. If the tag specifies a pattern, the first line of the display shall
24681 contain the beginning of the matching text from the first line of the file that
24682 contains that pattern. If the line does not exist in the file or matching text is not
24683 found, an informational message to this effect shall be displayed, and *more* shall
24684 display the default screen as if **-t** had not been specified.
- 24685 If both the **-t tagstring** and **-p command** options are given, the **-t tagstring** shall be
24686 processed first; that is, the file and starting line for the display shall be as specified
24687 by **-t**, and then the **-p more** command shall be executed. If the line (matching text)
24688 specified by the **-t** command does not exist (is not found), no **-p more** command
24689 shall be executed for this file at any time.
- 24690 **-u** Treat a <backspace> character as a printable control character, displayed as an
24691 implementation-defined character sequence (see the EXTENDED DESCRIPTION
24692 section), suppressing backspacing and the special handling that produces
24693 underlined or standout mode text on some terminal types. Also, do not ignore a
24694 <carriage-return> character at the end of a line.

24695 OPERANDS

24696 The following operand shall be supported:

- 24697 **file** A path name of an input file. If no *file* operands are specified, the standard input
24698 shall be used. If a *file* is '-', the standard input shall be read at that point in the
24699 sequence.

24700 STDIN

24701 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.

24702 **INPUT FILES**

24703 The input files being examined shall be text files. If standard output is a terminal, standard error
 24704 shall be used to read commands from the user. If standard output is a terminal, standard error is
 24705 not readable, and command input is needed, *more* may attempt to obtain user commands from
 24706 the controlling terminal (for example, */dev/tty*); otherwise, *more* shall terminate with an error
 24707 indicating that it was unable to read user commands. If standard output is not a terminal, no
 24708 error shall result if standard error cannot be opened for reading.

24709 **ENVIRONMENT VARIABLES**

24710 The following environment variables shall affect the execution of *more*:

24711 *COLUMNS* Override the system-selected horizontal screen size. See the Base Definitions
 24712 volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables for valid
 24713 values and results when it is unset or null.

24714 *EDITOR* Used by the *v* command to select an editor. See the EXTENDED DESCRIPTION
 24715 section.

24716 *LANG* Provide a default value for the internationalization variables that are unset or null.
 24717 If *LANG* is unset or null, the corresponding value from the implementation-
 24718 defined default locale shall be used. If any of the internationalization variables
 24719 contains an invalid setting, the utility shall behave as if none of the variables had
 24720 been defined.

24721 *LC_ALL* If set to a non-empty string value, override the values of all the other
 24722 internationalization variables.

24723 *LC_COLLATE*

24724 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 24725 character collating elements within regular expressions.

24726 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 24727 characters (for example, single-byte as opposed to multi-byte characters in
 24728 arguments and input files) and the behavior of character classes within regular
 24729 expressions.

24730 *LC_MESSAGES*

24731 Determine the locale that should be used to affect the format and contents of
 24732 diagnostic messages written to standard error and informative messages written to
 24733 standard output.

24734 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

24735 *LINES* Override the system-selected vertical screen size, used as the number of lines in a
 24736 screenful. See the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 8,
 24737 Environment Variables for valid values and results when it is unset or null. The *-n*
 24738 option shall take precedence over the *LINES* variable for determining the number
 24739 of lines in a screenful.

24740 *MORE* Determine a string containing options described in the OPTIONS section preceded
 24741 with hyphens and <blank> character-separated as on the command line. Any
 24742 command line options shall be processed after those in the *MORE* variable, as if
 24743 the command line were:

24744 `more $MORE options operands`

24745 The *MORE* variable shall take precedence over the *TERM* and *LINES* variables for
 24746 determining the number of lines in a screenful.

- 24793 • A sequence of *n* <backspace> characters (where *n* is the same as the number of column
24794 positions that the previous character occupies) that appears between two identical printable
24795 characters shall cause the first of those two characters to be written as emboldened text (that
24796 is, visually brighter, standout mode, or inverse-video mode), if the terminal type supports
24797 that, and the second to be discarded. Immediately subsequent occurrences of
24798 <backspace>/character pairs for that same character also shall be discarded. (For example,
24799 the sequence "a\ba\ba\ba" is interpreted as a single emboldened 'a'.)
- 24800 • The *more* utility shall logically discard all other <backspace> characters from the line as well
24801 as the character which precedes them, if any.
- 24802 • A <carriage-return> character at the end of a line shall be ignored, rather than being written
24803 as a non-printable character, as described in the next paragraph.
- 24804 It is implementation-defined how other non-printable characters are written. Implementations
24805 should use the same format that they use for the *ex print* command; see the OPTIONS section
24806 within the *ed* utility. It is unspecified whether a multi-column character shall be separated if it
24807 crosses a logical line boundary; it shall not be discarded. The behavior is unspecified if the
24808 number of columns on the display is less than the number of columns any single character in the
24809 line being displayed would occupy.
- 24810 When each new file is displayed (or redisplayed), *more* shall write the first screen of the file.
24811 Once the initial screen has been written, *more* shall prompt for a user command. If the execution
24812 of the user command results in a screen that has lines in common with the current screen, and
24813 the device has sufficient terminal capabilities, *more* shall scroll the screen; otherwise, it is
24814 unspecified whether the screen is scrolled or redrawn.
- 24815 For all files but the last (including standard input if no file was specified, and for the last file as
24816 well, if the *-e* option was not specified), when *more* has written the last line in the file, *more* shall
24817 prompt for a user command. This prompt shall contain the name of the next file as well as an
24818 indication that *more* has reached end-of-file. If the user command is *f*, <control>-F, <space>, *j*,
24819 <newline>, *d*, <control>-D, or *s*, *more* shall display the next file. Otherwise, if displaying the last
24820 file, *more* shall exit. Otherwise, *more* shall execute the user command specified.
- 24821 Several of the commands described in this section display a previous screen from the input
24822 stream. In the case that text is being taken from a non-rewindable stream, such as a pipe, it is
24823 implementation-defined how much backwards motion is supported. If a command cannot be
24824 executed because of a limitation on backwards motion, an error message to this effect shall be
24825 displayed, the current screen shall not change, and the user shall be prompted for another
24826 command.
- 24827 If a command cannot be performed because there are insufficient lines to display, *more* shall alert
24828 the terminal. If a command cannot be performed because there are insufficient lines to display or
24829 a / command fails: if the input is the standard input, the last screen in the file may be displayed;
24830 otherwise, the current file and screen shall not change, and the user shall be prompted for
24831 another command.
- 24832 The interactive commands in the following sections shall be supported. Some commands can be
24833 preceded by a decimal integer, called *count* in the following descriptions. If not specified with
24834 the command, *count* shall default to 1. In the following descriptions, *pattern* is a basic regular
24835 expression, as described in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 9.3,
24836 Basic Regular Expressions. The term “examine” is historical usage meaning “open the file for
24837 viewing”; for example, *more foo* would be expressed as examining file *foo*. In the following
24838 descriptions, unless otherwise specified, *line* is a logical line in the *more* display, not a line from
24839 the file being examined.

24840 In the following descriptions, the *current position* refers to two things:

- 24841 1. The position of the current line on the screen
- 24842 2. The line number (in the file) of the current line on the screen

24843 Usually, the line on the screen corresponding to the current position is the third line on the
 24844 screen. If this is not possible (there are fewer than three lines to display or this is the first page of
 24845 the file, or it is the last page of the file), then the current position is either the first or last line on
 24846 the screen as described later.

24847 **Help**

24848 *Synopsis:* h

24849 Write a summary of these commands and other implementation-defined commands. The
 24850 behavior shall be as if the *more* utility were executed with the *-e* option on a file that contained
 24851 the summary information. The user shall be prompted as described earlier in this section when
 24852 end-of-file is reached. If the user command is one of those specified to continue to the next file,
 24853 *more* shall return to the file and screen state from which the **h** command was executed.

24854 **Scroll Forward One Screenful**

24855 *Synopsis:* [*count*]f
 24856 [*count*]<control>-F

24857 Scroll forward *count* lines, with a default of one screenful. If *count* is more than the screen size,
 24858 only the final screenful shall be written.

24859 **Scroll Backward One Screenful**

24860 *Synopsis:* [*count*]b
 24861 [*count*]<control>-B

24862 Scroll backward *count* lines, with a default of one screenful (see the *-n* option). If *count* is more
 24863 than the screen size, only the final screenful shall be written.

24864 **Scroll Forward One Line**

24865 *Synopsis:* [*count*]<space>
 24866 [*count*]j
 24867 [*count*]<newline>

24868 Scroll forward *count* lines. The default *count* for the <space> character shall be one screenful; for **j**
 24869 and <newline> character, one line. The entire *count* lines shall be written, even if *count* is more
 24870 than the screen size.

24871 **Scroll Backward One Line**

24872 *Synopsis:* [*count*]k

24873 Scroll backward *count* lines. The entire *count* lines shall be written, even if *count* is more than the
 24874 screen size.

24875 Scroll Forward One Half Screenful

24876 *Synopsis:* [count]d
24877 [count]<control>-D

24878 Scroll forward *count* lines, with a default of one half of the screen size. If *count* is specified, it
24879 shall become the new default for subsequent **d**, <control>-D, and **u** commands.

24880 Skip Forward One Line

24881 *Synopsis:* [count]s

24882 Display the screenful beginning with the line *count* lines after the last line on the current screen.
24883 If *count* would cause the current position to be such that less than one screenful would be
24884 written, the last screenful in the file shall be written.

24885 Scroll Backward One Half Screenful

24886 *Synopsis:* [count]u
24887 [count]<control>-U

24888 Scroll backward *count* lines, with a default of one half of the screen size. If *count* is specified, it
24889 shall become the new default for subsequent **d**, <control>-D, **u**, and <control>-U commands.
24890 The entire *count* lines shall be written, even if *count* is more than the screen size.

24891 Go to Beginning of File

24892 *Synopsis:* [count]g

24893 Display the screenful beginning with line *count*.

24894 Go to End-of-File

24895 *Synopsis:* [count]G

24896 If *count* is specified, display the screenful beginning with the line *count*. Otherwise, display the
24897 last screenful of the file.

24898 Refresh the Screen

24899 *Synopsis:* r
24900 <control>-L

24901 Refresh the screen.

24902 Discard and Refresh

24903 *Synopsis:* R

24904 Refresh the screen, discarding any buffered input. If the current file is non-seekable, buffered
24905 input shall not be discarded and the **R** command is equivalent to the **r** command.

24906 Mark Position

24907 *Synopsis:* *mletter*

24908 Mark the current position with the letter named by *letter*, where *letter* represents the name of one
24909 of the lowercase letters of the portable character set. When a new file is examined, all marks may
24910 be lost.

24911 Return to Mark

24912 *Synopsis:* *'letter*

24913 Return to the position that was previously marked with the letter named by *letter*, making that
24914 line the current position.

24915 Return to Previous Position

24916 *Synopsis:* *''*

24917 Return to the position from which the last large movement command was executed (where a
24918 "large movement" is defined as any movement of more than a screenful of lines). If no such
24919 movements have been made, return to the beginning of the file.

24920 Search Forward for Pattern

24921 *Synopsis:* [*count*]/[!]*pattern*<newline>

24922 Display the screenful beginning with the *count*th line containing the pattern. The search shall
24923 start after the first line currently displayed. The null regular expression (*'/'* followed by a
24924 <newline> character) shall repeat the search using the previous regular expression, with a
24925 default *count*. If the character *'!'* is included, the matching lines shall be those that do not
24926 contain the *pattern*. If no match is found for the *pattern*, a message to that effect shall be
24927 displayed.

24928 Search Backward for Pattern

24929 *Synopsis:* [*count*?][!]*pattern*<newline>

24930 Display the screenful beginning with the *count*th previous line containing the pattern. The
24931 search shall start on the last line before the first line currently displayed. The null regular
24932 expression (*'?'* followed by a <newline> character) shall repeat the search using the previous
24933 regular expression, with a default *count*. If the character *'!'* is included, matching lines shall be
24934 those that do not contain the *pattern*.

24935 If no match is found for the *pattern*, a message to that effect shall be displayed.

24936 Repeat Search

24937 *Synopsis:* [*count*]n

24938 Repeat the previous search for *count*th line containing the last *pattern* (or not containing the last
24939 *pattern*, if the previous search was *"/!"* or *"?!"*).

24940 **Repeat Search in Reverse**24941 *Synopsis:* [count]N24942 Repeat the search in the opposite direction of the previous search for the *count*th line containing
24943 the last *pattern* (or not containing the last *pattern*, if the previous search was `"/!"` or `"?!"`).24944 **Examine New File**24945 *Synopsis:* :e [*filename*]
<newline>24946 Examine a new file. If the *filename* argument is not specified, the current file (see the `:n` and `:p`
24947 commands below) shall be re-examined. The *filename* shall be subjected to the process of shell
24948 word expansions (see Section 2.6 (on page 2244)); if more than a single path name results, the
24949 effects are unspecified. If *filename* is a number sign (`'#'`), the previously examined file shall be
24950 re-examined. If *filename* is not accessible for any reason (including that it is a non-seekable file),
24951 an error message to this effect shall be displayed and the current file and screen shall not change.24952 **Examine Next File**24953 *Synopsis:* [count]:n24954 Examine the next file. If a number *count* is specified, the *count*th next file shall be examined. If
24955 *filename* refers to a non-seekable file, the results are unspecified.24956 **Examine Previous File**24957 *Synopsis:* [count]:p24958 Examine the previous file. If a number *count* is specified, the *count*th previous file shall be
24959 examined. If *filename* refers to a non-seekable file, the results are unspecified.24960 **Go to Tag**24961 *Synopsis:* :t *tagstring*
<newline>24962 If the file containing the tag named by the *tagstring* argument is not the current file, examine the
24963 file, as if the `:e` command was executed with that file as the argument. Otherwise, or in addition,
24964 display the screenful beginning with the tag, as described for the `-t` option (see the OPTIONS
24965 section). If the *ctags* utility is not supported by the system, the use of `:t` produces undefined
24966 results.24967 **Invoke Editor**24968 *Synopsis:* v24969 Invoke an editor to edit the current file being examined. If standard input is being examined, the
24970 results are unspecified. The name of the editor shall be taken from the environment variable
24971 *EDITOR*, or shall default to *vi*. If the last path name component in *EDITOR* is either *vi* or *ex*, the
24972 editor shall be invoked with a `-c linenumber` command line argument, where *linenumber* is the
24973 line number of the physical line containing the logical line currently displayed as the first line of
24974 the screen. It is implementation-defined whether line-setting options are passed to editors other
24975 than *vi* and *ex*.24976 When the editor exits, *more* shall resume with the same file and screen as when the editor was
24977 invoked.

24978 **Display Position**

24979 *Synopsis:* =
 24980 <control>-G

24981 Write a message for which the information references the first byte of the line after the last line of
 24982 the file on the screen. This message shall include the name of the file currently being examined,
 24983 its number relative to the total number of files there are to examine, the physical line number,
 24984 the byte number and the total bytes in the file, and what percentage of the file precedes the
 24985 current position. If *more* is reading from standard input, or the file is shorter than a single screen,
 24986 the line number, the byte number, the total bytes, and the percentage need not be written.

24987 **Quit**

24988 *Synopsis:* q
 24989 :q
 24990 ZZ

24991 Exit *more*.

24992 **EXIT STATUS**

24993 The following exit values shall be returned:

24994 0 Successful completion.
 24995 >0 An error occurred.

24996 **CONSEQUENCES OF ERRORS**

24997 If an error is encountered accessing a file when using the **:n** command, *more* shall attempt to
 24998 examine the next file in the argument list, but the final exit status shall be affected. If an error is
 24999 encountered accessing a file via the **:p** command, *more* shall attempt to examine the previous file
 25000 in the argument list, but the final exit status shall be affected. If an error is encountered accessing
 25001 a file via the **:e** command, *more* shall remain in the current file and the final exit status shall not
 25002 be affected.

25003 **APPLICATION USAGE**

25004 When the standard output is not a terminal, only the **-s** filter-modification option is effective.
 25005 This is based on historical practice. For example, a typical implementation of *man* pipes its
 25006 output through *more -s* to squeeze excess white space for terminal users. When *man* is piped to
 25007 *lp*, however, it is undesirable for this squeezing to happen.

25008 **EXAMPLES**

25009 The **-p** allows arbitrary commands to be executed at the start of each file. Examples are:

25010 *more -p G file1 file2*
 25011 Examine each file starting with its last screenful.

25012 *more -p 100 file1 file2*
 25013 Examine each file starting with line 100 in the current position (usually the third line, so line
 25014 98 would be the first line written).

25015 *more -p /100 file1 file2*
 25016 Examine each file starting with the first line containing the string "100" in the current
 25017 position

25018 **RATIONALE**

25019 The *more* utility, available in BSD and BSD-derived systems, was chosen as the prototype for the
 25020 POSIX file display program since it is more widely available than either the public-domain
 25021 program *less* or than *pg*, a pager provided in System V. The 4.4 BSD *more* is the model for the

25022 features selected; it is almost fully upward-compatible from the 4.3 BSD version in wide use and
 25023 has become more amenable for *vi* users. Several features originally derived from various file
 25024 editors, found in both *less* and *pg*, have been added to this volume of IEEE Std. 1003.1-200x as
 25025 they have proved extremely popular with users.

25026 There are inconsistencies between *more* and *vi* that result from historical practice. For example,
 25027 the single-character commands **h**, **f**, **b**, and <space> are screen movers in *more*, but cursor
 25028 movers in *vi*. These inconsistencies were maintained because the cursor movements are not
 25029 applicable to *more* and the powerful functionality achieved without the use of the control key
 25030 justifies the differences.

25031 The tags interface has been included in a program that is not a text editor because it promotes
 25032 another degree of consistent operation with *vi*. It is conceivable that the paging environment of
 25033 *more* would be superior for browsing source code files in some circumstances.

25034 The operating mode referred to for block-mode terminals effectively adds a <newline> to each
 25035 Synopsis line that currently has none. So, for example, **d**<newline> would page one screenful.
 25036 The mode could be triggered by a command line option, environment variable, or some other
 25037 method. The details are not imposed by this volume of IEEE Std. 1003.1-200x because there are
 25038 so few systems known to support such terminals. Nevertheless, it was considered that all
 25039 systems should be able to support *more* given the exception cited for this small community of
 25040 terminals because, in comparison to *vi*, the cursor movements are few and the command set
 25041 relatively amenable to the optional <newline>s.

25042 Some versions of *more* provide a shell escaping mechanism similar to the *ex* ! command. The
 25043 standard developers did not consider that this was necessary in a paginator, particularly given
 25044 the wide acceptance of multiple window terminals and job control features. (They chose to
 25045 retain such features in the editors and *mailx* because the shell interaction also gives an
 25046 opportunity to modify the editing buffer, which is not applicable to *more*).

25047 The **-p** (position) option replaces the **+** command because of the Utility Syntax Guidelines. In
 25048 early proposals, it took a *pattern* argument, but historical *less* provided the *more* general facility of
 25049 a command. It would have been desirable to use the same **-c** as *ex* and *vi*, but the letter was
 25050 already in use.

25051 The text stating “from a non-rewindable stream ... implementations may limit the amount of
 25052 backwards motion supported” would allow an implementation that permitted no backwards
 25053 motion beyond text already on the screen. It was not possible to require a minimum amount of
 25054 backwards motion that would be effective for all conceivable device types. The implementation
 25055 should allow the user to back up as far as possible, within device and reasonable memory
 25056 allocation constraints.

25057 Historically, non-printable characters were displayed using the ARPA standard mappings,
 25058 which are as follows:

- 25059 1. Printable characters are left alone.
- 25060 2. Control characters less than \177 are represented as followed by the character offset from
 25061 the '@' character in the ASCII map; for example, \007 is represented as 'G'.
- 25062 3. \177 is represented as followed by '? '.

25063 The display of characters having their eighth bit set was less standard. Existing implementations
 25064 use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed characters with their
 25065 eighth bit set as the two characters "M-, " followed by the seven bit display as described
 25066 previously.) The latter probably has the best claim to historical practice because it was used with
 25067 the **-v** option of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

- 25068 No specific display format is required by IEEE Std. 1003.1-200x. Implementations are
25069 encouraged to conform to historic practice in the absence of any strong reason to diverge.
- 25070 **FUTURE DIRECTIONS**
- 25071 None.
- 25072 **SEE ALSO**
- 25073 *ctags, ed, ex, vi*
- 25074 **CHANGE HISTORY**
- 25075 First released in Issue 4.
- 25076 **Issue 5**
- 25077 FUTURE DIRECTIONS section added.
- 25078 **Issue 6**
- 25079 This utility is now marked as part of the User Portability Utilities option.
- 25080 The obsolescent SYNOPSIS is removed.
- 25081 The utility has been extensively reworked for alignment with the IEEE P1003.2b draft standard:
- 25082
- Changes have been made as result of IEEE PASC Interpretations 1003.2 #37 and #109.
 - The *more* utility should be able to handle underlined and emboldened displays of characters that are wider than a single column position.
- 25083
- 25084

25085 **NAME**

25086 mv — move files

25087 **SYNOPSIS**25088 mv [-fi] *source_file target_file*25089 mv [-fi] *source_file... target_file*25090 **DESCRIPTION**

25091 In the first synopsis form, the *mv* utility shall move the file named by the *source_file* operand to
 25092 the *destination* specified by the *target_file*. This first synopsis form is assumed when the final
 25093 operand does not name an existing directory and is not a symbolic link referring to an existing
 25094 directory.

25095 In the second synopsis form, *mv* shall move each file named by a *source_file* operand to a
 25096 *destination* file in the existing directory named by the *target_dir* operand, or referenced if
 25097 *target_dir* is a symbolic link referring to an existing directory. The *destination* path for each
 25098 *source_file* shall be the concatenation of the target directory, a single slash character, and the last
 25099 path name component of the *source_file*. This second form is assumed when the final operand
 25100 names an existing directory.

25101 If any operand specifies an existing file of a type not specified by the System Interfaces volume
 25102 of IEEE Std. 1003.1-200x, the behavior is implementation-defined.

25103 For each *source_file* the following steps shall be taken:

25104 1. If the destination path exists, the *-f* option is not specified, and either of the following
 25105 conditions is true:

25106 a. The permissions of the destination path do not permit writing and the standard input
 25107 is a terminal.

25108 b. The *-i* option is specified.

25109 the *mv* utility shall write a prompt to standard error and read a line from standard input. If
 25110 the response is not affirmative, *mv* shall do nothing more with the current *source_file* and
 25111 go on to any remaining *source_files*.

25112 2. The *mv* utility shall perform actions equivalent to the *rename()* function defined in the
 25113 System Interfaces volume of IEEE Std. 1003.1-200x, called with the following arguments:

25114 a. The *source_file* operand is used as the *old* argument.

25115 b. The destination path is used as the *new* argument.

25116 If this succeeds, *mv* shall do nothing more with the current *source_file* and go on to any
 25117 remaining *source_files*. If this fails for any reasons other than those described for the *errno*
 25118 [EXDEV] in the System Interfaces volume of IEEE Std. 1003.1-200x, *mv* shall write a
 25119 diagnostic message to standard error, do nothing more with the current *source_file*, and go
 25120 on to any remaining *source_files*.

25121 3. If the destination path exists, and it is a file of type directory and *source_file* is not a file of
 25122 type directory, or it is a file not of type directory and *source_file* is a file of type directory,
 25123 *mv* shall write a diagnostic message to standard error, do nothing more with the current
 25124 *source_file*, and go on to any remaining *source_files*.

25125 4. If the destination path exists, *mv* shall attempt to remove it. If this fails for any reason, *mv*
 25126 shall write a diagnostic message to standard error, do nothing more with the current
 25127 *source_file*, and go on to any remaining *source_files*.

25128 5. The file hierarchy rooted in *source_file* shall be duplicated as a file hierarchy rooted in the
 25129 *destination* path. If *source_file* or any of the files below it in the hierarchy are symbolic links,
 25130 the links themselves shall be duplicated, including their contents, rather than any files to
 25131 which they refer. The following characteristics of each file in the file hierarchy shall be
 25132 duplicated:

- 25133 • The time of last data modification and time of last access
- 25134 • The user ID and group ID
- 25135 • The file mode

25136 If the user ID, group ID, or file mode of a regular file cannot be duplicated, the file mode
 25137 bits S_ISUID and S_ISGID shall not be duplicated.

25138 When files are duplicated to another file system, the implementation may require that the
 25139 process invoking *mv* has read access to each file being duplicated.

25140 If the duplication of the file hierarchy fails for any reason, *mv* shall write a diagnostic
 25141 message to standard error, do nothing more with the current *source_file*, and go on to any
 25142 remaining *source_files*.

25143 If the duplication of the file characteristics fails for any reason, *mv* shall write a diagnostic
 25144 message to standard error, but this failure shall not cause *mv* to modify its exit status.

25145 6. The file hierarchy rooted in *source_file* shall be removed. If this fails for any reason, *mv* shall
 25146 write a diagnostic message to the standard error, do nothing more with the current
 25147 *source_file*, and go on to any remaining *source_files*.

25148 OPTIONS

25149 The *mv* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 25150 12.2, Utility Syntax Guidelines.

25151 The following options shall be supported:

- 25152 **-f** Do not prompt for confirmation if the destination path exists. Any previous
 25153 occurrences of the **-i** option is ignored.
- 25154 **-i** Prompt for confirmation if the destination path exists. Any previous occurrences of
 25155 the **-f** option is ignored.

25156 Specifying more than one of the **-f** or **-i** options shall not be considered an error. The last option
 25157 specified shall determine the behavior of *mv*.

25158 OPERANDS

25159 The following operands shall be supported:

- 25160 *source_file* A path name of a file or directory to be moved.
- 25161 *target_file* A new path name for the file or directory being moved.
- 25162 *target_dir* A path name of an existing directory into which to move the input files.

25163 STDIN

25164 Used to read an input line in response to each prompt specified in the STDERR section.
 25165 Otherwise, the standard input shall not be used.

25166 INPUT FILES

25167 The input files specified by each *source_file* operand can be of any file type.

25168 **ENVIRONMENT VARIABLES**

25169 The following environment variables shall affect the execution of *mv*:

25170 *LANG* Provide a default value for the internationalization variables that are unset or null.
 25171 If *LANG* is unset or null, the corresponding value from the implementation-
 25172 defined default locale shall be used. If any of the internationalization variables
 25173 contains an invalid setting, the utility shall behave as if none of the variables had
 25174 been defined.

25175 *LC_ALL* If set to a non-empty string value, override the values of all the other
 25176 internationalization variables.

25177 *LC_COLLATE*

25178 Determine the locale for the behavior of ranges, equivalence classes and multi-
 25179 character collating elements used in the extended regular expression defined for
 25180 the **yesexpr** locale keyword in the *LC_MESSAGES* category.

25181 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 25182 characters (for example, single-byte as opposed to multi-byte characters in
 25183 arguments and input files), the behavior of character classes used in the extended
 25184 regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES*
 25185 category.

25186 *LC_MESSAGES*

25187 Determine the locale for the processing of affirmative responses that should be
 25188 used to affect the format and contents of diagnostic messages written to standard
 25189 error.

25190 *NSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

25191 **ASYNCHRONOUS EVENTS**

25192 Default.

25193 **STDOUT**

25194 Not used.

25195 **STDERR**

25196 Prompts shall be written to the standard error under the conditions specified in the
 25197 *DESCRIPTION* section. The prompts shall contain the *destination* path name, but their format is
 25198 otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.

25199 **OUTPUT FILES**

25200 The output files may be of any file type.

25201 **EXTENDED DESCRIPTION**

25202 None.

25203 **EXIT STATUS**

25204 The following exit values shall be returned:

25205 0 All input files were moved successfully.

25206 >0 An error occurred.

25207 **CONSEQUENCES OF ERRORS**

25208 If the copying or removal of *source_file* is prematurely terminated by a signal or error, *mv* may
 25209 leave a partial copy of *source_file* at the source or destination. The *mv* utility shall not modify
 25210 both *source_file* and the destination path simultaneously; termination at any point shall leave
 25211 either *source_file* or the destination path complete.

25212 APPLICATION USAGE

25213 None.

25214 EXAMPLES

25215 If the current directory contains only files **a** (of any type defined by the System Interfaces
25216 volume of IEEE Std. 1003.1-200x), **b** (also of any type), and a directory **c**:

25217 mv a b c

25218 mv c d

25219 results with the original files **a** and **b** residing in the directory **d** in the current directory.

25220 RATIONALE

25221 Early proposals diverged from the SVID and BSD historical practice in that they required that
25222 when the destination path exists, the **-f** option is not specified, and input is not a terminal, *mv*
25223 fails. This was done for compatibility with *cp*. The current text returns to historical practice. It
25224 should be noted that this is consistent with the *rename()* function defined in the System
25225 Interfaces volume of IEEE Std. 1003.1-200x, which does not require write permission on the
25226 target.

25227 For absolute clarity, paragraph (1), describing the behavior of *mv* when prompting for
25228 confirmation, should be interpreted in the following manner:

```
25229 if (exists AND (NOT f_option) AND
25230     ((not_writable AND input_is_terminal) OR i_option))
```

25231 The **-i** option exists on BSD systems, giving applications and users a way to avoid accidentally
25232 unlinking files when moving others. When the standard input is not a terminal, the 4.3 BSD *mv*
25233 deletes all existing destination paths without prompting, even when **-i** is specified; this is
25234 inconsistent with the behavior of the 4.3 BSD *cp* utility, which always generates an error when
25235 the file is unwritable and the standard input is not a terminal. The standard developers decided
25236 that use of **-i** is a request for interaction, so when the *destination* path exists, the utility takes
25237 instructions from whatever responds to standard input.

25238 The *rename()* function is able to move directories within the same file system. Some historical
25239 versions of *mv* have been able to move directories, but not to a different file system. The
25240 standard developers considered that this was an annoying inconsistency, so this volume of
25241 IEEE Std. 1003.1-200x requires directories to be able to be moved even across file systems. There
25242 is no **-R** option to confirm that moving a directory is actually intended, since such an option was
25243 not required for moving directories in historical practice. Requiring the application to specify it
25244 sometimes, depending on the destination, seemed just as inconsistent. The semantics of the
25245 *rename()* function were preserved as much as possible. For example, *mv* is not permitted to
25246 “rename” files to or from directories, even though they might be empty and removable.

25247 Historic implementations of *mv* did not exit with a non-zero exit status if they were unable to
25248 duplicate any file characteristics when moving a file across file systems, nor did they write a
25249 diagnostic message for the user. The former behavior has been preserved to prevent scripts from
25250 breaking; a diagnostic message is now required, however, so that users are alerted that the file
25251 characteristics have changed.

25252 The exact format of the interactive prompts is unspecified. Only the general nature of the
25253 contents of prompts are specified because implementations may desire more descriptive
25254 prompts than those used on historical implementations. Therefore, an application not using the
25255 **-f** option or using the **-i** option relies on the system to provide the most suitable dialog directly
25256 with the user, based on the behavior specified.

25257 When *mv* is dealing with a single file system and *source_file* is a symbolic link, the link itself is
25258 moved as a consequence of the dependence on the *rename()* functionality, per the

- 25259 DESCRIPTION. Across file systems, this has to be made explicit.
- 25260 **FUTURE DIRECTIONS**
- 25261 None.
- 25262 **SEE ALSO**
- 25263 *cp, ln*
- 25264 **CHANGE HISTORY**
- 25265 First released in Issue 2.
- 25266 **Issue 4**
- 25267 Aligned with the ISO/IEC 9945-2:1993 standard.
- 25268 **Issue 6**
- 25269 The *mv* utility is changed to describe processing of symbolic links as specified in the
- 25270 IEEE P1003.2b draft standard.

25271 NAME

25272 newgrp — change to a new group

25273 SYNOPSIS

25274 UP newgrp [-l][group]

25275

25276 DESCRIPTION

25277 The *newgrp* utility shall create a new shell execution environment with a new real and effective
 25278 group identification. Of the attributes listed in Section 2.13 (on page 2273), the new shell
 25279 execution environment shall retain the working directory, file creation mask, and exported
 25280 variables from the previous environment (that is, open files, traps, unexported variables, alias
 25281 definitions, shell functions, and *set* options may be lost). All other aspects of the process
 25282 environment that are preserved by the *exec* family of functions defined in the System Interfaces
 25283 volume of IEEE Std. 1003.1-200x shall also be preserved by *newgrp*; whether other aspects are
 25284 preserved is unspecified.

25285 A failure to assign the new group identifications (for example, for security or password-related
 25286 reasons) shall not prevent the new shell execution environment from being created.

25287 The *newgrp* utility shall affect the supplemental groups for the process as follows:

- 25288 • On systems where the effective group ID is normally in the supplementary group list (or
 25289 whenever the old effective group ID actually is in the supplementary group list):
 - 25290 — If the new effective group ID is also in the supplementary group list, *newgrp* shall change
 25291 the effective group ID.
 - 25292 — If the new effective group ID is not in the supplementary group list, *newgrp* shall add the
 25293 new effective group ID to the list, if there is room to add it.
- 25294 • On systems where the effective group ID is not normally in the supplementary group list (or
 25295 whenever the old effective group ID is not in the supplementary group list):
 - 25296 — If the new effective group ID is in the supplementary group list, *newgrp* shall delete it.
 - 25297 — If the old effective group ID is not in the supplementary list, *newgrp* shall add it if there is
 25298 room.

25299 **Note:** The System Interfaces volume of IEEE Std. 1003.1-200x does not specify whether the
 25300 effective group ID of a process is included in its supplementary group list.

25301 With no operands, *newgrp* shall change the effective group back to the groups identified in the
 25302 user's user entry, and shall set the list of supplementary groups to that set in the user's group
 25303 database entries.

25304 If a password is required for the specified group, and the user is not listed as a member of that
 25305 group in the group database, the user shall be prompted to enter the correct password for that
 25306 group. If the user is listed as a member of that group, no password is requested. If no password
 25307 is required for the specified group, it is implementation-defined whether users not listed as
 25308 members of that group can change to that group. Whether or not a password is required,
 25309 implementation-defined system accounting or security mechanisms may impose additional
 25310 authorization restrictions that may cause *newgrp* to write a diagnostic message and suppress the
 25311 changing of the group identification.

25312 OPTIONS

25313 The *newgrp* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 25314 12.2, Utility Syntax Guidelines.

25315 The following option shall be supported:

25316 **-l** (The letter ell.) Change the environment to what would be expected if the user
25317 actually logged in again.

25318 OPERANDS

25319 The following operand shall be supported:

25320 *group* A group name from the group database or a non-negative numeric group ID.
25321 Specifies the group ID to which the real and effective group IDs shall be set. If
25322 *group* is a non-negative numeric string and exists in the group database as a group
25323 name (see *getgrnam()*), the numeric group ID associated with that group name
25324 shall be used as the group ID.

25325 STDIN

25326 Not used.

25327 INPUT FILES

25328 The file */dev/tty* shall be used to read a single line of text for password checking, when one is
25329 required.

25330 ENVIRONMENT VARIABLES

25331 The following environment variables shall affect the execution of *newgrp*:

25332 *LANG* Provide a default value for the internationalization variables that are unset or null.
25333 If *LANG* is unset or null, the corresponding value from the implementation-
25334 defined default locale shall be used. If any of the internationalization variables
25335 contains an invalid setting, the utility shall behave as if none of the variables had
25336 been defined.

25337 *LC_ALL* If set to a non-empty string value, override the values of all the other
25338 internationalization variables.

25339 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
25340 characters (for example, single-byte as opposed to multi-byte characters in
25341 arguments).

25342 LC_MESSAGES

25343 Determine the locale that should be used to affect the format and contents of
25344 diagnostic messages written to standard error.

25345 *XSI NLS_PATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

25346 ASYNCHRONOUS EVENTS

25347 Default.

25348 STDOUT

25349 Not used.

25350 STDERR

25351 Used for diagnostic messages and a prompt string for a password, if one is required. Diagnostic
25352 messages may be written in cases where the exit status is not available. See the EXIT STATUS
25353 section.

25354 OUTPUT FILES

25355 None.

25356 **EXTENDED DESCRIPTION**

25357 None.

25358 **EXIT STATUS**

25359 If *newgrp* succeeds in creating a new shell execution environment, whether or not the group
25360 identification was changed successfully, the exit status shall be the exit status of the shell.
25361 Otherwise, the following exit value shall be returned:

25362 >0 An error occurred.

25363 **CONSEQUENCES OF ERRORS**

25364 The invoking shell may terminate.

25365 **APPLICATION USAGE**

25366 There is no convenient way to enter a password into the Group Database. Use of group
25367 passwords is not encouraged, because by their very nature they encourage poor security
25368 practices. Group passwords may disappear in the future.

25369 A common implementation of *newgrp* is that the current shell uses *exec* to overlay itself with
25370 *newgrp*, which in turn overlays itself with a new shell after changing group. On some systems,
25371 however, this may not occur and *newgrp* may be invoked as a subprocess.

25372 The *newgrp* command is intended only for use from an interactive terminal. It does not offer a
25373 useful interface for the support of applications.

25374 The exit status of *newgrp* is generally inapplicable. If *newgrp* is used in a script, in most cases it
25375 successfully invokes a new shell and the rest of the original shell script is bypassed when the
25376 new shell exits. Used interactively, *newgrp* displays diagnostic messages to indicate problems.
25377 But usage such as:

25378 `newgrp foo`25379 `echo $?`

25380 is not useful because the new shell might not have access to any status *newgrp* may have
25381 generated (and most historical systems do not provide this status). A zero status echoed here
25382 does not necessarily indicate that the user has changed to the new group successfully. Following
25383 *newgrp* with the *id* command provides a portable means of determining whether the group
25384 change was successful or not.

25385 **EXAMPLES**

25386 None.

25387 **RATIONALE**

25388 Most historical implementations use one of the *exec* functions to implement the behavior of
25389 *newgrp*. Errors detected before the *exec* leave the environment unchanged, while errors detected
25390 after the *exec* leave the user in a changed environment. While it would be useful to have *newgrp*
25391 issue a diagnostic message to tell the user that the environment changed, it would be
25392 inappropriate to require this change to some historical implementations.

25393 The password mechanism is allowed in the group database, but how this would be
25394 implemented is not specified.

25395 The *newgrp* utility was retained in this volume of IEEE Std. 1003.1-200x, even given the existence
25396 of the multiple group permissions feature in the System Interfaces volume of
25397 IEEE Std. 1003.1-200x, for several reasons. First, in some systems, the group ownership of a
25398 newly created file is determined by the group of the directory in which the file is created, as
25399 allowed by the System Interfaces volume of IEEE Std. 1003.1-200x; on other systems, the group
25400 ownership of a newly created file is determined by the effective group ID. On systems of the
25401 latter type, *newgrp* allows files to be created with a specific group ownership. Finally, many

25402 systems use the real group ID in accounting, and on such systems, *newgrp* allows the accounting
25403 identity of the user to be changed.

25404 **FUTURE DIRECTIONS**

25405 None.

25406 **SEE ALSO**

25407 *sh*, the System Interfaces volume of IEEE Std. 1003.1-200x, *exec*

25408 **CHANGE HISTORY**

25409 First released in Issue 2.

25410 **Issue 4**

25411 Aligned with the ISO/IEC 9945-2: 1993 standard.

25412 The *newgrp* utility is now mandatory; it is optional in Issue 3.

25413 **Issue 6**

25414 This utility is now marked as part of the User Portability Utilities option.

25415 The obsolescent SYNOPSIS is removed.

25416 The text describing supplemental groups is no longer conditional on {NGROUPS_MAX} being
25417 greater than 1. This is because {NGROUPS_MAX} now has a minimum value of 8. This is a FIPS
25418 requirement.

25419 NAME

25420 nice — invoke a utility with an altered nice value

25421 SYNOPSIS

25422 UP nice [-n *increment*] *utility* [*argument...*]

25423

25424 DESCRIPTION

25425 The *nice* utility shall invoke a utility, requesting that it be run with a different nice value (see the
 25426 Base Definitions volume of IEEE Std. 1003.1-200x, Section 3.241, Nice Value). With no options
 25427 and only if the user has appropriate privileges, the executed utility shall be run with a nice value
 25428 that is some implementation-defined quantity less than or equal to the nice value of the current
 25429 process. If the user lacks appropriate privileges to affect the nice value in the requested manner,
 25430 the *nice* utility shall not affect the nice value; in this case, a warning message may be written to
 25431 standard error, but this shall not prevent the invocation of *utility* or affect the exit status.

25432 OPTIONS

25433 The *nice* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 25434 12.2, Utility Syntax Guidelines.

25435 The following option is supported:

25436 **-n *increment*** Specify how the nice value of the executed utility shall be adjusted. The *increment*
 25437 option-argument shall be a positive or negative decimal integer that shall be used
 25438 to modify the nice value of the executed utility in an implementation-defined
 25439 manner.

25440 Positive *increment* values shall cause a lower or unchanged nice value. Negative
 25441 *increment* values may require appropriate privileges and shall cause a higher or
 25442 unchanged nice value.

25443 The nice value shall be bounded in an implementation-defined manner. If the
 25444 requested *increment* would raise or lower the nice value of the executed utility
 25445 beyond implementation-defined limits, then the limit whose value was exceeded
 25446 shall be used.

25447 OPERANDS

25448 The following operands shall be supported:

25449 ***utility*** The name of a utility that is to be invoked. If the *utility* operand names any of the
 25450 special built-in utilities in Section 2.15 (on page 2276), the results are undefined.

25451 ***argument*** Any string to be supplied as an argument when invoking the utility named by the
 25452 *utility* operand.

25453 STDIN

25454 Not used.

25455 INPUT FILES

25456 None.

25457 ENVIRONMENT VARIABLES

25458 The following environment variables shall affect the execution of *nice*:

25459 ***LANG*** Provide a default value for the internationalization variables that are unset or null.
 25460 If *LANG* is unset or null, the corresponding value from the implementation-
 25461 defined default locale shall be used. If any of the internationalization variables
 25462 contains an invalid setting, the utility shall behave as if none of the variables had
 25463 been defined.

- 25464 *LC_ALL* If set to a non-empty string value, override the values of all the other
25465 internationalization variables.
- 25466 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
25467 characters (for example, single-byte as opposed to multi-byte characters in
25468 arguments).
- 25469 *LC_MESSAGES*
25470 Determine the locale that should be used to affect the format and contents of
25471 diagnostic messages written to standard error.
- 25472 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 25473 *PATH* Determine the search path used to locate the utility to be invoked. See the Base
25474 Definitions volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables.
- 25475 **ASYNCHRONOUS EVENTS**
- 25476 Default.
- 25477 **STDOUT**
- 25478 Not used.
- 25479 **STDERR**
- 25480 Used only for diagnostic messages.
- 25481 **OUTPUT FILES**
- 25482 None.
- 25483 **EXTENDED DESCRIPTION**
- 25484 None.
- 25485 **EXIT STATUS**
- 25486 If the *utility* utility is invoked, the exit status of *nice* shall be the exit status of *utility*; otherwise,
25487 the *nice* utility shall exit with one of the following values:
- 25488 1-125 An error occurred in the *nice* utility.
- 25489 126 The utility specified by *utility* was found but could not be invoked.
- 25490 127 The utility specified by *utility* could not be found.
- 25491 **CONSEQUENCES OF ERRORS**
- 25492 Default.
- 25493 **APPLICATION USAGE**
- 25494 The only guaranteed portable uses of this utility are:
- 25495 *nice utility*
25496 Run *utility* with the default lower nice value.
- 25497 *nice -n <positive integer> utility*
25498 Run *utility* with a lower nice value.
- 25499 On some systems they have no discernible effect on the invoked utility and on some others they
25500 are exactly equivalent.
- 25501 Historical systems have frequently supported the *<positive integer>* up to 20. Since there is no
25502 error penalty associated with guessing a number that is too high, users without access to the
25503 system conformance document (to see what limits are actually in place) could use the historical
25504 1 to 20 range or attempt to use very large numbers if the job should be truly low priority.
- 25505 The nice value value of a process can be displayed using the command:

25506 ps -o nice

25507 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if
25508 an error occurs so that applications can distinguish “failure to find a utility” from “invoked
25509 utility exited with an error indication”. The value 127 was chosen because it is not commonly
25510 used for other meanings; most utilities use small values for “normal error conditions” and the
25511 values above 128 can be confused with termination due to receipt of a signal. The value 126 was
25512 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
25513 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
25514 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
25515 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
25516 any other reason.

25517 EXAMPLES

25518 None.

25519 RATIONALE

25520 Due to the text about the limits of the nice value being implementation-defined, *nice* is not
25521 actually required to change the nice value of the executed command; the limits could be zero
25522 differences from the system default, although the implementor is required to document this fact
25523 in the conformance document.

25524 The 4.3 BSD version of *nice* does not check if *increment* is a valid decimal integer. The command
25525 *nice -x utility*, for example, would be treated the same as the command *nice --1 utility*. If the
25526 user does not have appropriate privileges, this results in a “permission denied” error. This is
25527 considered a bug.

25528 When a user without appropriate privileges gives a negative *increment*, System V treats it like
25529 the command *nice -0 utility*, while 4.3 BSD writes a “permission denied” message and does not
25530 run the utility. Neither was considered clearly superior, so the behavior was left unspecified.

25531 The C shell has a built-in version of *nice* that has a different interface from the one described in
25532 this volume of IEEE Std. 1003.1-200x.

25533 The term “utility” is used, rather than “command”, to highlight the fact that shell compound
25534 commands, pipelines, and so on, cannot be used. Special built-ins also cannot be used.
25535 However, “utility” includes user application programs and shell scripts, not just utilities defined
25536 in this volume of IEEE Std. 1003.1-200x.

25537 Historical implementations of *nice* provide a nice value range of 40 or 41 discrete steps, with the
25538 default nice value being the midpoint of that range. By default, they lower the nice value of the
25539 executed utility by 10.

25540 Some historical documentation states that the *increment* value must be within a fixed range. This
25541 is misleading; the valid *increment* values on any invocation are determined by the current
25542 process nice value, which is not always the default.

25543 The definition of nice value is not intended to suggest that all processes in a system have
25544 priorities that are comparable. Scheduling policy extensions such as the realtime priorities in
25545 POSIX.4 make the notion of a single underlying priority for all scheduling policies problematic.
25546 Some systems may implement the *nice*-related features to affect all processes on the system,
25547 others to affect just the general time-sharing activities implied by this volume of
25548 IEEE Std. 1003.1-200x, and others may have no effect at all. Because of the use of
25549 “implementation-defined” in *nice* and *renice*, a wide range of implementation strategies are
25550 possible.

25551 **FUTURE DIRECTIONS**

25552 None.

25553 **SEE ALSO**25554 *renice*25555 **CHANGE HISTORY**

25556 First released in Issue 4.

25557 **Issue 6**

25558 This utility is now marked as part of the User Portability Utilities option.

25559 The obsolescent SYNOPSIS is removed.

25560 NAME

25561 nl — line numbering filter

25562 SYNOPSIS

```
25563 xsl nl [-p][-b type][-d delim][-f type][-h type][-i incr][-l num][-n format]
25564 [-s sep][-v startnum][-w width][file]
25565
```

25566 DESCRIPTION

25567 The *nl* utility shall read lines from the named *file* or the standard input if no *file* is named and
 25568 shall reproduce the lines to standard output. Lines shall be numbered on the left. Additional
 25569 functionality may be provided in accordance with the command options in effect.

25570 The *nl* utility views the text it reads in terms of logical pages. Line numbering is reset at the start
 25571 of each logical page. A logical page consists of a header, a body, and a footer section. Empty
 25572 sections are valid. Different line numbering options are independently available for header,
 25573 body, and footer (for example, no numbering of header and footer lines while numbering blank
 25574 lines only in the body).

25575 The starts of logical page sections are signaled by input lines containing nothing but the
 25576 following delimiter characters:

25577

Line	Start of
\: \: \:	Header
\: \:	Body
\:	Footer

25578

25579

25580

25581 Unless otherwise specified, *nl* assumes the text being read is in a single logical page body.

25582 OPTIONS

25583 The *nl* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
 25584 Utility Syntax Guidelines. Only one file can be named.

25585 The following options shall be supported:

25586 **-b type** Specify which logical page body lines shall be numbered. Recognized *types* and
 25587 their meaning are:

25588 **a** Number all lines.

25589 **t** Number only non-empty lines.

25590 **n** No line numbering.

25591 **pstring** Number only lines that contain the basic regular expression specified in
 25592 *string*.

25593 The default *type* for logical page body is **t** (text lines numbered).

25594 **-d delim** Specify the delimiter characters that indicate the start of a logical page section.
 25595 These can be changed from the default characters "\:" to two user-specified
 25596 characters. If only one character is entered, the second character remains the
 25597 default character ' : '.

25598 **-f type** Specify the same as **b type** except for footer. The default for logical page footer is **n**
 25599 (no lines numbered).

25600 **-h type** Specify the same as **b type** except for header. The default *type* for logical page
 25601 header is **n** (no lines numbered).

- 25602 **-i incr** Specify the increment value used to number logical page lines. The default is 1.
- 25603 **-l num** Specify the number of blank lines to be considered as one. For example, **-l 2** results
25604 in only the second adjacent blank line being numbered (if the appropriate **-h a**,
25605 **-b a**, or **-f a** option is set). The default is 1.
- 25606 **-n format** Specify the line numbering format. Recognized values are: **ln**, left justified, leading
25607 zeros suppressed; **rn**, right justified, leading zeros suppressed; **rz**, right justified,
25608 leading zeros kept. The default *format* is **rn** (right justified).
- 25609 **-p** Specify that numbering should not be restarted at logical page delimiters.
- 25610 **-s sep** Specify the characters used in separating the line number and the corresponding
25611 text line. The default *sep* is a <tab>.
- 25612 **-v startnum** Specify the initial value used to number logical page lines. The default is 1.
- 25613 **-w width** Specify the number of characters to be used for the line number. The default *width*
25614 is 6.

25615 OPERANDS

25616 The following operand shall be supported:

- 25617 *file* A path name of a text file to be line-numbered.

25618 STDIN

25619 The standard input is a text file that is used if no *file* operand is given.

25620 INPUT FILES

25621 The input file named by the *file* operand is a text file.

25622 ENVIRONMENT VARIABLES

25623 The following environment variables shall affect the execution of *nl*:

- 25624 *LANG* Provide a default value for the internationalization variables that are unset or null.
25625 If *LANG* is unset or null, the corresponding value from the implementation-
25626 defined default locale shall be used. If any of the internationalization variables
25627 contains an invalid setting, the utility shall behave as if none of the variables had
25628 been defined.
- 25629 *LC_ALL* If set to a non-empty string value, override the values of all the other
25630 internationalization variables.
- 25631 *LC_COLLATE* Determine the locale for the behavior of ranges, equivalence classes and multi-
25632 character collating elements within regular expressions.
25633
- 25634 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
25635 characters (for example, single-byte as opposed to multi-byte characters in
25636 arguments and input files), the behavior of character classes within regular
25637 expressions, and for deciding which characters are in character class **graph** (for the
25638 **-b t**, **-f t**, and **-h t** options).
- 25639 *LC_MESSAGES* Determine the locale that should be used to affect the format and contents of
25640 diagnostic messages written to standard error.
25641
- 25642 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

25643 **ASYNCHRONOUS EVENTS**

25644 Default.

25645 **STDOUT**

25646 The standard output shall be a text file in the following format:

25647 "%s%s%s", <line number>, <separator>, <input line>

25648 where <line number> is one of the following numeric formats:

25649 %6d When the **rn** format is used (the default; see **-n**).25650 %06d When the **rz** format is used.25651 %-6d When the **ln** format is used.25652 <empty> When line numbers are suppressed for a portion of the page; the <separator> is also
25653 suppressed.25654 In the preceding list, the number 6 is the default width; the **-w** option can change this value.25655 **STDERR**

25656 Used only for diagnostic messages.

25657 **OUTPUT FILES**

25658 None.

25659 **EXTENDED DESCRIPTION**

25660 None.

25661 **EXIT STATUS**

25662 The following exit values shall be returned:

25663 0 Successful completion.

25664 >0 An error occurred.

25665 **CONSEQUENCES OF ERRORS**

25666 Default.

25667 **APPLICATION USAGE**25668 In using the **-d delim** option, care should be taken to escape characters that have special meaning
25669 to the command interpreter.25670 **EXAMPLES**

25671 The command:

25672 nl -v 10 -i 10 -d \!+ file1

25673 numbers *file1* starting at line number 10 with an increment of 10. The logical page delimiter is
25674 "!+". Note that the '!' has to be escaped when using *csh* as a command interpreter because of
25675 its history substitution syntax. For *ksh* and *sh* the escape is not necessary, but does not do any
25676 harm.25677 **RATIONALE**

25678 None.

25679 **FUTURE DIRECTIONS**

25680 None.

25681 **SEE ALSO**25682 *pr*25683 **CHANGE HISTORY**

25684 First released in Issue 2.

25685 **Issue 4**

25686 Format reorganized.

25687 Utility Syntax Guideline support mandated.

25688 Internationalized environment variable support mandated.

25689 **Issue 5**25690 The option `[-f type]` is added to the SYNOPSIS. The option descriptions are presented in
25691 alphabetic order. The description of `-bt` is changed to “Number only non-empty lines”.25692 **Issue 6**25693 The obsolescent behavior allowing the options to be intermingled with the optional *file* operand
25694 is removed.

25695 NAME

25696 nm — write the name list of an object file (DEVELOPMENT)

25697 SYNOPSIS

25698 UP SD XSI nm [-APv][-efox][-g | -u][-t *format*] *file...*

25699

25700 DESCRIPTION

25701 This utility shall be provided on systems that support both the User Portability Utilities option
25702 and the Software Development Utilities option. On other systems it is optional. Certain options
25703 are only available on XSI-conformant systems.

25704 The *nm* utility shall display symbolic information appearing in the object file, executable file or
25705 object-file library named by *file*. If no symbolic information is available for a valid input file, the
25706 *nm* utility shall report that fact, but not consider it an error condition.

25707 XSI The default base used when numeric values are written is unspecified. On XSI-conformant
25708 systems, it shall be decimal.

25709 OPTIONS

25710 The *nm* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
25711 12.2, Utility Syntax Guidelines.

25712 The following options shall be supported:

25713 **-A** Write the full path name or library name of an object on each line.

25714 XSI **-e** Write only external (global) and static symbol information.

25715 XSI **-f** Produce full output. Write redundant symbols (**.text**, **.data**, and **.bss**), normally
25716 suppressed.

25717 **-g** Write only external (global) symbol information.

25718 XSI **-o** Write numeric values in octal (equivalent to **-t o**).

25719 **-P** Write information in a portable output format, as specified in the STDOUT section.

25720 **-t format** Write each numeric value in the specified format. The format shall be dependent
25721 on the single character used as the *format* option-argument:

25722 XSI d The offset is written in decimal (default).

25723 o The offset is written in octal.

25724 x The offset is written in hexadecimal.

25725 **-u** Write only undefined symbols.

25726 **-v** Sort output by value instead of alphabetically.

25727 XSI **-x** Write numeric values in hexadecimal (equivalent to **-t x**).

25728 OPERANDS

25729 The following operand shall be supported:

25730 *file* A path name of an object file, executable file, or object-file library.

25731 STDIN

25732 See the INPUT FILES section.

25733 **INPUT FILES**

25734 The input file shall be an object file, an object-file library whose format is the same as those
 25735 produced by the *ar* utility for link editing, or an executable file. The *nm* utility may accept
 25736 additional implementation-defined object library formats for the input file.

25737 **ENVIRONMENT VARIABLES**

25738 The following environment variables shall affect the execution of *nm*:

25739 *LANG* Provide a default value for the internationalization variables that are unset or null.
 25740 If *LANG* is unset or null, the corresponding value from the implementation-
 25741 defined default locale shall be used. If any of the internationalization variables
 25742 contains an invalid setting, the utility shall behave as if none of the variables had
 25743 been defined.

25744 *LC_ALL* If set to a non-empty string value, override the values of all the other
 25745 internationalization variables.

25746 *LC_COLLATE*

25747 Determine the locale for character collation information for the symbol-name and
 25748 symbol-value collation sequences.

25749 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 25750 characters (for example, single-byte as opposed to multi-byte characters in
 25751 arguments).

25752 *LC_MESSAGES*

25753 Determine the locale that should be used to affect the format and contents of
 25754 diagnostic messages written to standard error.

25755 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

25756 **ASYNCHRONOUS EVENTS**

25757 Default.

25758 **STDOUT**

25759 If symbolic information is present in the input files, then for each file or for each member of an
 25760 archive, the *nm* utility shall write the following information to standard output. By default, the
 25761 format is unspecified, but the output shall be sorted alphabetically by symbol name:

- 25762 • Library or object name, if *-A* is specified
- 25763 • Symbol name
- 25764 • Symbol type, which shall either be one of the following single characters or an
 25765 implementation-defined type represented by a single character:
- 25766 A Global absolute symbol.
- 25767 a Local absolute symbol.
- 25768 B Global “bss” (that is, uninitialized data space) symbol.
- 25769 b Local bss symbol.
- 25770 D Global data symbol.
- 25771 d Local data symbol.
- 25772 T Global text symbol.
- 25773 t Local text symbol.

25774 U Undefined symbol.

25775 • Value of the symbol

25776 • The size associated with the symbol, if applicable

25777 This information may be supplemented by additional information specific to the
25778 implementation.

25779 If the **-P** option is specified, the previous information shall be displayed using the following
25780 portable format. The three versions differ depending on whether **-t d**, **-t o**, or **-t x** was specified,
25781 respectively:

25782 "%s%s %s %d %d\n", <library/object name>, <name>, <type>,
25783 <value>, <size>

25784 "%s%s %s %o %o\n", <library/object name>, <name>, <type>,
25785 <value>, <size>

25786 "%s%s %s %x %x\n", <library/object name>, <name>, <type>,
25787 <value>, <size>

25788 where

25789 <library/object name> shall be formatted as follows:

25790 • If **-A** is not specified, <library/object name> shall be an empty string.

25791 • If **-A** is specified and the corresponding *file* operand does not name a library:

25792 "%s: ", <file>

25793 • If **-A** is specified and the corresponding *file* operand names a library. In this case, <object file>
25794 shall name the object file in the library containing the symbol being described:

25795 "%s[%s]: ", <file>, <object file>

25796 If **-A** is not specified, then if more than one *file* operand is specified or if only one *file* operand is
25797 specified and it names a library, *nm* shall write a line identifying the object containing the
25798 following symbols before the lines containing those symbols, in the form:

25799 • If the corresponding *file* operand does not name a library:

25800 "%s:\n", <file>

25801 • If the corresponding *file* operand names a library; in this case, <object file> shall be the name
25802 of the file in the library containing the following symbols:

25803 "%s[%s]:\n", <file>, <object file>

25804 If **-P** is specified, but **-t** is not, the format shall be as if **-t x** had been specified.

25805 **STDERR**

25806 Used only for diagnostic messages.

25807 **OUTPUT FILES**

25808 None.

25809 **EXTENDED DESCRIPTION**

25810 None.

25811 **EXIT STATUS**

25812 The following exit values shall be returned:

25813 0 Successful completion.

25814 >0 An error occurred.

25815 **CONSEQUENCES OF ERRORS**

25816 Default.

25817 **APPLICATION USAGE**

25818 Mechanisms for dynamic linking make this utility less meaningful when applied to an executable file because a dynamically linked executable may omit numerous library routines that would be found in a statically linked executable.

25821 **EXAMPLES**

25822 None.

25823 **RATIONALE**

25824 Historical implementations of *nm* have used different bases for numeric output and supplied different default types of symbols that were reported. The *-t format* option, similar to that used in *od* and *strings*, can be used to specify the numeric base; *-g* and *-u* can be used to restrict the amount of output or the types of symbols included in the output.

25828 The option list was significantly reduced from that provided by historical implementations.

25829 The *nm* description is a subset of both the System V and BSD *nm* utilities with no specified default output.

25831 It was recognized that mechanisms for dynamic linking make this utility less meaningful when applied to an executable file (because a dynamically linked executable file may omit numerous library routines that would be found in a statically linked executable file), but the value of *nm* during software development was judged to outweigh other limitations.

25835 The compromise of using *-t format* versus using *-d*, *-o*, and other similar options was necessary because of differences in the meaning of *-o* between implementations. The *-o* option from BSD has been provided here as *-A* to avoid confusion with the *-o* from System V (which has been provided here as *-t* and as *-o* on XSI-conformant systems).

25839 The default output format of *nm* is not specified because of differences in historical implementations. The *-P* option was added to allow some type of portable output format. After a comparison of the different formats used in SunOS, BSD, SVR3, and SVR4, it was decided to create one that did not match the current format of any of these four systems. The format devised is easy to parse by humans, easy to parse in shell scripts, and does not need to vary depending on locale (because no English descriptions are included). All of the systems currently have the information available to use this format.

25846 The format given in *nm* STDOUT uses spaces between the fields, which may be any number of <blank>s required to align the columns. The single-character types were selected to match historical practice, and the requirement that implementation additions also be single characters made parsing the information easier for shell scripts.

25850 **FUTURE DIRECTIONS**

25851 None.

25852 **SEE ALSO**

25853 *ar*, *c99*

25854 **CHANGE HISTORY**

25855 First released in Issue 2.

25856 **Issue 4**

25857 Aligned with the ISO/IEC 9945-2:1993 standard.

25858 **Issue 6**

25859 This utility is now marked as supported when both the User Portability Utilities option and the
25860 Software Development Utilities option are supported.

25861 **NAME**

25862 nohup — invoke a utility immune to hangups

25863 **SYNOPSIS**25864 nohup *utility* [*argument...*]25865 **DESCRIPTION**

25866 The *nohup* utility shall invoke the utility named by the *utility* operand with arguments supplied
 25867 as the *argument* operands. At the time the named *utility* is invoked, the SIGHUP signal shall be
 25868 set to be ignored.

25869 If the standard output is a terminal, all output written by the named *utility* to its standard output
 25870 shall be appended to the end of the file **nohup.out** in the current directory. If **nohup.out** cannot
 25871 be created or opened for appending, the output shall be appended to the end of the file
 25872 **nohup.out** in the directory specified by the *HOME* environment variable. If neither file can be
 25873 created or opened for appending, *utility* shall not be invoked. If a file is created, the file's
 25874 permission bits shall be set to S_IRUSR | S_IWUSR.

25875 If the standard error is a terminal, all output written by the named *utility* to its standard error
 25876 shall be redirected to the same file descriptor as the standard output.

25877 **OPTIONS**

25878 None.

25879 **OPERANDS**

25880 The following operands shall be supported:

25881 *utility* The name of a utility that is to be invoked. If the *utility* operand names any of the
 25882 special built-in utilities in Section 2.15 (on page 2276), the results are undefined.

25883 *argument* Any string to be supplied as an argument when invoking the utility named by the
 25884 *utility* operand.

25885 **STDIN**

25886 Not used.

25887 **INPUT FILES**

25888 None.

25889 **ENVIRONMENT VARIABLES**25890 The following environment variables shall affect the execution of *nohup*:

25891 *HOME* Determine the path name of the user's home directory: if the output file **nohup.out**
 25892 cannot be created in the current directory, the *nohup* utility shall use the directory
 25893 named by *HOME* to create the file.

25894 *LANG* Provide a default value for the internationalization variables that are unset or null.
 25895 If *LANG* is unset or null, the corresponding value from the implementation-
 25896 defined default locale shall be used. If any of the internationalization variables
 25897 contains an invalid setting, the utility behave as if none of the variables had been
 25898 defined.

25899 *LC_ALL* If set to a non-empty string value, override the values of all the other
 25900 internationalization variables.

25901 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 25902 characters (for example, single-byte as opposed to multi-byte characters in
 25903 arguments).

- 25904 *LC_MESSAGES*
25905 Determine the locale that should be used to affect the format and contents of
25906 diagnostic messages written to standard error.
- 25907 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 25908 *PATH* Determine the search path that is used to locate the utility to be invoked. See the
25909 Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 8, Environment
25910 Variables.
- 25911 **ASYNCHRONOUS EVENTS**
25912 The *nohup* utility shall take the standard action for all signals except that **SIGHUP** shall be
25913 ignored.
- 25914 **STDOUT**
25915 If the standard output is not a terminal, the standard output of *nohup* shall be the standard
25916 output generated by the execution of the *utility* specified by the operands. Otherwise, nothing
25917 shall be written to the standard output.
- 25918 **STDERR**
25919 If the standard output is a terminal, a message shall be written to the standard error, indicating
25920 the name of the file to which the output is being appended. The name of the file shall be either
25921 **nohup.out** or **\$HOME/nohup.out**.
- 25922 **OUTPUT FILES**
25923 If the standard output is a terminal, all output written by the named *utility* to the standard
25924 output and standard error is appended to the file **nohup.out**, which is created if it does not
25925 already exist.
- 25926 **EXTENDED DESCRIPTION**
25927 None.
- 25928 **EXIT STATUS**
25929 The following exit values shall be returned:
25930 126 The utility specified by *utility* was found but could not be invoked.
25931 127 An error occurred in the *nohup* utility or the utility specified by *utility* could not be
25932 found.
25933 Otherwise, the exit status of *nohup* shall be that of the utility specified by the *utility* operand.
- 25934 **CONSEQUENCES OF ERRORS**
25935 Default.
- 25936 **APPLICATION USAGE**
25937 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if
25938 an error occurs so that applications can distinguish “failure to find a utility” from “invoked
25939 utility exited with an error indication”. The value 127 was chosen because it is not commonly
25940 used for other meanings; most utilities use small values for “normal error conditions” and the
25941 values above 128 can be confused with termination due to receipt of a signal. The value 126 was
25942 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
25943 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
25944 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
25945 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
25946 any other reason.

25947 **EXAMPLES**

25948 It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done by
25949 placing pipelines and command lists in a single file; this file can then be invoked as a utility, and
25950 the *nohup* applies to everything in the file.

25951 Alternatively, the following command can be used to apply *nohup* to a complex command:

```
25952 nohup sh -c 'complex-command-line'
```

25953 **RATIONALE**

25954 The 4.3 BSD version ignores SIGTERM and SIGHUP, and if *./nohup.out* cannot be used, it fails
25955 instead of trying to use *\$HOME/nohup.out*.

25956 The *cs* utility has a built-in version of *nohup* that acts differently from the POSIX Shell and
25957 Utilities *nohup*.

25958 The term *utility* is used, rather than *command*, to highlight the fact that shell compound
25959 commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility*
25960 includes user application programs and shell scripts, not just the standard utilities.

25961 Historical versions of the *nohup* utility use default file creation semantics. Some more recent
25962 versions use the permissions specified here as an added security precaution.

25963 Some historical implementations ignore SIGQUIT in addition to SIGHUP; others ignore
25964 SIGTERM. An early proposal allowed, but did not require, SIGQUIT to be ignored. Several
25965 reviewers objected that *nohup* should only modify the handling of SIGHUP as required by this
25966 volume of IEEE Std. 1003.1-200x.

25967 **FUTURE DIRECTIONS**

25968 None.

25969 **SEE ALSO**

25970 *sh*, the System Interfaces volume of IEEE Std. 1003.1-200x, *signal()*

25971 **CHANGE HISTORY**

25972 First released in Issue 2.

25973 **Issue 4**

25974 Aligned with the ISO/IEC 9945-2:1993 standard.

25975 NAME

25976 od — dump files in various formats

25977 SYNOPSIS

25978 od [-v][-A *address_base*][-j *skip*][-N *count*][-t *type_string*]...
25979 [*file*...]25980 XSI od [-bcdosx][*file*] [[+]offset[.][b]]

25981

25982 DESCRIPTION

25983 The *od* utility shall write the contents of its input files to standard output in a user-specified
25984 format.

25985 OPTIONS

25986 The *od* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
25987 XSI Utility Syntax Guidelines, except that the order of presentation of the **-t** options and the
25988 **-bcdosx** options is significant.

25989 The following options shall be supported:

25990 **-A** *address_base*25991 Specify the input offset base. See the EXTENDED DESCRIPTION section. The
25992 application shall ensure that the *address_base* option-argument is a character. The
25993 characters 'd', 'o', and 'x' specify that the offset base shall be written in
25994 decimal, octal, or hexadecimal, respectively. The character 'n' specifies that the
25995 offset shall not be written.25996 XSI **-b** Interpret bytes in octal. This is equivalent to **-t o1**.25997 XSI **-c** Interpret bytes as characters specified by the current setting of the *LC_CTYPE*
25998 category. Certain non-graphic characters appear as C escapes: "NUL=\0",
25999 "BS=\b", "FF=\f", "NL=\n", "CR=\r", "HT=\t"; others appear as 3-digit octal
26000 numbers.26001 XSI **-d** Interpret *words* (two-byte units) in unsigned decimal. This is equivalent to **-t u2**.26002 **-j** *skip* Jump over *skip* bytes from the beginning of the input. The *od* utility shall read or
26003 seek past the first *skip* bytes in the concatenated input files. If the combined input
26004 is not at least *skip* bytes long, the *od* utility shall write a diagnostic message to
26005 standard error and exit with a non-zero exit status.26006 By default, the *skip* option-argument shall be interpreted as a decimal number.
26007 With a leading "0x" or "0X", the offset shall be interpreted as a hexadecimal
26008 number; otherwise, with a leading '0', the offset shall be interpreted as an octal
26009 number. Appending the character 'b', 'k', or 'm' to offset shall cause it to be
26010 MAN interpreted as a multiple of 512, 1024, or 1048576 bytes, respectively. If the *skip*
26011 number is hexadecimal, any appended 'b' shall be considered to be the final
26012 hexadecimal digit.26013 **-N** *count* Format no more than *count* bytes of input. By default, *count* shall be interpreted as
26014 a decimal number. With a leading "0x" or "0X", *count* shall be interpreted as a
26015 hexadecimal number; otherwise, with a leading '0', it shall be interpreted as an
26016 octal number. If *count* bytes of input (after successfully skipping, if **-j** *skip*
26017 is specified) are not available, it shall not be considered an error; the *od* utility shall
26018 format the input that is available.26019 XSI **-o** Interpret *words* (two-byte units) in octal. This is equivalent to **-t o2**.

- 26020 XSI **-s** Interpret *words* (two-byte units) in signed decimal. This is equivalent to **-t d2**.
- 26021 **-t *type_string***
- 26022 Specify one or more output types. See the EXTENDED DESCRIPTION section. The
- 26023 application shall ensure that the *type_string* option-argument is a string specifying
- 26024 the types to be used when writing the input data. The string shall consist of the
- 26025 type specification characters **a**, **c**, **d**, **f**, **o**, **u**, and **x**, specifying named character,
- 26026 character, signed decimal, floating point, octal, unsigned decimal, and
- 26027 hexadecimal, respectively. The type specification characters **d**, **f**, **o**, **u**, and **x** can be
- 26028 followed by an optional unsigned decimal integer that specifies the number of
- 26029 bytes to be transformed by each instance of the output type. The type specification
- 26030 character **f** can be followed by an optional **F**, **D**, or **L** indicating that the conversion
- 26031 should be applied to an item of type **float**, **double**, or **long double**, respectively.
- 26032 The type specification characters **d**, **o**, **u** and **x** can be followed by an optional **C**, **S**,
- 26033 **I**, or **L** indicating that the conversion should be applied to an item of type **char**,
- 26034 **short**, **int**, or **long**, respectively. Multiple types can be concatenated within the
- 26035 same *type_string* and multiple **-t** options can be specified. Output lines shall be
- 26036 written for each type specified in the order in which the type specification
- 26037 characters are specified.
- 26038 **-v** Write all input data. Without the **-v** option, any number of groups of output lines,
- 26039 which would be identical to the immediately preceding group of output lines
- 26040 (except for the byte offsets), shall be replaced with a line containing only an
- 26041 asterisk (**' * '**).
- 26042 XSI **-x** Interpret *words* (two-byte units) in hexadecimal. This is equivalent to **-t x2**.
- 26043 XSI Multiple types can be specified by using multiple **-bcdostx** options. Output lines are written for
- 26044 each type specified in the order in which the types are specified.
- 26045 **OPERANDS**
- 26046 The following operands shall be supported:
- 26047 **file** A path name of a file to be read. If no *file* operands are specified, the standard
- 26048 input shall be used. If the first character of *file* is a plus sign (**' + '**) or the first
- 26049 character of the first *file* operand is numeric, no more than two operands are given,
- 26050 XSI and none of the **-A**, **-j**, **-N**, or **-t** options is specified, the results are unspecified.
- 26051 On XSI-conformant systems, the operand shall be assumed to be an *offset*.
- 26052 XSI **[+]*offset*[.][**b**]**
- 26053 The *offset* operand specifies the offset in the file where dumping is to commence.
- 26054 This operand is normally interpreted as octal bytes. If **' . '** is appended, the offset
- 26055 shall be interpreted in decimal. If **' b '** is appended, the offset shall be interpreted
- 26056 in units of 512 bytes. If the *file* argument is omitted, and none of the **-A**, **-j**, **-N**, or
- 26057 **-t** options is specified, the application shall ensure that the *offset* argument is
- 26058 preceded by **' + '**.
- 26059 **STDIN**
- 26060 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
- 26061 section.
- 26062 **INPUT FILES**
- 26063 The input files can be any file type.

26064 **ENVIRONMENT VARIABLES**

26065 The following environment variables shall affect the execution of *od*:

26066 *LANG* Provide a default value for the internationalization variables that are unset or null.
 26067 If *LANG* is unset or null, the corresponding value from the implementation-
 26068 defined default locale shall be used. If any of the internationalization variables
 26069 contains an invalid setting, the utility shall behave as if none of the variables had
 26070 been defined.

26071 *LC_ALL* If set to a non-empty string value, override the values of all the other
 26072 internationalization variables.

26073 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 26074 characters (for example, single-byte as opposed to multi-byte characters in
 26075 arguments and input files).

26076 *LC_MESSAGES*
 26077 Determine the locale that should be used to affect the format and contents of
 26078 diagnostic messages written to standard error.

26079 *LC_NUMERIC*
 26080 Determine the locale for selecting the radix character used when writing floating-
 26081 point formatted output.

26082 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

26083 **ASYNCHRONOUS EVENTS**

26084 Default.

26085 **STDOUT**

26086 See the EXTENDED DESCRIPTION section.

26087 **STDERR**

26088 Used only for diagnostic messages.

26089 **OUTPUT FILES**

26090 None.

26091 **EXTENDED DESCRIPTION**

26092 The *od* utility shall copy sequentially each input file to standard output, transforming the input
 26093 XSI data according to the output types specified by the *-t* options or the *-bcdosx* options. If no
 26094 output type is specified, the default output shall be as if *-t oS* had been specified.

26095 The number of bytes transformed by the output type specifier *c* may be variable depending on
 26096 the *LC_CTYPE* category.

26097 The default number of bytes transformed by output type specifiers *d*, *f*, *o*, *u*, and *x* corresponds
 26098 to the various C-language types as follows. If the *c99* compiler is present on the system, these
 26099 specifiers shall correspond to the sizes used by default in that compiler. Otherwise, these sizes
 26100 may vary among systems that conform to IEEE Std. 1003.1-200x.

- 26101 • For the type specifier characters *d*, *o*, *u*, and *x*, the default number of bytes shall correspond
- 26102 to the size of the underlying implementation's basic integer type. For these specifier
- 26103 characters, the implementation shall support values of the optional number of bytes to be
- 26104 converted corresponding to the number of bytes in the C-language types **char**, **short**, **int**, and
- 26105 **long**. These numbers can also be specified by an application as the characters 'C', 'S', 'I',
- 26106 and 'L', respectively.

26107 **Notes to Reviewers**26108 *This section with side shading will not appear in the final copy. - Ed.*26109 D3, XCU, ERN 99: We need to address long long, which usually uses the notation LL;
26110 however, that is 2 characters. Do we need to invent a new single character notation for long
26111 long?26112 The implementation shall also support the values 1, 2, and 4, even if it provides no C-
26113 Language types of those sizes. The byte order used when interpreting numeric values is
26114 implementation-defined, but shall correspond to the order in which a constant of the
26115 corresponding type is stored in memory on the system.

- 26116 • For the type specifier character
- f**
- , the default number of bytes shall correspond to the number
-
- 26117 of bytes in the underlying implementation's basic double precision floating-point data type.
-
- 26118 The implementation shall support values of the optional number of bytes to be converted
-
- 26119 corresponding to the number of bytes in the C-language types
- float**
- ,
- double**
- , and
- long**
-
- 26120
- double**
- . These numbers can also be specified by an application as the characters 'F', 'D',
-
- 26121 and 'L', respectively.

26122 The type specifier character **a** specifies that bytes are interpreted as named characters from the
26123 International Reference Version (IRV) of the ISO/IEC 646:1991 standard. Only the least
26124 significant seven bits of each byte shall be used for this type specification. Bytes with the values
26125 listed in the following table shall be written using the corresponding names for those characters.26126 **Table 4-12** Named Characters in *od*

Value	Name	Value	Name	Value	Name	Value	Name
\000	nul	\001	soh	\002	stx	\003	etx
\004	eot	\005	enq	\006	ack	\007	bel
\010	bs	\011	ht	\012	lf or nl*	\013	vt
\014	ff	\015	cr	\016	so	\017	si
\020	dle	\021	dc1	\022	dc2	\023	dc3
\024	dc4	\025	nak	\026	syn	\027	etb
\030	can	\031	em	\032	sub	\033	esc
\034	fs	\035	gs	\036	rs	\037	us
\040	sp	\177	del				

26138 **Note:** The "\012" value may be written either as **lf** or **nl**.26139 The type specifier character **c** specifies that bytes are interpreted as characters specified by the
26140 current setting of the *LC_CTYPE* locale category. Characters listed in the table in the Base
26141 Definitions volume of IEEE Std. 1003.1-200x, Chapter 5, File Format Notation ("\\", '\a',
26142 '\b', '\f', '\n', '\r', '\t', '\v') shall be written as the corresponding escape sequences,
26143 except that backslash shall be written as a single backslash and a NUL shall be written as '\0'.
26144 Other non-printable characters shall be written as one three-digit octal number for each byte in
26145 the character. If the size of a byte on the system is greater than nine bits, the format used for
26146 non-printable characters is implementation-defined. Printable multi-byte characters shall be
26147 written in the area corresponding to the first byte of the character; the two-character sequence
26148 "***" shall be written in the area corresponding to each remaining byte in the character, as an
26149 indication that the character is continued. When either the **-j skip** or **-N count** option is specified
26150 along with the **c** type specifier, and this results in an attempt to start or finish in the middle of a
26151 multi-byte character, the result is implementation-defined.26152 The input data shall be manipulated in blocks, where a block is defined as a multiple of the least
26153 common multiple of the number of bytes transformed by the specified output types. If the least

26154 common multiple is greater than 16, the results are unspecified. Each input block shall be
 26155 written as transformed by each output type, one per written line, in the order that the output
 26156 types were specified. If the input block size is larger than the number of bytes transformed by
 26157 the output type, the output type shall sequentially transform the parts of the input block, and
 26158 the output from each of the transformations shall be separated by one or more <blank>
 26159 characters.

26160 If, as a result of the specification of the `-N` option or end-of-file being reached on the last input
 26161 file, input data only partially satisfies an output type, the input shall be extended sufficiently
 26162 with null bytes to write the last byte of the input.

26163 Unless `-A n` is specified, the first output line produced for each input block shall be preceded by
 26164 the input offset, cumulative across input files, of the next byte to be written. The format of the
 26165 input offset is unspecified; however, it shall not contain any <blank> characters, shall start at the
 26166 first character of the output line, and shall be followed by one or more <blank> characters. In
 26167 addition, the offset of the byte following the last byte written shall be written after all the input
 26168 data has been processed, but shall not be followed by any <blank> characters.

26169 If no `-A` option is specified, the input offset base is unspecified.

26170 EXIT STATUS

26171 The following exit values shall be returned:

26172 0 All input files were processed successfully.

26173 >0 An error occurred.

26174 CONSEQUENCES OF ERRORS

26175 Default.

26176 APPLICATION USAGE

26177 Applications are warned not to use file names starting with `'+'` or a first operand starting with
 26178 a numeric character so that the old functionality can be maintained by implementations, unless
 26179 they specify one of the new options specified by the ISO/IEC 9945-2:1993 standard. To
 26180 guarantee that one of these file names is always interpreted as a file name, an application could
 26181 always specify the address base format with the `-A` option.

26182 EXAMPLES

26183 If a file containing 128 bytes with decimal values zero to 127, in increasing order, is supplied as
 26184 standard input to the command:

```
26185 od -A d -t a
```

26186 on an implementation using an input block size of 16 bytes, the standard output, independent of
 26187 the current locale setting, would be similar to:

```
26188 0000000 nul soh stx etx eot enq ack bel bs ht nl vt ff cr so si
26189 0000016 dle dc1 dc2 dc3 dc4 nak syn etb can em sub esc fs gs rs us
26190 0000032 sp ! " # $ % & ' ( ) * + , - . /
26191 0000048 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
26192 0000064 @ A B C D E F G H I J K L M N O
26193 0000080 P Q R S T U V W X Y Z [ \ ] ^ _
26194 0000096 ` a b c d e f g h i j k l m n o
26195 0000112 p q r s t u v w x y z { | } ~ del
26196 0000128
```

26197 Note that this volume of IEEE Std. 1003.1-200x allows `nl` or `lf` to be used as the name for the
 26198 ISO/IEC 646:1991 standard IRV character with decimal value 10. The IRV names this character
 26199 `lf` (line feed), but traditional implementations have referred to this character as newline (`nl`) and

26200 the POSIX locale character set symbolic name for the corresponding character is a <newline>
26201 character.

26202 The command:

```
26203 od -A o -t o2x2x -n 18
```

26204 on a system with 32-bit words and an implementation using an input block size of 16 bytes
26205 could write 18 bytes in approximately the following format:

```
26206 0000000 032056 031440 041123 042040 052516 044530 020043 031464
26207          342e 3320 4253 4420 554e 4958 2023 3334
26208          342e3320 42534420 554e4958 20233334
```

```
26209 0000020 032472
```

```
26210          353a
```

```
26211          353a0000
```

```
26212 0000022
```

26213 The command:

```
26214 od -A d -t f -t o4 -t x4 -n 24 -j 0x15
```

26215 on a system with 64-bit doubles (for example, IEEE Std. 754-1985 double precision floating-point
26216 format) would skip 21 bytes of input data and then write 24 bytes in approximately the
26217 following format:

```
26218 0000000 1.0000000000000000e+00 1.5735000000000000e+01
26219 07774000000 00000000000 10013674121 35341217270
26220 3ff00000 00000000 402f3851 eb851eb8
```

```
26221 0000016 1.4066823000000000e+02
```

```
26222 10030312542 04370303230
```

```
26223 40619562 23e18698
```

```
26224 0000024
```

26225 RATIONALE

26226 The *od* utility went through several names in early proposals, including *hd*, *xd*, and most recently
26227 *hexdump*. There were several objections to all of these based on the following reasons:

- 26228 • The *hd* and *xd* names conflicted with historical utilities that behaved differently.
- 26229 • The *hexdump* description was much more complex than needed for a simple dump utility.
- 26230 • The *od* utility has been available on all historical implementations and there was no need to
26231 create a new name for a utility so similar to the historical *od* utility.

26232 The original reasons for not standardizing historical *od* were also fairly widespread. Those
26233 reasons are given below along with rationale explaining why the standard developers believe
26234 that this version does not suffer from the indicated problem:

- 26235 • The BSD and System V versions of *od* have diverged, and the intersection of features
26236 provided by both does not meet the needs of the user community. In fact, the System V
26237 version only provides a mechanism for dumping octal bytes and **shorts**, signed and unsigned
26238 decimal **shorts**, hexadecimal **shorts**, and ASCII characters. BSD added the ability to dump
26239 **floats**, **doubles**, named ASCII characters, and octal, signed decimal, unsigned decimal, and
26240 hexadecimal **longs**. The version presented here provides more normalized forms for
26241 dumping bytes, **shorts**, **ints**, and **longs** in octal, signed decimal, unsigned decimal, and
26242 hexadecimal; **float**, **double**, and **long double**; and named ASCII as well as current locale
26243 characters.

- 26244
26245
26246
26247
- It would not be possible to come up with a compatible superset of the BSD and System V flags that met the requirements of the standard developers. The historical default *od* output is the specified default output of this utility. None of the option letters chosen for this version of *od* conflict with any of the options to historical versions of *od*.
- 26248
26249
26250
26251
26252
26253
26254
26255
26256
26257
26258
26259
26260
26261
26262
- On systems with different sizes for **short**, **int**, and **long**, there was no way to ask for dumps of **ints**, even in the BSD version. Because of the way options are named, the name space could not be extended to solve these problems. This is why the **-t** option was added (with type specifiers more closely matched to the *printf()* formats used in the rest of this volume of IEEE Std. 1003.1-200x) and the optional field sizes were added to the **d**, **f**, **o**, **u**, and **x** type specifiers. It is also one of the reasons why the historical practice was not mandated as a required obsolescent form of *od*. (Although the old versions of *od* are not listed as an obsolescent form, implementations are urged to continue to recognize the older forms for several more years.) The **a**, **c**, **f**, **o**, and **x** types match the meaning of the corresponding format characters in the historical implementations of *od* except for the default sizes of the fields converted. The **d** format is signed in this volume of IEEE Std. 1003.1-200x to match the *printf()* notation. (Historical versions of *od* used **d** as a synonym for **u** in this version. The System V implementation uses **s** for signed decimal; BSD uses **i** for signed decimal and **s** for null-terminated strings.) Other than **d** and **u**, all of the type specifiers match format characters in the historical BSD version of *od*.
- 26263
26264
26265
26266
26267
26268
26269
26270
26271
26272
26273
26274
26275
26276
26277
26278
26279
- The sizes of the C-language types **char**, **short**, **int**, **long**, **float**, **double**, and **long double** are used even though it is recognized that there may be zero or more than one compiler for the C language on an implementation and that they may use different sizes for some of these types. (For example, one compiler might use 2 bytes **shorts**, 2 bytes **ints**, and 4 bytes **longs**, while another compiler (or an option to the same compiler) uses 2 bytes **shorts**, 4 bytes **ints**, and 4 bytes **longs**.) Nonetheless, there has to be a basic size known by the implementation for these types, corresponding to the values reported by invocations of the *getconf* utility when called with *system_var* operands {UCHAR_MAX}, {USHORT_MAX}, {UINT_MAX}, and {ULONG_MAX} for the types **char**, **short**, **int**, and **long**, respectively. There are similar constants required by the ISO C standard, but not required by the System Interfaces volume of IEEE Std. 1003.1-200x or this volume of IEEE Std. 1003.1-200x. They are {FLT_MANT_DIG}, {DBL_MANT_DIG}, and {LDBL_MANT_DIG} for the types **float**, **double**, and **long double**, respectively. If the optional *c99* utility is provided by the implementation and used as specified by this volume of IEEE Std. 1003.1-200x, these are the sizes that would be provided. If an option is used that specifies different sizes for these types, there is no guarantee that the *od* utility is able to interpret binary data output by such a program correctly.
- 26280
26281
26282
- This volume of IEEE Std. 1003.1-200x requires that the numeric values of these lengths be recognized by the *od* utility and that symbolic forms also be recognized. Thus, a portable application can always look at an array of **unsigned long** data elements using *od -t uL*.
- 26283
26284
26285
- The method of specifying the format for the address field based on specifying a starting offset in a file unnecessarily tied the two together. The **-A** option now specifies the address base and the **-S** option specifies a starting offset.
- 26286
26287
26288
26289
26290
26291
26292
26293
- It would be difficult to break the dependence on U.S. ASCII to achieve an internationalized utility. It does not seem to be any harder for *od* to dump characters in the current locale than it is for the *ed* or *sed* **l** commands. The **c** type specifier does this without difficulty and is completely compatible with the historical implementations of the **c** format character when the current locale uses a superset of the ISO/IEC 646:1991 standard as a codeset. The **a** type specifier (from the BSD **a** format character) was left as a portable means to dump ASCII (or more correctly ISO/IEC 646:1991 standard (IRV)) so that headers produced by *pax* could be deciphered even on systems that do not use the ISO/IEC 646:1991 standard as a subset of

- 26294 their base codeset.
- 26295 The use of "***" as an indication of continuation of a multi-byte character in `c` specifier output
26296 was chosen based on seeing an implementation that uses this method. The continuation bytes
26297 have to be marked in a way that is not ambiguous with another single-byte or multi-byte
26298 character.
- 26299 An early proposal used `-S` and `-n`, respectively, for the `-j` and `-N` options eventually selected.
26300 These were changed to avoid conflicts with historical implementations.
- 26301 The original standard specified `-t o2` as the default when no output type was given. This was
26302 changed to `-t oS` (the length of a **short**) to accommodate a supercomputer implementation that
26303 historically used 64 bits as its default (and that defined shorts as 64 bits). This change should not
26304 affect portable applications. The requirement to support lengths of 1, 2, and 4 was added at the
26305 same time to address an historical implementation that had no two-byte data types in its C
26306 compiler.
- 26307 The use of a basic integer data type is intended to allow the implementation to choose a word
26308 size commonly used by applications on that architecture.
- 26309 **FUTURE DIRECTIONS**
- 26310 All option and operand interfaces marked as extensions may be withdrawn in a future issue.
- 26311 **SEE ALSO**
- 26312 *c99, sed*
- 26313 **CHANGE HISTORY**
- 26314 First released in Issue 2.
- 26315 **Issue 4**
- 26316 Aligned with the ISO/IEC 9945-2:1993 standard.
- 26317 **Issue 4, Version 2**
- 26318 The description of the `-c` option is made dependent on the current setting of the `LC_CTYPE`
26319 category, and a reference to the POSIX locale is deleted.
- 26320 **Issue 5**
- 26321 In the description of the `-c` option, the phrase "This is equivalent to `-t c`." is deleted.
26322 The FUTURE DIRECTIONS section has been modified.
- 26323 **Issue 6**
- 26324 The `od` utility is changed to remove the assumption that **short** was a two-byte entity, as per the
26325 revisions in the IEEE P1003.2b draft standard.
- 26326 The normative text is reworded to avoid use of the term "must" for application requirements.

26327 NAME

26328 paste — merge corresponding or subsequent lines of files

26329 SYNOPSIS

26330 paste [-s][-d *list*] *file...*

26331 DESCRIPTION

26332 The *paste* utility shall concatenate the corresponding lines of the given input files, and writes the
26333 resulting lines to standard output.

26334 The default operation of *paste* shall concatenate the corresponding lines of the input files. The
26335 <newline> character of every line except the line from the last input file shall be replaced with a
26336 <tab> character.

26337 If an end-of-file condition is detected on one or more input files, but not all input files, *paste* shall
26338 behave as though empty lines were read from the files on which end-of-file was detected, unless
26339 the *-s* option is specified.

26340 OPTIONS

26341 The *paste* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
26342 12.2, Utility Syntax Guidelines.

26343 The following options shall be supported:

26344 *-d list* Unless a backslash character appears in *list*, each character in *list* is an element
26345 specifying a delimiter character. If a backslash character appears in *list*, the
26346 backslash character and one or more characters following it are an element
26347 specifying a delimiter character as described below. These elements specify one or
26348 more delimiters to use, instead of the default <tab> character, to replace the
26349 <newline> character of the input lines. The elements in *list* shall be used circularly;
26350 that is, when the list is exhausted the first element from the list is reused. When the
26351 *-s* option is specified:

- 26352 • The last <newline> character in a file shall not be modified.
- 26353 • The delimiter shall be reset to the first element of list after each *file* operand is
26354 processed.

26355 When the *-s* option is not specified:

- 26356 • The <newline> characters in the file specified by the last *file* operand shall not
26357 be modified.
- 26358 • The delimiter shall be reset to the first element of list each time a line is
26359 processed from each file.

26360 If a backslash character appears in *list*, it and the character following it shall be
26361 used to represent the following delimiter characters:

26362 \n <newline> character.

26363 \t <tab> character.

26364 \\ Backslash character.

26365 \0 Empty string (not a null character). If '\0' is immediately followed by the
26366 character 'x', the character 'X', or any character defined by the *LC_CTYPE*
26367 **digit** keyword (see the Base Definitions volume of IEEE Std. 1003.1-200x,
26368 Chapter 7, Locale), the results are unspecified.

- 26369 If any other characters follow the backslash, the results are unspecified.
- 26370 **-s** Concatenate all of the lines of each separate input file in command line order. The
26371 <newline> character of every line except the last line in each input file shall be
26372 replaced with the <tab> character, unless otherwise specified by the **-d** option.
- 26373 **OPERANDS**
- 26374 The following operand shall be supported:
- 26375 **file** A path name of an input file. If **'-'** is specified for one or more of the *files*, the
26376 standard input shall be used; the standard input shall be read one line at a time,
26377 circularly, for each instance of **'-'**. Implementations shall support pasting of at
26378 least 12 *file* operands.
- 26379 **STDIN**
- 26380 The standard input shall be used only if one or more *file* operands is **'-'**. See the INPUT FILES
26381 section.
- 26382 **INPUT FILES**
- 26383 The input files shall be text files, except that line lengths shall be unlimited.
- 26384 **ENVIRONMENT VARIABLES**
- 26385 The following environment variables shall affect the execution of *paste*:
- 26386 **LANG** Provide a default value for the internationalization variables that are unset or null.
26387 If *LANG* is unset or null, the corresponding value from the implementation-
26388 defined default locale shall be used. If any of the internationalization variables
26389 contains an invalid setting, the utility shall behave as if none of the variables had
26390 been defined.
- 26391 **LC_ALL** If set to a non-empty string value, override the values of all the other
26392 internationalization variables.
- 26393 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
26394 characters (for example, single-byte as opposed to multi-byte characters in
26395 arguments and input files).
- 26396 **LC_MESSAGES**
- 26397 Determine the locale that should be used to affect the format and contents of
26398 diagnostic messages written to standard error.
- 26399 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 26400 **ASYNCHRONOUS EVENTS**
- 26401 Default.
- 26402 **STDOUT**
- 26403 Concatenated lines of input files shall be separated by the <tab> character (or other characters
26404 under the control of the **-d** option) and terminated by a <newline> character.
- 26405 **STDERR**
- 26406 Used only for diagnostic messages.
- 26407 **OUTPUT FILES**
- 26408 None.
- 26409 **EXTENDED DESCRIPTION**
- 26410 None.

26411 **EXIT STATUS**

26412 The following exit values shall be returned:

26413 0 Successful completion.

26414 >0 An error occurred.

26415 **CONSEQUENCES OF ERRORS**

26416 If one or more input files cannot be opened when the `-s` option is not specified, a diagnostic message shall be written to standard error, but no output is written to standard output. If the `-s` option is specified, the `paste` utility shall provide the default behavior described in Section 1.11 (on page 2224).

26420 **APPLICATION USAGE**

26421 When the escape sequences of the `list` option-argument are used in a shell script, they must be quoted; otherwise, the shell treats the `'\'` as a special character.

26423 Portable applications should only use the specific backslash escaped delimiters presented in this volume of IEEE Std. 1003.1-200x. Historical implementations treat `'\x'`, where `'x'` is not in this list, as `'x'`, but future implementations are free to expand this list to recognize other common escapes similar to those accepted by `printf` and other standard utilities.

26427 Most of the standard utilities work on text files. The `cut` utility can be used to turn files with arbitrary line lengths into a set of text files containing the same data. The `paste` utility can be used to create (or recreate) files with arbitrary line lengths. For example, if `file` contains long lines:

```
26430           cut -b 1-500 -n file > file1
```

```
26431           cut -b 501- -n file > file2
```

26432 creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline> character) and **file2** that contains the remainder of the data from `file`. Note that **file2** is not a text file if there are lines in `file` that are longer than 500 + {LINE_MAX} bytes. The original file can be recreated from **file1** and **file2** using the command:

```
26436           paste -d "\0" file1 file2 > file
```

26437 The commands:

```
26438           paste -d "\0" ...
```

```
26439           paste -d " " ...
```

26440 are not necessarily equivalent; the latter is not specified by this volume of IEEE Std. 1003.1-200x and may result in an error. The construct `'\0'` is used to mean “no separator” because historical versions of `paste` did not follow the syntax guidelines, and the command:

```
26443           paste -d" " ...
```

26444 could not be handled properly by `getopt()`.

26445 **EXAMPLES**

26446 1. Write out a directory in four columns:

```
26447           ls | paste - - - -
```

26448 2. Combine pairs of lines from a file into single lines:

```
26449           paste -s -d "\t\n" file
```

26450 **RATIONALE**

26451 None.

26452 **FUTURE DIRECTIONS**

26453 None.

26454 **SEE ALSO**26455 *cut, grep, pr*26456 **CHANGE HISTORY**

26457 First released in Issue 2.

26458 **Issue 4**

26459 Aligned with the ISO/IEC 9945-2:1993 standard.

26460 **Issue 6**

26461 The normative text is reworded to avoid use of the term “must” for application requirements.

26462 NAME

26463 patch — apply changes to files

26464 SYNOPSIS

```
26465 UP patch [-blNR][ -c | -e | -n][-d dir][-D define][-i patchfile]
26466 [-o outfile][-p num][-r rejectfile][file]
```

26467

26468 DESCRIPTION

26469 The *patch* utility shall read a source (patch) file containing any of the three forms of difference (diff) listings produced by the *diff* utility (normal, context or in the style of *ed*) and apply those differences to a file. By default, *patch* shall read from the standard input.

26472 The *patch* utility shall attempt to determine the type of the *diff* listing, unless overruled by a *-c*, *-e*, or *-n* option.

26474 If the patch file contains more than one patch, *patch* shall attempt to apply each of them as if they came from separate patch files. (In this case, the application shall ensure that the name of the patch file is determinable for each *diff* listing.)

26477 OPTIONS

26478 The *patch* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

26480 The following options shall be supported:

26481 **-b** Save a copy of the original contents of each modified file, before the differences are applied, in a file of the same name with the suffix **.orig** appended to it. If the file already exists, it shall be overwritten; if multiple patches are applied to the same file, the **.orig** file shall be written only for the first patch. When the *-o outfile* option is also specified, *file.orig* shall not be created but, if *outfile* already exists, *outfile.orig* shall be created.

26487 **-c** Interpret the patch file as a context difference (the output of the utility *diff* when the *-c* or *-C* options are specified).

26489 **-d dir** Change the current directory to *dir* before processing as described in the EXTENDED DESCRIPTION section.

26491 **-D define** Mark changes with one of the following C preprocessor constructs:

```
26492 #ifdef define
26493 ...
26494 #endif
26495 #ifndef define
26496 ...
26497 #endif
```

26498 optionally combined with the C preprocessor construct **#else**.

26499 **-e** Interpret the patch file as an *ed* script, rather than a *diff* script.

26500 **-i patchfile** Read the patch information from the file named by the path name *patchfile*, rather than the standard input.

26502 **-l** (The letter ell.) Cause any sequence of <blank> characters in the difference script to match any sequence of <blank> characters in the input file. Other characters shall be matched exactly.

26505 **-n** Interpret the script as a normal difference.

26506 **-N** Ignore patches where the differences have already been applied to the file; by
26507 default, already-applied patches shall be rejected.

26508 **-o outfile** Instead of modifying the files (specified by the *file* operand or the difference
26509 listings) directly, write a copy of the file referenced by each patch, with the
26510 appropriate differences applied, to *outfile*. Multiple patches for a single file shall
26511 be applied to the intermediate versions of the file created by any previous patches,
26512 and shall result in multiple, concatenated versions of the file being written to
26513 *outfile*.

26514 **-p num** For all path names in the patch file that indicate the names of files to be patched,
26515 delete *num* path name components from the beginning of each path name. If the
26516 path name in the patch file is absolute, any leading slashes shall be considered the
26517 first component (that is, **-p 1** shall remove the leading slashes). Specifying **-p 0**
26518 shall cause the full path name to be used. If **-p** is not specified, only the basename
26519 (the final path name component) shall be used.

26520 **-R** Reverse the sense of the patch script; that is, assume that the difference script was
26521 created from the new version to the old version. The **-R** option cannot be used
26522 with *ed* scripts. The *patch* utility shall attempt to reverse each portion of the script
26523 before applying it. Rejected differences shall be saved in swapped format. If this
26524 option is not specified, and until a portion of the patch file is successfully applied,
26525 *patch* attempts to apply each portion in its reversed sense as well as in its normal
26526 sense. If the attempt is successful, the user shall be prompted to determine if the
26527 **-R** option should be set.

26528 **-r rejectfile** Override the default reject file name. In the default case, the reject file shall have
26529 the same name as the output file, with the suffix **.rej** appended to it; see **Patch**
26530 **Application** (on page 2903).

26531 **OPERANDS**

26532 The following operand shall be supported:

26533 *file* A path name of a file to patch.

26534 **STDIN**

26535 See the INPUT FILES section.

26536 **INPUT FILES**

26537 Input files shall be text files.

26538 **ENVIRONMENT VARIABLES**

26539 The following environment variables shall affect the execution of *patch*:

26540 **LANG** Provide a default value for the internationalization variables that are unset or null.
26541 If **LANG** is unset or null, the corresponding value from the implementation-
26542 defined default locale shall be used. If any of the internationalization variables
26543 contains an invalid setting, the utility shall behave as if none of the variables had
26544 been defined.

26545 **LC_ALL** If set to a non-empty string value, override the values of all the other
26546 internationalization variables.

26547 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
26548 characters (for example, single-byte as opposed to multi-byte characters in
26549 arguments and input files).

- 26550 **LC_MESSAGES**
- 26551 Determine the locale that should be used to affect the format and contents of
- 26552 diagnostic messages written to standard error and informative messages written to
- 26553 standard output.
- 26554 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 26555 **LC_TIME** Determine the locale for recognizing the format of file timestamps written by the
- 26556 *diff* utility in a context-difference input file.
- 26557 **ASYNCHRONOUS EVENTS**
- 26558 Default.
- 26559 **STDOUT**
- 26560 Not used.
- 26561 **STDERR**
- 26562 Used for diagnostic and informational messages.
- 26563 **OUTPUT FILES**
- 26564 The output of the *patch* utility, the save files (**.orig** suffixes) and the reject files (**.rej** suffixes) shall
- 26565 be text files.
- 26566 **EXTENDED DESCRIPTION**
- 26567 A patchfile may contain patching instructions for more than one file; file names shall be
- 26568 determined as specified in **File Name Determination** (on page 2903). When the **-b** option is
- 26569 specified, for each patched file, the original shall be saved in a file of the same name with the
- 26570 suffix **.orig** appended to it.
- 26571 For each patched file, a reject file may also be created as noted in **Patch Application** (on page
- 26572 2903). In the absence of a **-r** option, the name of this file shall be formed by appending the suffix
- 26573 **.rej** to the original file name.
- 26574 **Patchfile Format**
- 26575 The patch file shall contain zero or more lines of header information followed by one or more
- 26576 patches. Each patch shall contain zero or more lines of file name identification in the format
- 26577 produced by *diff -c*, and one or more sets of *diff* output, which are customarily called *hunks*.
- 26578 The *patch* utility shall recognize the following expression in the header information:
- 26579 **Index:** *pathname*
- 26580 The file to be patched is named *pathname*.
- 26581 If all lines (including headers) within a patch begin with the same leading sequence of <blank>
- 26582 characters, the *patch* utility shall remove this sequence before proceeding. Within each patch, if
- 26583 the type of difference is context, the *patch* utility shall recognize the following expressions:
- 26584 ******* *filename timestamp*
- 26585 The patches arose from *filename*.
- 26586 **---** *filename timestamp*
- 26587 The patches should be applied to *filename*.
- 26588 Each hunk within a patch shall be the *diff* output to change a line range within the original file.
- 26589 The line numbers for successive hunks within a patch shall occur in ascending order.

26590 **File Name Determination**

26591 If no *file* operand is specified, *patch* shall perform the following steps to determine the file name
26592 to use:

- 26593 1. If the type of *diff* is context, the *patch* utility shall delete path name components (as
26594 specified by the **-p** option) from the file name on the line beginning with "****", then test
26595 for the existence of this file relative to the current directory (or the directory specified with
26596 the **-d** option). If the file exists, the *patch* utility shall use this file name.
- 26597 2. If the type of *diff* is context, the *patch* utility shall delete the path name components (as
26598 specified by the **-p** option) from the file name on the line beginning with "——", then test
26599 for the existence of this file relative to the current directory (or the directory specified with
26600 the **-d** option). If the file exists, the *patch* utility shall use this file name.
- 26601 3. If the header information contains a line beginning with the string **Index:**, the *patch* utility
26602 shall delete path name components (as specified by the **-p** option) from this line, then test
26603 for the existence of this file relative to the current directory (or the directory specified with
26604 the **-d** option). If the file exists, the *patch* utility shall use this file name.
- 26605 XSI 4. If an **SCCS** directory exists in the current directory, *patch* shall attempt to perform a *get -e*
26606 **SCCS/s.filename** command to retrieve an editable version of the file. If the file exists, the
26607 *patch* utility shall use this file name.
- 26608 5. The *patch* utility shall write a prompt to standard output and request a file name
26609 interactively from the controlling terminal (for example, **/dev/tty**).

26610 **Patch Application**

26611 If the **-c**, **-e**, or **-n** option is present, the *patch* utility shall interpret information within each hunk
26612 as a context difference, an *ed* difference or a normal difference, respectively. In the absence of
26613 any of these options, the *patch* utility shall determine the type of difference based on the format
26614 of information within the hunk.

26615 For each hunk, the *patch* utility shall begin to search for the place to apply the patch at the line
26616 number at the beginning of the hunk, plus or minus any offset used in applying the previous
26617 hunk. If lines matching the hunk context are not found, *patch* shall scan both forwards and
26618 backwards at least 1 000 bytes for a set of lines that match the hunk context.

26619 If no such place is found and it is a context difference, then another scan shall take place,
26620 ignoring the first and last line of context. If that fails, the first two and last two lines of context
26621 shall be ignored and another scan shall be made. Implementations may search more extensively
26622 for installation locations.

26623 If no location can be found, the *patch* utility shall append the hunk to the reject file. The rejected
26624 hunk shall be written in context-difference format regardless of the format of the patch file. If the
26625 input was a normal or *ed-style* difference, the reject file may contain differences with zero lines
26626 of context. The line numbers on the hunks in the reject file may be different from the line
26627 numbers in the patch file since they shall reflect the approximate locations for the failed hunks in
26628 the new file rather than the old one.

26629 If the type of patch is an *ed* diff, the implementation may accomplish the patching by invoking
26630 the *ed* utility.

26631 **EXIT STATUS**

26632 The following exit values shall be returned:

- 26633 0 Successful completion.

26634 1 One or more lines were written to a reject file.

26635 >1 An error occurred.

26636 CONSEQUENCES OF ERRORS

26637 Patches that cannot be correctly placed in the file shall be written to a reject file.

26638 APPLICATION USAGE

26639 The **-R** option does not work with *ed* scripts because there is too little information to reconstruct
26640 the reverse operation.

26641 The **-p** option makes it possible to customize a patchfile to local user directory structures
26642 without manually editing the patchfile. For example, if the file name in the patch file was:

26643 /curds/whey/src/blurfl/blurfl.c

26644 Setting **-p 0** gives the entire path name unmodified; **-p 1** gives:

26645 curds/whey/src/blurfl/blurfl.c

26646 without the leading slash, **-p 4** gives:

26647 blurfl/blurfl.c

26648 and not specifying **-p** at all gives:

26649 blurfl.c .

26650 EXAMPLES

26651 None.

26652 RATIONALE

26653 Some of the functionality in historical *patch* implementations was not specified. The following
26654 documents those features present in historical implementations that have not been specified.

26655 A deleted piece of functionality was the '+' pseudo-option allowing an additional set of options
26656 and a patch file operand to be given. This was seen as being insufficiently useful to standardize.

26657 In historical implementations, if the string "Prereq:" appeared in the header, the *patch* utility
26658 would search for the corresponding version information (the string specified in the header,
26659 delimited by <blank>s or the beginning or end of a line or the file) anywhere in the original file.
26660 This was deleted as too simplistic and insufficiently trustworthy a mechanism to standardize.
26661 For example, if:

26662 Prereq: 1.2

26663 were in the header, the presence of a delimited 1.2 anywhere in the file would satisfy the
26664 prerequisite.

26665 The following options were dropped from historical implementations of *patch* as insufficiently
26666 useful to standardize:

26667 **-b** The **-b** option historically provided a method for changing the name extension of
26668 the backup file from the default **.orig**. This option has been modified and retained
26669 in this volume of IEEE Std. 1003.1-200x.

26670 **-F** The **-F** option specified the number of lines of a context diff to ignore when
26671 searching for a place to install a patch.

26672 **-f** The **-f** option historically caused *patch* not to request additional information from
26673 the user.

- 26674 **-r** The **-r** option historically provided a method of overriding the extension of the
26675 reject file from the default **.rej**.
- 26676 **-s** The **-s** option historically caused *patch* to work silently unless an error occurred.
- 26677 **-x** The **-x** option historically set internal debugging flags.
- 26678 In some file system implementations, the saving of a **.orig** file may produce unwanted results. In
26679 the case of 12, 13, or 14-character file names (on file systems supporting 14-character maximum
26680 file names), the **.orig** file overwrites the new file. The reject file may also exceed this file name
26681 limit. It was suggested, due to some historical practice, that a tilde ('~') suffix be used instead
26682 of **.orig** and some other character instead of the **.rej** suffix. This was rejected because it is not
26683 obvious to the user which file is which. The suffixes **.orig** and **.rej** are clearer and more
26684 understandable.
- 26685 The **-b** option has the opposite sense in some historical implementations—do not save the **.orig**
26686 file. The default case here is not to save the files, making *patch* behave more consistently with the
26687 other standard utilities.
- 26688 The **-w** option in early proposals was changed to **-I** to match historical practice.
- 26689 The **-N** option was included because without it, a non-interactive application cannot reject
26690 previously applied patches. For example, if a user is piping the output of *diff* into the *patch*
26691 utility, and the user only wants to patch a file to a newer version non-interactively, the **-N**
26692 option is required.
- 26693 Changes to the **-I** option description were proposed to allow matching across <newline>s in
26694 addition to just <blank>s. Since this is not historical practice, and since some ambiguities could
26695 result, it is suggested that future developments in this area utilize another option letter, such as
26696 **-L**.
- 26697 **FUTURE DIRECTIONS**
- 26698 None.
- 26699 **SEE ALSO**
- 26700 *ed*, *diff*
- 26701 **CHANGE HISTORY**
- 26702 First released in Issue 4.
- 26703 **Issue 5**
- 26704 **FUTURE DIRECTIONS** section added.
- 26705 **Issue 6**
- 26706 This utility is now marked as part of the User Portability Utilities option.
- 26707 The description of the **-D** option and the steps in **File Name Determination** (on page 2903) are
26708 changed to match historical practice as defined in the IEEE P1003.2b draft standard.
- 26709 The normative text is reworded to avoid use of the term “must” for application requirements.

26710 NAME

26711 pathchk — check path names

26712 SYNOPSIS

26713 pathchk [-p] *pathname...*

26714 DESCRIPTION

26715 The *pathchk* utility shall check that one or more path names are valid (that is, they could be used
 26716 to access or create a file without causing syntax errors) and portable (that is, no file name
 26717 truncation results). More extensive portability checks are provided by the **-p** option.

26718 By default, the *pathchk* utility shall check each component of each *pathname* operand based on the
 26719 underlying file system. A diagnostic shall be written for each *pathname* operand that:

- 26720 • Is longer than {PATH_MAX} bytes (see **Path Name Variable Values** in the Base Definitions
 26721 volume of IEEE Std. 1003.1-200x, Chapter 13, Headers, <limits.h>)
- 26722 • Contains any component longer than {NAME_MAX} bytes in its containing directory
- 26723 • Contains any component in a directory that is not searchable
- 26724 • Contains any character in any component that is not valid in its containing directory

26725 The format of the diagnostic message is not specified, but shall indicate the error detected and
 26726 the corresponding *pathname* operand.

26727 It shall not be considered an error if one or more components of a *pathname* operand do not exist
 26728 as long as a file matching the path name specified by the missing components could be created
 26729 that does not violate any of the checks specified above.

26730 OPTIONS

26731 The *pathchk* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 26732 12.2, Utility Syntax Guidelines.

26733 The following option shall be supported:

- 26734 **-p** Instead of performing checks based on the underlying file system, write a
 26735 diagnostic for each *pathname* operand that:
 - 26736 • Is longer than {_POSIX_PATH_MAX} bytes (see **Minimum Values** in the Base
 26737 Definitions volume of IEEE Std. 1003.1-200x, Chapter 13, Headers, <limits.h>)
 - 26738 • Contains any component longer than {_POSIX_NAME_MAX} bytes
 - 26739 • Contains any character in any component that is not in the portable file name
 26740 character set

26741 OPERANDS

26742 The following operand shall be supported:

26743 *pathname* A path name to be checked.

26744 STDIN

26745 Not used.

26746 INPUT FILES

26747 None.

26748 ENVIRONMENT VARIABLES

26749 The following environment variables shall affect the execution of *pathchk*:

26750 *LANG* Provide a default value for the internationalization variables that are unset or null.
 26751 If *LANG* is unset or null, the corresponding value from the implementation-

26752 defined default locale shall be used. If any of the internationalization variables
 26753 contains an invalid setting, the utility shall behave as if none of the variables had
 26754 been defined.

26755 **LC_ALL** If set to a non-empty string value, override the values of all the other
 26756 internationalization variables.

26757 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 26758 characters (for example, single-byte as opposed to multi-byte characters in
 26759 arguments).

26760 **LC_MESSAGES**
 26761 Determine the locale that should be used to affect the format and contents of
 26762 diagnostic messages written to standard error.

26763 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

26764 **ASYNCHRONOUS EVENTS**
 26765 Default.

26766 **STDOUT**
 26767 Not used.

26768 **STDERR**
 26769 Used only for diagnostic messages.

26770 **OUTPUT FILES**
 26771 None.

26772 **EXTENDED DESCRIPTION**
 26773 None.

26774 **EXIT STATUS**
 26775 The following exit values shall be returned:
 26776 0 All *pathname* operands passed all of the checks.
 26777 >0 An error occurred.

26778 **CONSEQUENCES OF ERRORS**
 26779 Default.

26780 **APPLICATION USAGE**
 26781 The *test* utility can be used to determine whether a given path name names an existing file; it
 26782 does not, however, give any indication of whether or not any component of the path name was
 26783 truncated in a directory where the **_POSIX_NO_TRUNC** feature is not in effect. The *pathchk*
 26784 utility does not check for file existence; it performs checks to determine if a path name does exist
 26785 or could be created with no path name component truncation.

26786 The *noclobber* option in the shell (see the *set* (on page 2297) special built-in) can be used to
 26787 atomically create a file. As with all file creation semantics in the System Interfaces volume of
 26788 IEEE Std. 1003.1-200x, it guarantees atomic creation, but still depends on applications to agree
 26789 on conventions and cooperate on the use of files after they have been created.

26790 **EXAMPLES**
 26791 To verify that all path names in an imported data interchange archive are legitimate and
 26792 unambiguous on the current system:
 26793

```
pax -f archive | sed -e '/ == ./s///' | xargs pathchk
```


 26794

```
if [ $? -eq 0 ]
```


 26795

```
then
```

```

26796         pax -r -f archive
26797     else
26798         echo Investigate problems before importing files.
26799         exit 1
26800     fi

26801     To verify that all files in the current directory hierarchy could be moved to any system
26802     conforming to the System Interfaces volume of IEEE Std. 1003.1-200x that also supports the pax
26803     utility:

26804     find . -print | xargs pathchk -p
26805     if [ $? -eq 0 ]
26806     then
26807         pax -w -f archive .
26808     else
26809         echo Portable archive cannot be created.
26810         exit 1
26811     fi

26812     To verify that a user-supplied path name names a readable file and that the application can
26813     create a file extending the given path without truncation and without overwriting any existing
26814     file:

26815     case $- in
26816         *C*)     reset="";;
26817         *)       reset="set +C"
26818                 set -C;;
26819     esac
26820     test -r "$path" && pathchk "$path.out" &&
26821         rm "$path.out" > "$path.out"
26822     if [ $? -ne 0 ]; then
26823         printf "%s: %s not found or %s.out fails \
26824     creation checks.\n" $0 "$path" "$path"
26825         $reset      # Reset the noclobber option in case a trap
26826                   # on EXIT depends on it.
26827         exit 1
26828     fi
26829     $reset
26830     PROCESSING < "$path" > "$path.out"

26831     The following assumptions are made in this example:

26832     1. PROCESSING represents the code that is used by the application to use $path once it is
26833     verified that $path.out works as intended.

26834     2. The state of the noclobber option is unknown when this code is invoked and should be set
26835     on exit to the state it was in when this code was invoked. (The reset variable is used in this
26836     example to restore the initial state.)

26837     3. Note the usage of:

26838     rm "$path.out" > "$path.out"

26839     a. The pathchk command has already verified, at this point, that $path.out is not
26840     truncated.

26841     b. With the noclobber option set, the shell verifies that $path.out does not already exist
26842     before invoking rm.

```


26843 c. If the shell succeeded in creating **\$path.out**, *rm* removes it so that the application can
 26844 create the file again in the **PROCESSING** step.

26845 d. If the **PROCESSING** step wants the file to exist already when it is invoked, the:

26846 `rm "$path.out" > "$path.out"`

26847 should be replaced with:

26848 `> "$path.out"`

26849 which verifies that the file did not already exist, but leaves **\$path.out** in place for use
 26850 by **PROCESSING**.

26851 RATIONALE

26852 The *pathchk* utility is new, commissioned for this volume of IEEE Std. 1003.1-200x. It, along with
 26853 the *set -C(noclobber)* option added to the shell, replaces the *mktemp*, *validnam*, and *create* utilities
 26854 that appeared in early proposals. All of these utilities were attempts to solve several common
 26855 problems:

- 26856 • Verify the validity (for several different definitions of “valid”) of a path name supplied by a
 26857 user, generated by an application, or imported from an external source.

- 26858 • Atomically create a file.

- 26859 • Perform various string handling functions to generate a temporary file name.

26860 The *create* utility, included in an early proposal, provided checking and atomic creation in a
 26861 single invocation of the utility; these are orthogonal issues and need not be grouped into a single
 26862 utility. Note that the *noclobber* option also provides a way of creating a lock for process
 26863 synchronization; since it provides an atomic *create*, there is no race between a test for existence
 26864 and the following creation if it did not exist.

26865 Having a function like *tmpnam()* in the ISO C standard is important in many high-level
 26866 languages. The shell programming language, however, has built-in string manipulation
 26867 facilities, making it very easy to construct temporary file names. The names needed obviously
 26868 depend on the application, but are frequently of a form similar to:

26869 `$TMPDIR/application_abbreviation$$.suffix`

26870 In cases where there is likely to be contention for a given suffix, a simple shell *for* or *while* loop
 26871 can be used with the shell *noclobber* option to create a file without risk of collisions, as long as
 26872 applications trying to use the same file name name space are cooperating on the use of files after
 26873 they have been created.

26874 FUTURE DIRECTIONS

26875 None.

26876 SEE ALSO

26877 *test*, Section 2.7 (on page 2251)

26878 CHANGE HISTORY

26879 First released in Issue 4.

26880 NAME

26881 pax — portable archive interchange

26882 SYNOPSIS

26883 pax [-cdnv][[-H|-L][[-f *archive*][[-s *replstr*]]...[*pattern*...]26884 pax -r[-cdiknuv][[-H|-L][[-f *archive*][[-o *options*]]...[-p *string*]]...
26885 [-s *replstr*]]...[*pattern*...]26886 pax -w[-dituvX][[-H|-L][[-b *blocksize*][[-a][[-f *archive*][[-o *options*]]...]
26887 [-s *replstr*]]...[-x *format*][*file*...]26888 pax -r -w[-diklntuvX][[-H|-L][[-p *string*]]...[-s *replstr*]]...
26889 [*file*...] *directory*

26890 DESCRIPTION

26891 The *pax* utility shall read, write, and write lists of the members of archive files and copy
26892 directory hierarchies. A variety of archive formats shall be supported; see the *-x format* option.26893 The action to be taken depends on the presence of the *-r* and *-w* options. The four combinations
26894 of *-r* and *-w* are referred to as the four modes of operation: **list**, **read**, **write**, and **copy** modes,
26895 corresponding respectively to the four forms shown in the SYNOPSIS section.26896 **list** In **list** mode (when neither *-r* nor *-w* are specified), *pax* shall write the names of
26897 the members of the archive file read from the standard input, with path names
26898 matching the specified patterns, to standard output. If a named file is of type
26899 directory, the file hierarchy rooted at that file shall be listed as well.26900 **read** In **read** mode (when *-r* is specified, but *-w* is not), *pax* shall extract the members of
26901 the archive file read from the standard input, with path names matching the
26902 specified patterns. If an extracted file is of type directory, the file hierarchy rooted
26903 at that file shall be extracted as well. The extracted files shall be created relative to
26904 the current file hierarchy.26905 If an attempt is made to extract a directory when the directory already exists, this
26906 shall not be considered to be an error. If an attempt is made to extract a FIFO when
26907 the FIFO already exists, this shall not be considered to be an error.26908 The ownership, access, and modification times, and file mode of the restored files
26909 are discussed under the *-p* option.26910 **write** In **write** mode (when *-w* is specified, but *-r* is not), *pax* shall write the contents of
26911 the *file* operands to the standard output in an archive format. If no *file* operands are
26912 specified, a list of files to copy, one per line, shall be read from the standard input.
26913 A file of type directory shall include all of the files in the file hierarchy rooted at the
26914 file.26915 **copy** In **copy** mode (when both *-r* and *-w* are specified), *pax* shall copy the *file* operands
26916 to the destination directory.26917 If no *file* operands are specified, a list of files to copy, one per line, shall be read
26918 from the standard input. A file of type directory shall include all of the files in the
26919 file hierarchy rooted at the file.26920 The effect of the **copy** shall be as if the copied files were written to an archive file
26921 and then subsequently extracted, except that there may be hard links between the
26922 original and the copied files. If the destination directory is a subdirectory of one of
26923 the files to be copied, the results are unspecified. If the destination directory is a
26924 file of a type not defined by the System Interfaces volume of IEEE Std. 1003.1-200x,

26925 the results are implementation-defined; otherwise, it shall be an error for the file
 26926 named by the *directory* operand not to exist, not be writable by the user, or not be a
 26927 file of type directory.

26928 In **read** or **copy** modes, if intermediate directories are necessary to extract an archive member,
 26929 *pax* shall perform actions equivalent to the *mkdir()* function defined in the System Interfaces
 26930 volume of IEEE Std. 1003.1-200x, called with the following arguments:

- 26931 • The intermediate directory used as the *path* argument
- 26932 • The value of the bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO as the *mode*
 26933 argument

26934 If any specified *pattern* or *file* operands are not matched by at least one file or archive member,
 26935 *pax* shall write a diagnostic message to standard error for each one that did not match and exit
 26936 with a non-zero exit status.

26937 The archive formats described in the EXTENDED DESCRIPTION section shall be automatically
 26938 detected on input. The default output archive format shall be implementation-defined.

26939 A single archive can span multiple files. The *pax* utility shall determine, in an implementation-
 26940 defined manner, what file to read or write as the next file.

26941 If the selected archive format supports the specification of linked files, it shall be an error if these
 26942 files cannot be linked when the archive is extracted. For archive formats that do not store file
 26943 contents with each name that causes a hard link, if the file that contains the data is not extracted
 26944 during this *pax* session, either the data shall be restored from the original file, or a diagnostic
 26945 message shall be displayed with the name of a file that can be used to extract the data. In
 26946 traversing directories, *pax* shall detect infinite loops; that is, entering a previously visited
 26947 directory that is an ancestor of the last file visited. When it detects an infinite loop, *pax* shall
 26948 write a diagnostic message to standard error and shall terminate.

26949 OPTIONS

26950 The *pax* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 26951 12.2, Utility Syntax Guidelines, except that the order of presentation of the **-o**, **-p**, and **-s** options
 26952 is significant.

26953 The following options shall be supported:

- 26954 **-r** Read an archive file from standard input.
- 26955 **-w** Write files to the standard output in the specified archive format.
- 26956 **-a** Append files to the end of the archive. It is implementation-defined which devices
 26957 on the system support appending. Additional file formats unspecified by this
 26958 volume of IEEE Std. 1003.1-200x may impose restrictions on appending.
- 26959 **-b *blocksize*** Block the output at a positive decimal integer number of bytes per write to the
 26960 archive file. Devices and archive formats may impose restrictions on blocking.
 26961 Blocking shall be automatically determined on input. Portable applications shall
 26962 not specify a *blocksize* value larger than 32 256. Default blocking when creating
 26963 archives depends on the archive format. (See the **-x** option below.)
- 26964 **-c** Match all file or archive members except those specified by the *pattern* or *file*
 26965 operands.
- 26966 **-d** Cause files of type directory being copied or archived or archive members of type
 26967 directory being extracted or listed to match only the file or archive member itself
 26968 and not the file hierarchy rooted at the file.

26969	-f <i>archive</i>	Specify the path name of the input or output archive, overriding the default standard input (in list or read modes) or standard output (write mode).
26970		
26971	-H	If a symbolic link referencing a file of type directory is specified on the command line, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. The default behavior shall be to archive the symbolic link itself.
26972		
26973		
26974		
26975	-i	Interactively rename files or archive members. For each archive member matching a <i>pattern</i> operand or file matching a <i>file</i> operand, a prompt shall be written to the file /dev/tty . The prompt shall contain the name of the file or archive member, but the format is otherwise unspecified. A line shall then be read from /dev/tty . If this line is blank, the file or archive member shall be skipped. If this line consists of a single period, the file or archive member shall be processed with no modification to its name. Otherwise, its name shall be replaced with the contents of the line. The <i>pax</i> utility shall immediately exit with a non-zero exit status if end-of-file is encountered when reading a response or if /dev/tty cannot be opened for reading and writing.
26976		
26977		
26978		
26979		
26980		
26981		
26982		
26983		
26984		
26985		The results of extracting a hard link to a file that has been renamed during extraction are unspecified.
26986		
26987	-k	Prevent the overwriting of existing files.
26988	-l	(The letter ell.) In copy mode, hard links shall be made between the source and destination file hierarchies whenever possible.
26989		
26990	-L	If a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. The default behavior shall be to archive the symbolic link itself.
26991		
26992		
26993		
26994		
26995	-n	Select the first archive member that matches each <i>pattern</i> operand. No more than one archive member shall be matched for each pattern (although members of type directory shall still match the file hierarchy rooted at that file).
26996		
26997		
26998	-o <i>options</i>	Provide information to the implementation to modify the algorithm for extracting or writing files. The value of <i>options</i> shall consist of one or more comma-separated keywords of the form:
26999		
27000		
27001		<i>keyword</i> [[:]= <i>value</i>][, <i>keyword</i> [[:]= <i>value</i>], ...]
27002		Some keywords apply only to certain file formats, as indicated with each description. Use of keywords that are inapplicable to the file format being processed produces undefined results.
27003		
27004		
27005		Keywords in the <i>options</i> argument shall be a string that would be a valid portable file name as described in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 3.282, Portable File Name Character Set.
27006		
27007		
27008	Note:	Keywords are not expected to be file names, merely to follow the same character composition rules as portable file names.
27009		
27010		Keywords can be preceded with white space. The <i>value</i> field shall consist of zero or more characters; within <i>value</i> , the application shall precede any literal comma with a backslash, which shall be ignored, but preserves the comma as part of <i>value</i> . A comma as the final character, or a comma followed solely by white space as the final characters, in <i>options</i> shall be ignored. Multiple -o options can be specified; if
27011		
27012		
27013		
27014		

27015 keywords given to these multiple `-o` options conflict, the keywords and values
 27016 appearing later in command line sequence shall take precedence and the earlier
 27017 shall be silently ignored. The following keyword values of *options* shall be
 27018 supported for the file formats as indicated:

27019 **delete=pattern**

27020 (Applicable only to the `-x pax` format.) When used in **write** or **copy** mode, *pax*
 27021 shall omit from extended header records that it produces any keywords
 27022 matching the string pattern. When used in **read** or **list** mode, *pax* shall ignore
 27023 any keywords matching the string pattern in the extended header records. In
 27024 both cases, matching shall be performed using the pattern matching notation
 27025 described in Section 2.14.1 (on page 2274) and Section 2.14.2 (on page 2274).
 27026 For example:

27027 `-o delete=security.*`

27028 would suppress security-related information. See **pax Extended Header** (on
 27029 page 2923) for extended header record keyword usage.

27030 **exthdr.name=string**

27031 (Applicable only to the `-x pax` format.) This keyword allows user control over
 27032 the name that is written into the **ustar** header blocks for the extended header
 27033 produced under the circumstances described in **pax Header Block** (on page
 27034 2922). The name shall be the contents of *string*, after the following character
 27035 substitutions have been made:

<i>string</i> Includes:	Replaced By:
%d	The directory name of the file, equivalent to the result of the <i>dirname</i> utility on the translated path name.
%f	The file name of the file, equivalent to the result of the <i>basename</i> utility on the translated path name.
%%	A '%' character.

27043 Any other '%' characters in *string* produce undefined results.

27044 If no `-o exthdr.name=string` is specified, *pax* shall use the following default
 27045 value:

27046 `%d/PaxHeaders/%f`

27047 **globexthdr.name=string**

27048 (Applicable only to the `-x pax` format.) When used in **write** or **copy** mode with
 27049 the appropriate options, *pax* creates global extended header records with **ustar**
 27050 header blocks that will be treated as regular files by previous versions of *pax*.
 27051 This keyword allows user control over the name that is written into the **ustar**
 27052 header blocks for global extended header records. The name shall be the
 27053 contents of *string*, after the following character substitutions have been made:

<i>string</i> Includes:	Replaced By:
%n	An integer that represents the sequence number of the global extended header record in the archive, starting at 1.
%%	A '%' character.

27054
 27055
 27056
 27057
 27058

27059 Any other ‘%’ characters in *string* produce undefined results.

27060 If no **-o globexthdr.name=string** is specified, *pax* shall use the following
27061 default value:

27062 \$TMPDIR/GlobalHead.%n

27063 where \$TMPDIR represents the value of the *TMPDIR* environment variable. If
27064 *TMPDIR* is not set, *pax* shall use **/tmp**.

27065 **invalid=action**
27066 (Applicable only to the **-x pax** format.) This keyword allows user control over
27067 the action *pax* takes upon encountering values in an extended header record
27068 that, in **read** or **copy** mode, are invalid in the destination hierarchy or, in **list**
27069 mode, cannot be written in the codeset and current locale of the
27070 implementation. The following are invalid values that shall be recognized by
27071 *pax*:

- 27072 — In **read** or **copy** mode, a file name or link name that contains character
27073 encodings invalid in the destination hierarchy. (For example, the name
27074 may contain embedded NULs.)
- 27075 — In **read** or **copy** mode, a file name or link name that is longer than the
27076 maximum allowed in the destination hierarchy (for either a path name
27077 component or the entire path name).
- 27078 — In **list** mode, any character string value (file name, link name, user name,
27079 and so on) that cannot be written in the codeset and current locale of the
27080 implementation.

27081 The following mutually-exclusive values of the *action* argument are
27082 supported:

27083 **bypass** In **read** or **copy** mode, *pax* shall bypass the file, causing no
27084 change to the destination hierarchy. In **list** mode, *pax* shall write
27085 all requested valid values for the file, but its method for writing
27086 invalid values is unspecified.

27087 **rename** In **read** or **copy** mode, *pax* shall act as if the **-i** option were in
27088 effect for each file with invalid file name or link name values,
27089 allowing the user to provide a replacement name interactively.
27090 In **list** mode, *pax* shall behave identically to the **bypass** action.

27091 **UTF-8** When used in **read**, **copy**, or **list** mode and a file name, link
27092 name, owner name, or any other field in an extended header
27093 record cannot be translated from the *pax* UTF-8 codeset format
27094 to the codeset and current locale of the implementation, *pax*
27095 shall use the actual UTF-8 encoding for the name.

27096 **write** In **read** or **copy** mode, *pax* shall write the file, translating or
27097 truncating the name, regardless of whether this may overwrite
27098 an existing file with a valid name. In **list** mode, *pax* shall behave
27099 identically to the **bypass** action.

27100 If no **-o invalid=** option is specified, *pax* shall act as if **-o invalid=bypass** were
27101 specified. Any overwriting of existing files that may be allowed by the
27102 **-o invalid=** actions shall be subject to permission (**-p**) and modification time
27103 (**-u**) restrictions, and shall be suppressed if the **-k** option is also specified.

27104 **linkdata** (Applicable only to the **-x pax** format.) In **write** mode, *pax* shall write the
 27105 contents of a file to the archive even when that file is merely a hard link to a
 27106 file whose contents have already been written to the archive.

27107 **listopt=format**

27108 This keyword specifies the output format of the table of contents produced
 27109 when the **-v** option is specified in **list** mode. See **List Mode Format**
 27110 **Specifications** (on page 2918). To avoid ambiguity, the **listopt=format** shall be
 27111 the only or final **keyword=value** pair in a **-o** option-argument; all characters in
 27112 the remainder of the option-argument shall be considered part of the format
 27113 string. When multiple **-olistopt=format** options are specified, the format
 27114 strings shall be considered a single, concatenated string, evaluated in
 27115 command line order.

27116 **times**

27117 (Applicable only to the **-x pax** format.) When used in **write** or **copy** mode, *pax*
 27118 shall include **atime**, **ctime**, and **mtime** extended header records for each file.
 27119 See **pax Extended Header File Times** (on page 2926).

27120 In addition to these keywords, if the **-x pax** format is specified, any of the
 27121 keywords and values defined in **pax Extended Header** (on page 2923), including
 27122 implementation extensions, can be used in **-o** option-arguments, in either of two
 27123 modes:

27124 **keyword=value**

27125 When used in **write** or **copy** mode, these keyword/value pairs shall be
 27126 included at the beginning of the archive as **typeflag g** global extended header
 27127 records. When used in **read** or **list** mode, these keyword/value pairs shall act
 27128 as if they had been at the beginning of the archive as **typeflag g** global
 27129 extended header records.

27130 **keyword:=value**

27131 When used in **write** or **copy** mode, these keyword/value pairs shall be
 27132 included as records at the beginning of a **typeflag x** extended header for each
 27133 file. (This is equivalent to the equal-sign form except that it creates no
 27134 **typeflag g** global extended header records.) When used in **read** or **list** mode,
 27135 these keyword/value pairs shall act as if they were included as records at the
 27136 end of each extended header; thus, they shall override any global or file-
 27137 specific extended header record keywords of the same names. For example, in
 27138 the command:

```
27139 pax -r -o "  
27140 gname:=mygroup,  
27141 " <archive
```

27142 the group name will be forced to a new value for all files read from the
 27143 archive.

27144 The precedences of **-o** keywords over various fields in the archive are described in
 27145 **pax Extended Header Keyword Precedence** (on page 2925).

27146 **-p string**

27147 Specify one or more file characteristic options (privileges). The *string* option-
 27148 argument shall be a string specifying file characteristics to be retained or discarded
 27149 on extraction. The string shall consist of the specification characters **a**, **e**, **m**, **o**, and
 27150 **p**. Other implementation-defined characters can be included. Multiple
 27151 characteristics can be concatenated within the same string and multiple **-p** options
 can be specified. The meaning of the specification characters are as follows:

- 27152 **a** Do not preserve file access times.
- 27153 **e** Preserve the user ID, group ID, file mode bits (see the Base Definitions volume
27154 of IEEE Std. 1003.1-200x, Section 3.170, File Mode Bits), access time,
27155 modification time, and any other implementation-defined file characteristics.
- 27156 **m** Do not preserve file modification times.
- 27157 **o** Preserve the user ID and group ID.
- 27158 **p** Preserve the file mode bits. Other implementation-defined file mode attributes
27159 may be preserved.
- 27160 In the preceding list, “preserve” indicates that an attribute stored in the archive
27161 shall be given to the extracted file, subject to the permissions of the invoking
27162 process. The access and modification times of the file shall be preserved unless
27163 otherwise specified with the **-p** option or not stored in the archive. All attributes
27164 that are not preserved shall be determined as part of the normal file creation action
27165 (see Section 1.7.1.4 (on page 2209)).
- 27166 If neither the **e** nor the **o** specification character is specified, or the user ID and
27167 group ID are not preserved for any reason, *pax* shall not set the S_ISUID and
27168 S_ISGID bits of the file mode.
- 27169 If the preservation of any of these items fails for any reason, *pax* shall write a
27170 diagnostic message to standard error. Failure to preserve these items shall affect
27171 the final exit status, but shall not cause the extracted file to be deleted.
- 27172 If file characteristic letters in any of the *string* option-arguments are duplicated or
27173 conflict with each other, the ones given last shall take precedence. For example, if
27174 **-p eme** is specified, file modification times are preserved.
- 27175 **-s replstr** Modify file or archive member names named by *pattern* or *file* operands according
27176 to the substitution expression *replstr*, using the syntax of the *ed* utility. The
27177 concepts of “address” and “line” are meaningless in the context of the *pax* utility,
27178 and shall not be supplied. The format shall be:
- 27179 **-s /old/new/[gp]**
- 27180 where as in *ed*, *old* is a basic regular expression and *new* can contain an ampersand,
27181 ‘\n’ (where *n* is a digit) backreferences, or subexpression matching. The *old* string
27182 also shall be permitted to contain <newline> characters.
- 27183 Any non-null character can be used as a delimiter (‘/’ shown here). Multiple **-s**
27184 expressions can be specified; the expressions shall be applied in the order
27185 specified, terminating with the first successful substitution. The optional trailing
27186 ‘g’ is as defined in the *ed* utility. The optional trailing ‘p’ shall cause successful
27187 substitutions to be written to standard error. File or archive member names that
27188 substitute to the empty string shall be ignored when reading and writing archives.
- 27189 **-t** Cause the access times of the archived files to be the same as they were before
27190 being read by *pax*.
- 27191 **-u** Ignore files that are older (having a less recent file modification time) than a pre-
27192 existing file or archive member with the same name. In **read** mode, an archive
27193 member with the same name as a file in the file system shall be extracted if the
27194 archive member is newer than the file. In **write** mode, an archive file member with
27195 the same name as a file in the file system shall be superseded if the file is newer
27196 than the archive member. If **-a** is also specified, this is accomplished by appending

27197 to the archive; otherwise, it is unspecified whether this is accomplished by actual
 27198 replacement in the archive or by appending to the archive. In **copy** mode, the file in
 27199 the destination hierarchy shall be replaced by the file in the source hierarchy or by
 27200 a link to the file in the source hierarchy if the file in the source hierarchy is newer.

27201 **-v** In **list** mode, produce a verbose table of contents (see the STDOUT section).
 27202 Otherwise, write archive member path names to standard error (see the STDERR
 27203 section).

27204 **-x format** Specify the output archive format. The *pax* utility shall support the following
 27205 formats:

27206 **cpio** The *cpio* interchange format; see the EXTENDED DESCRIPTION
 27207 section. The default *blocksize* for this format for character special
 27208 archive files shall be 5120. Implementations shall support all
 27209 *blocksize* values less than or equal to 32 256 that are multiples of 512.

27210 **pax** The *pax* interchange format; see the EXTENDED DESCRIPTION
 27211 section. The default *blocksize* for this format for character special
 27212 archive files shall be 5120. Implementations shall support all
 27213 *blocksize* values less than or equal to 32 256 that are multiples of 512.

27214 **ustar** The *tar* interchange format; see the EXTENDED DESCRIPTION
 27215 section. The default *blocksize* for this format for character special
 27216 archive files shall be 10 240. Implementations shall support all
 27217 *blocksize* values less than or equal to 32 256 that are multiples of 512.

27218 Implementation-defined formats shall specify a default block size as well as any
 27219 other block sizes supported for character special archive files.

27220 Any attempt to append to an archive file in a format different from the existing
 27221 archive format shall cause *pax* to exit immediately with a non-zero exit status.

27222 In **copy** mode, if no **-x** format is specified, *pax* shall behave as if **-xpax** were
 27223 specified.

27224 **-X** When traversing the file hierarchy specified by a path name, *pax* shall not descend
 27225 into directories that have a different device ID (*st_dev*; see the System Interfaces
 27226 volume of IEEE Std. 1003.1-200x, *stat()*).

27227 The options that operate on the names of files or archive members (**-c**, **-i**, **-n**, **-s**, **-u**, and **-v**)
 27228 shall interact as follows. In **read** mode, the archive members shall be selected based on the user-
 27229 specified *pattern* operands as modified by the **-c**, **-n**, and **-u** options. Then, any **-s** and **-i** options
 27230 shall modify, in that order, the names of the selected files. The **-v** option shall write names
 27231 resulting from these modifications.

27232 In **write** mode, the files shall be selected based on the user-specified path names as modified by
 27233 the **-n** and **-u** options. Then, any **-s** and **-i** options shall modify, in that order, the names of
 27234 these selected files. The **-v** option shall write names resulting from these modifications.

27235 If both the **-u** and **-n** options are specified, *pax* shall not consider a file selected unless it is newer
 27236 than the file to which it is compared.

27237 **List Mode Format Specifications**

27238 In **list** mode with the `-o listopt=format` option, the *format* argument shall be applied for each
 27239 selected file. The *pax* utility shall append a <newline> character to the **listopt** output for each
 27240 selected file. The *format* argument shall be used as the *format* string described in the Base
 27241 Definitions volume of IEEE Std. 1003.1-200x, Chapter 5, File Format Notation, with the
 27242 exceptions 1. through 5. defined in the EXTENDED DESCRIPTION section of *printf*, plus the
 27243 following exceptions:

27244 6. The sequence (*keyword*) can occur before a format conversion specifier. The conversion
 27245 argument is defined by the value of *keyword*. The implementation shall support the
 27246 following keywords:

27247 — Any of the Field Name entries in Table 4-13 (on page 2927) and Table 4-15 (on page
 27248 2930). The implementation may support the *cpio* keywords without the leading `c_` in
 27249 addition to the form required by Table 4-16 (on page 2931).

27250 — Any keyword defined for the extended header in **pax Extended Header** (on page 2923).

27251 — Any keyword provided as an implementation-defined extension within the extended
 27252 header defined in **pax Extended Header** (on page 2923).

27253 For example, the sequence "`%(charset)s`" is the string value of the name of the character
 27254 set in the extended header.

27255 The result of the keyword conversion argument shall be the value from the applicable
 27256 header field or extended header, without any trailing NULs.

27257 All keyword values used as conversion arguments shall be translated from the UTF-8
 27258 encoding to the character set appropriate for the local file system, user database, and so on,
 27259 as applicable.

27260 7. An additional conversion character, **T**, shall be used to specify time formats. The **T**
 27261 conversion character can be preceded by the sequence (*keyword=subformat*), where *subformat*
 27262 is a date format as defined by *date* operands. The default *keyword* shall be **mtime** and the
 27263 default subformat shall be:

27264 `%b %e %H:%M %Y`

27265 8. An additional conversion character, **M**, shall be used to specify the file mode string as
 27266 defined in *ls* Standard Output. If (*keyword*) is omitted, the **mode** keyword shall be used. For
 27267 example, `%.1M` writes the single character corresponding to the <entry type> field of the *ls*
 27268 `-l` command.

27269 9. An additional conversion character, **D**, shall be used to specify the device for block or
 27270 special files, if applicable, in an implementation-defined format. If not applicable, and
 27271 (*keyword*) is specified, then this conversion shall be equivalent to `%(keyword)u`. If not
 27272 applicable, and (*keyword*) is omitted, then this conversion shall be equivalent to <space>.

27273 10. An additional conversion character, **F**, shall be used to specify a path name. The **F**
 27274 conversion character can be preceded by a sequence of comma-separated keywords:

27275 `(keyword[,keyword] . . .)`

27276 The values for all the keywords that are non-null shall be concatenated together, each
 27277 separated by a `'/'`. The default shall be (**path**) if the keyword **path** is defined; otherwise,
 27278 the default shall be (**prefix,name**).

27279 11. An additional conversion character, **L**, shall be used to specify a symbolic line expansion. If
 27280 the current file is a symbolic link, then `%L` shall expand to:

27281 "%s -> %s", <value of keyword>, <contents of link>
 27282 Otherwise, the %L conversion character shall be the equivalent of %F.

27283 OPERANDS

27284 The following operands shall be supported:

27285 *directory* The destination directory path name for **copy** mode.
 27286 *file* A path name of a file to be copied or archived.
 27287 *pattern* A pattern matching one or more path names of archive members. A pattern must
 27288 be given in the name-generating notation of the pattern matching notation in
 27289 Section 2.14 (on page 2274), including the file name expansion rules in Section
 27290 2.14.3 (on page 2275). The default, if no *pattern* is specified, is to select all members
 27291 in the archive.

27292 STDIN

27293 In **write** mode, the standard input shall be used only if no *file* operands are specified. It shall be a
 27294 text file containing a list of path names, one per line, without leading or trailing <blank>
 27295 characters.

27296 In **list** and **read** modes, if **-f** is not specified, the standard input shall be an archive file.

27297 Otherwise, the standard input shall not be used.

27298 INPUT FILES

27299 The input file named by the *archive* option-argument, or standard input when the archive is read
 27300 from there, shall be a file formatted according to one of the specifications in the EXTENDED
 27301 DESCRIPTION section or some other implementation-defined format.

27302 The file **/dev/tty** shall be used to write prompts and read responses.

27303 ENVIRONMENT VARIABLES

27304 The following environment variables shall affect the execution of *pax*:

27305 *LANG* Provide a default value for the internationalization variables that are unset or null.
 27306 If *LANG* is unset or null, the corresponding value from the implementation-
 27307 defined default locale shall be used. If any of the internationalization variables
 27308 contains an invalid setting, the utility shall behave as if none of the variables had
 27309 been defined.

27310 *LC_ALL* If set to a non-empty string value, override the values of all the other
 27311 internationalization variables.

27312 *LC_COLLATE*

27313 Determine the locale for the behavior of ranges, equivalence classes and multi-
 27314 character collating elements used in the pattern matching expressions for the
 27315 *pattern* operand, the basic regular expression for the **-s** option, and the extended
 27316 regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES*
 27317 category.

27318 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 27319 characters (for example, single-byte as opposed to multi-byte characters in
 27320 arguments and input files), the behavior of character classes used in the extended
 27321 regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES*
 27322 category, and pattern matching.

27323 *LC_MESSAGES*

27324 Determine the locale for the processing of affirmative responses that should be

- 27325 used to affect the format and contents of diagnostic messages written to standard
27326 error.
- 27327 **LC_TIME** Determine the format and contents of date and time strings when the **-v** option is
27328 specified.
- 27329 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 27330 **TMPDIR** Determine the path name that provides part of the default global extended header
27331 record file, as described for the **-o globexthdr=** keyword as described in the
27332 OPTIONS section.
- 27333 **ASYNCHRONOUS EVENTS**
- 27334 Default.
- 27335 **STDOUT**
- 27336 In **write** mode, if **-f** is not specified, the standard output shall be the archive formatted
27337 according to one of the specifications in the EXTENDED DESCRIPTION section, or some other
27338 implementation-defined format (see **-x format**).
- 27339 In **list** mode, when the **-olistopt=format** has been specified, the selected archive members shall
27340 be written to standard output using the format described under **List Mode Format**
27341 **Specifications** (on page 2918). In **list** mode without the **-olistopt=format** option, the table of
27342 contents of the selected archive members shall be written to standard output using the
27343 following format:
- 27344 "%s\n", <path name>
- 27345 If the **-v** option is specified in **list** mode, the table of contents of the selected archive members
27346 shall be written to standard output using the following formats.
- 27347 For path names representing hard links to previous members of the archive:
- 27348 "%sΔ=Δ%s\n", <ls -l listing>, <linkname>
- 27349 For all other path names:
- 27350 "%s\n", <ls -l listing>
- 27351 where <ls -l listing> shall be the format specified by the *ls* utility with the **-l** option. When
27352 writing path names in this format, it is unspecified what is written for fields for which the
27353 underlying archive format does not have the correct information, although the correct number of
27354 <blank> character-separated fields shall be written.
- 27355 In **list** mode, standard output shall not be buffered more than a line at a time.
- 27356 **STDERR**
- 27357 If **-v** is specified in **read**, **write**, or **copy** modes, *pax* shall write the path names it processes to the
27358 standard error output using the following format:
- 27359 "%s\n", <path name>
- 27360 These path names shall be written as soon as processing is begun on the file or archive member,
27361 and shall be flushed to standard error. The trailing <newline> character, which shall not be
27362 buffered, is written when the file has been read or written.
- 27363 If the **-s** option is specified, and the replacement string has a trailing 'p', substitutions shall be
27364 written to standard error in the following format:
- 27365 "%sΔ>>Δ%s\n", <original path name>, <new path name>

27366 In all operating modes of *pax*, optional messages of unspecified format concerning the input
 27367 archive format and volume number, the number of files, blocks, volumes, and media parts as
 27368 well as other diagnostic messages may be written to standard error.

27369 In all formats, for both standard output and standard error, it is unspecified how non-printable
 27370 characters in path names or link names are written.

27371 When *pax* is in **read** mode or **list** mode, using the **-xpax** archive format, and a file name, link
 27372 name, owner name, or any other field in an extended header record cannot be translated from
 27373 the *pax* UTF-8 codeset format to the codeset and current locale of the implementation, *pax* shall
 27374 write a diagnostic message to standard error, shall process the file as described for the **-o**
 27375 **invalid=option**, and then shall process the next file in the archive.

27376 OUTPUT FILES

27377 In **read** mode, the extracted output files shall be of the archived file type. In **copy** mode, the
 27378 copied output files shall be the type of the file being copied. In either mode, existing files in the
 27379 destination hierarchy shall be overwritten only when all permission (**-p**), modification time (**-u**),
 27380 and invalid-value (**-oinvalid=**) tests allow it.

27381 In **write** mode, the output file named by the **-f** option-argument shall be a file formatted
 27382 according to one of the specifications in the EXTENDED DESCRIPTION section, or some other
 27383 implementation-defined format.

27384 EXTENDED DESCRIPTION

27385 **pax Interchange Format**

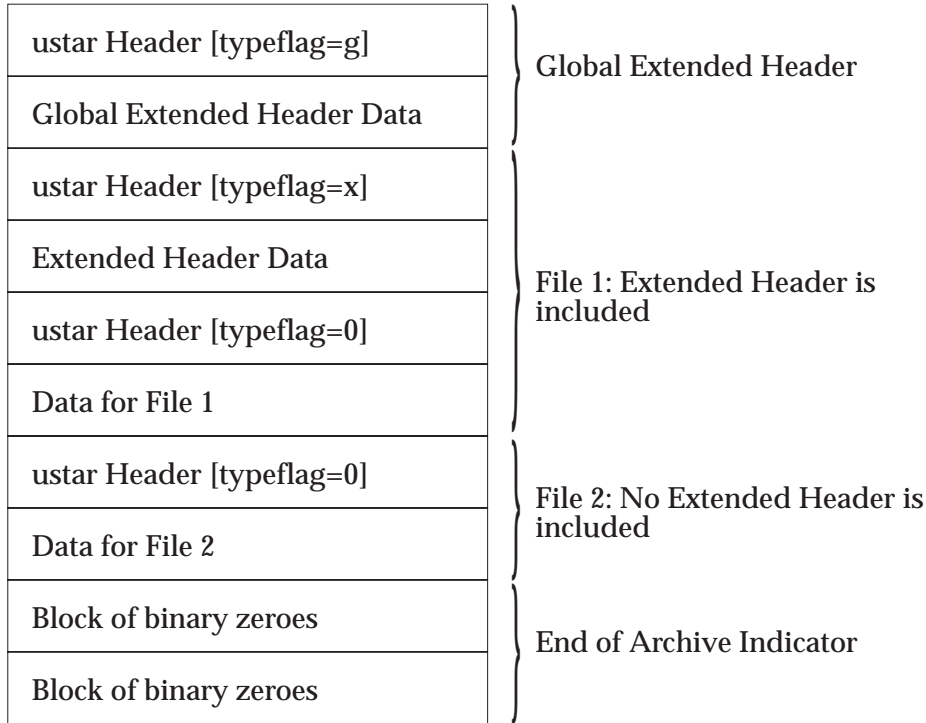
27386 A *pax* archive tape or file produced in the **-xpax** format shall contain a series of blocks. The
 27387 physical layout of the archive shall be identical to the **ustar** format described in **ustar**
 27388 **Interchange Format** (on page 2926). Each file archived shall be represented by the following
 27389 sequence:

- 27390 • An optional header block with extended header records. This header block is of the form
 27391 described in **pax Header Block** (on page 2922), with a *typflag* value of **x** or **g**. The extended
 27392 header records, described in **pax Extended Header** (on page 2923), are included as the data
 27393 for this header block.
- 27394 • A header block that describes the file. Any fields in the preceding optional extended header
 27395 override the associated fields in this header block for this file.
- 27396 • Zero or more blocks that contain the contents of the file.

27397 At the end of the archive file there shall be two 512-byte blocks filled with binary zeroes,
 27398 interpreted as an end-of-archive indicator.

27399 A schematic of an example archive with global extended header records and two actual files is
 27400 shown in Figure 4-1 (on page 2922). In the example, the second file in the archive has no
 27401 extended header preceding it, presumably because it has no need for extended attributes.

27402



27403

Figure 4-1 pax Format Archive Example

27404

pax Header Block

27405
27406

The *pax* header block shall be identical to the **ustar** header block described in **ustar Interchange Format** (on page 2926), except that two additional *typeflag* values are defined:

27407
27408
27409

x Represents extended header records for the following file in the archive (which shall have its own **ustar** header block). The format of these extended header records shall be as described in **pax Extended Header** (on page 2923).

27410
27411
27412
27413
27414
27415

g Represents global extended header records for the following files in the archive. The format of these extended header records shall be as described in **pax Extended Header** (on page 2923). Each value shall affect all subsequent files that do not override that value in their own extended header record and until another global extended header record is reached that provides another value for the same field. The *typeflag g* global headers should not be used with interchange media that could suffer partial data loss in transporting the archive.

27416
27417
27418
27419
27420
27421

For both of these types, the *size* field shall be the size of the extended header records in octets. The other fields in the header block are not meaningful to this version of the *pax* utility. However, if this archive is read by a *pax* utility conforming to a previous version of IEEE Std. 1003.1-200x, the header block fields are used to create a regular file that contains the extended header records as data. Therefore, header block field values should be selected to provide reasonable file access to this regular file.

27422
27423
27424

A further difference from the **ustar** header block is that data blocks for files of *typeflag 1* (the digit one) (hard link) may be included, which means that the *size* field may be greater than zero. Archives created by **pax -o linkdata** shall include these data blocks with the hard links.

27425 **pax Extended Header**

27426 A *pax* extended header contains values that are inappropriate for the **ustar** header block because
 27427 of limitations in that format: fields requiring a character encoding other than that described in
 27428 the ISO/IEC 646:1991 standard, fields representing file attributes not described in the **ustar**
 27429 header, and fields whose format or length do not fit the requirements of the **ustar** header. The
 27430 values in an extended header add attributes to the following file (or files; see the description of
 27431 the *typeflag g* header block) or override values in the following header block(s), as indicated in
 27432 the following list of keywords.

27433 An extended header shall consist of one or more records, each constructed as follows:

27434 "%d %s=%s\n", <length>, <keyword>, <value>

27435 The extended header records shall be encoded according to the ISO/IEC 10646-1:1993 standard
 27436 (UTF-8). The <length> field, <blank> character, equals sign, and <newline> character shown
 27437 shall be limited to the portable character set, as encoded in UTF-8. The <keyword> and <value>
 27438 fields can be any UTF-8 characters. The <length> field shall be the decimal length of the extended
 27439 header record in octets, including the trailing <newline> character.

27440 The <keyword> field shall be one of the entries from the following list or a keyword provided as
 27441 an implementation extension. Keywords consisting entirely of lowercase letters, digits, and
 27442 periods are reserved for future standardization. A keyword shall not include an equals sign. (In
 27443 the following list, the notations “file(s)” or “block(s)” is used to acknowledge that a keyword
 27444 affects the following single file after a *typeflag x* extended header, but possibly multiple files after
 27445 *typeflag g*. Any requirements in the list for *pax* to include a record when in **write** or **copy** mode
 27446 shall apply only when such a record has not already been provided through the use of the **-o**
 27447 option. When used in **copy** mode, *pax* shall behave as if an archive had been created with
 27448 applicable extended header records and then extracted.)

27449 **atime** The file access time for the following file(s), equivalent to the value of the *st_atime*
 27450 member of the **stat** structure for a file, as described by the *stat()* function. The
 27451 access time shall be restored if the process has the appropriate privilege required
 27452 to do so. The format of the <value> shall be as described in **pax Extended Header**
 27453 **File Times** (on page 2926).

27454 **charset** The name of the character set used to encode the data in the following file(s). The
 27455 entries in the following table are defined to refer to known standards; additional
 27456 names may be agreed on between the originator and recipient.

<value>	Formal Standard
ISO-IRΔ646Δ1990	ISO/IEC 646: 1990
ISO-IRΔ8859Δ1Δ1987	ISO/IEC 8859-1: 1987
ISO-IRΔ8859Δ2Δ1987	ISO/IEC 8859-2: 1987
ISO-IRΔ10646Δ1993	ISO/IEC 10646: 1993
ISO-IRΔ10646Δ1993ΔUTF-8	ISO/IEC 10646, UTF-8 encoding
BINARY	None.

27464 The encoding is included in an extended header for information only; when *pax* is
 27465 used as described in IEEE Std. 1003.1-200x, it shall not translate the file data into
 27466 any other encoding. The **BINARY** entry indicates unencoded binary data.

27467 When used in **write** or **copy** mode, it is implementation-defined whether *pax*
 27468 includes a **charset** extended header record for a file.

27469 **comment** A series of characters used as a comment. All characters in the <value> field shall
 27470 be ignored by *pax*.

27471	ctime	The file creation time for the following file(s), equivalent to the value of the <i>st_ctime</i> member of the stat structure for a file, as described by the <i>stat()</i> function. The creation time shall be restored if the process has the appropriate privilege required to do so. The format of the <i><value></i> shall be as described in pax Extended Header File Times (on page 2926).
27472		
27473		
27474		
27475		
27476	gid	The group ID of the group that owns the file, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the <i>gid</i> field in the following header block(s). When used in write or copy mode, <i>pax</i> shall include a <i>gid</i> extended header record for each file whose group ID is greater than 2 097 151 (octal 7 777 777).
27477		
27478		
27479		
27480		
27481	gname	The group of the file(s), formatted as a group name in the group database. This record shall override the <i>gid</i> and <i>gname</i> fields in the following header block(s), and any <i>gid</i> extended header record. When used in read , copy , or list mode, <i>pax</i> shall translate the name from the UTF-8 encoding in the header record to the character set appropriate for the group database on the receiving system. If any of the UTF-8 characters cannot be translated, and if the -oinvalid=UTF-8 option is not specified, the results are implementation-defined. When used in write or copy mode, <i>pax</i> shall include a gname extended header record for each file whose group name cannot be represented entirely with the letters and digits of the portable character set.
27482		
27483		
27484		
27485		
27486		
27487		
27488		
27489		
27490		
27491	linkpath	The path name of a link being created to another file, of any type, previously archived. This record shall override the <i>linkname</i> field in the following ustar header block(s). The following ustar header block shall determine the type of link created. If <i>typeflag</i> of the following header block is 1, it shall be a hard link. If <i>typeflag</i> is 2, it shall be a symbolic link and the linkpath value shall be the contents of the symbolic link. The <i>pax</i> utility shall translate the name of the link (contents of the symbolic link) from the UTF-8 encoding to the character set appropriate for the local file system. When used in write or copy mode, <i>pax</i> shall include a linkpath extended header record for each link whose path name cannot be represented entirely with the members of the portable character set other than NUL.
27492		
27493		
27494		
27495		
27496		
27497		
27498		
27499		
27500		
27501	mtime	The file modification time of the following file(s), equivalent to the value of the <i>st_mtime</i> member of the stat structure for a file, as described in the <i>stat()</i> function. This record shall override the <i>mtime</i> field in the following header block(s). The modification time shall be restored if the process has the appropriate privilege required to do so. The format of the <i><value></i> shall be as described in pax Extended Header File Times (on page 2926).
27502		
27503		
27504		
27505		
27506		
27507	path	The path name of the following file(s). This record shall override the <i>name</i> and <i>prefix</i> fields in the following header block(s). The <i>pax</i> utility shall translate the path name of the file from the UTF-8 encoding to the character set appropriate for the local file system. When used in write or copy mode, <i>pax</i> shall include a <i>path</i> extended header record for each file whose path name cannot be represented entirely with the members of the portable character set other than NUL.
27508		
27509		
27510		
27511		
27512	realtime.any	The keywords prefixed by “realtime.” are reserved for future standardization.
27513		
27514	security.any	The keywords prefixed by “security.” are reserved for future standardization.
27515		
27516	size	The size of the file in octets, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the <i>size</i> field in the following header block(s). When used in write or copy mode, <i>pax</i> shall include a
27517		
27518		
27518		

- 27519 *size* extended header record for each file with a size value greater than 8 589 934 591
27520 (octal 7 777 777 777).
- 27521 **uid** The user ID of the file owner, expressed as a decimal number using digits from the
27522 ISO/IEC 646:1991 standard. This record shall override the *uid* field in the
27523 following header block(s). When used in **write** or **copy** mode, *pax* shall include a
27524 *uid* extended header record for each file whose owner ID is greater than 2 097 151
27525 (octal 7 777 777).
- 27526 **uname** The owner of the following file(s), formatted as a user name in the user database.
27527 This record shall override the *uid* and *uname* fields in the following header block(s),
27528 and any *uid* extended header record. When used in **read**, **copy**, or **list** mode, *pax*
27529 shall translate the name from the UTF-8 encoding in the header record to the
27530 character set appropriate for the user database on the receiving system. If any of
27531 the UTF-8 characters cannot be translated, and if the **-oinvalid=** UTF-8 option is
27532 not specified, the results are implementation-defined. When used in **write** or **copy**
27533 mode, *pax* shall include a **uname** extended header record for each file whose user
27534 name cannot be represented entirely with the letters and digits of the portable
27535 character set.
- 27536 If the *<value>* field is zero length, it shall delete any header block field, previously entered
27537 extended header value, or global extended header value of the same name.
- 27538 If a keyword in an extended header record (or in a **-o** option-argument) overrides or deletes a
27539 corresponding field in the **ustar** header block, *pax* shall ignore the contents of that header block
27540 field.
- 27541 Unlike the **ustar** header block fields, NULs shall not delimit *<value>*s; all characters within the
27542 *<value>* field shall be considered data for the field. None of the length limitations of the **ustar**
27543 header block fields in Table 4-13 (on page 2927) shall apply to the extended header records.
- 27544 **pax Extended Header Keyword Precedence**
- 27545 This section describes the precedence in which the various header records and fields and
27546 command line options are selected to apply to a file in the archive. When *pax* is used in **read** or
27547 **list** modes, it shall determine a file attribute in the following sequence:
- 27548 1. If **-odelete=keyword-prefix** is used, the affected attributes shall be determined from step 7.,
27549 if applicable, or ignored otherwise.
 - 27550 2. If **-okeyword:=** is used, the affected attributes shall be ignored.
 - 27551 3. If **-okeyword:=value** is used, the affected attribute shall be assigned the value.
 - 27552 4. If there is a *typeflag* **x** extended header record, the affected attribute shall be assigned the
27553 *<value>*. When extended header records conflict, the last one given in the header shall take
27554 precedence.
 - 27555 5. If **-okeyword=value** is used, the affected attribute shall be assigned the value.
 - 27556 6. If there is a *typeflag* **g** global extended header record, the affected attribute shall be
27557 assigned the *<value>*. When global extended header records conflict, the last one given in
27558 the global header shall take precedence.
 - 27559 7. Otherwise, the attribute shall be determined from the **ustar** header block.

27560 **pax Extended Header File Times**27561 **Notes to Reviewers**

27562 *This section with side shading will not appear in the final copy. - Ed.*

27563 D3, XCU, ERN 158 proposes new wording for the first half of the following paragraph: "pax shall
27564 write an mtime record for each file in write or copy modes if the file's modification time cannot
27565 be represented exactly in the ustar header block described in ustar Interchange Format. This can
27566 occur if the time is out of ustar range, or if the file system of the underlying implementation
27567 supports non-integer time granularities and the time is not an integer."

27568 The *pax* utility shall write **atime** and **ctime** records for each file in **write** or **copy** modes only if
27569 the **-otimes** option is specified; *pax* shall write a **mtime** record for each file in **write** or **copy**
27570 modes if the file system of the underlying implementation supports time granularities smaller
27571 than that required by the **ustar** header block described in **ustar Interchange Format**. All of these
27572 time records shall be formatted as a decimal representation of the time in seconds since the
27573 Epoch. If a period (' . ') decimal point character is present, the digits to the right of the point
27574 shall represent the units of a subsecond timing granularity, where the first digit is tenths of a
27575 second and each subsequent digit is a tenth of the previous digit. Implementations may ignore
27576 any portion of the subsecond digits for which they do not support the necessary timing
27577 granularity; they shall not perform any rounding operation.

27578 **Notes to Reviewers**

27579 *This section with side shading will not appear in the final copy. - Ed.*

27580 D3, XCU, ERN 173, proposes new text for the previous sentence because a *pax* implementation
27581 on a single platform should not be allowed to lose information when it writes an extended
27582 header time and then reads it back in again: "In read or copy mode, the *pax* utility shall truncate
27583 the time of a file to the greatest value that is not greater than the input header file time. In write
27584 or copy mode, the *pax* utility shall output a time exactly if it can be represented exactly as a
27585 decimal number, and otherwise shall generate only enough digits so that the same time shall be
27586 recovered if the file is extracted on a system whose underlying implementation supports the
27587 same time granularity."

27588 **ustar Interchange Format**

27589 A **ustar** archive tape or file shall contain a series of blocks. Each block shall be a fixed-size block
27590 of 512 octets (see below). Although this format may be thought of as being stored on 9-track
27591 industry-standard 12.7mm (0.5in) magnetic tape, other types of transportable media are not
27592 excluded. Each file archived shall be represented by a header block that describes the file,
27593 followed by zero or more blocks that give the contents of the file. At the end of the archive file
27594 there shall be two 512-octet blocks filled with binary zeros, interpreted as an end-of-archive
27595 indicator.

27596 The blocks may be grouped for physical I/O operations, as described under the **-blocksize** and
27597 **-x ustar** options. Each group of blocks may be written with a single operation equivalent to the
27598 *write()* function. On magnetic tape, the result of this write shall be a single tape record. The last
27599 group of blocks always shall be at the full size, so blocks after the two zero blocks may contain
27600 undefined data.

27601 The header block shall be structured as shown in the following table. All lengths and offsets are
27602 in decimal.

27603 **Table 4-13** ustar Header Block

27604
27605
27606
27607
27608
27609
27610
27611
27612
27613
27614
27615
27616
27617
27618
27619
27620
27621

Field Name	Octet Offset	Length (in Octets)
<i>name</i>	0	100
<i>mode</i>	100	8
<i>uid</i>	108	8
<i>gid</i>	116	8
<i>size</i>	124	12
<i>mtime</i>	136	12
<i>chksum</i>	148	8
<i>typeflag</i>	156	1
<i>linkname</i>	157	100
<i>magic</i>	257	6
<i>version</i>	263	2
<i>uname</i>	265	32
<i>gname</i>	297	32
<i>devmajor</i>	329	8
<i>devminor</i>	337	8
<i>prefix</i>	345	155

27622
27623
27624
27625
27626
27627
27628

All characters in the header block shall be represented in the coded character set of the ISO/IEC 646:1991 standard. For maximum portability between implementations, names should be selected from characters represented by the portable file name character set as octets with the most significant bit zero. If an implementation supports the use of characters outside of slash and the portable file name character set in names for files, users, and groups, one or more implementation-defined encodings of these characters shall be provided for interchange purposes.

27629
27630
27631
27632
27633

However, the *pax* utility shall never create file names on the local system that cannot be accessed via the procedures described in IEEE Std. 1003.1-200x. If a file name is found on the medium that would create an invalid file name, it is implementation-defined whether the data from the file is stored on the file hierarchy and under what name it is stored. The *pax* utility may choose to ignore these files as long as it produces an error indicating that the file is being ignored.

27634
27635

Each field within the header block is contiguous; that is, there is no padding used. Each character on the archive medium shall be stored contiguously.

27636
27637
27638
27639
27640
27641

The fields *magic*, *uname*, and *gname* are character strings each terminated by a NUL character. The fields *name*, *linkname*, and *prefix* are NUL-terminated character strings except when all characters in the array contain non-NUL characters including the last character. The *version* field is two octets containing the characters "00" (zero-zero). The *typeflag* contains a single character. All other fields are leading zero-filled octal numbers using digits from the ISO/IEC 646:1991 standard IRV. Each numeric field is terminated by one or more <space> or NUL characters.

27642
27643
27644
27645
27646
27647
27648

The *name* and the *prefix* fields shall produce the path name of the file. A new path name shall be formed, if *prefix* is not an empty string (its first character is not NUL), by concatenating *prefix* (up to the first NUL character), a slash character, and *name*; otherwise, *name* is used alone. In either case, *name* is terminated at the first NUL character. If *prefix* begins with a NUL character, it shall be ignored. In this manner, path names of at most 256 characters can be supported. If a path name does not fit in the space provided, *pax* shall notify the user of the error, and shall not store any part of the file—header or data—on the medium.

27649
27650
27651

The *linkname* field, described below, shall not use the *prefix* to produce a path name. As such, a *linkname* is limited to 100 characters. If the name does not fit in the space provided, *pax* shall notify the user of the error, and shall not attempt to store the link on the medium.

27652 The *mode* field provides 12 bits encoded in the ISO/IEC 646:1991 standard octal digit
27653 representation. The encoded bits shall represent the following values:

27654 **Table 4-14** ustar *mode* Field

27655	Bit Value	IEEE Std. 1003.1-200x Bit	Description
27656	04 000	S_ISUID	Set UID on execution.
27657	02 000	S_ISGID	Set GID on execution.
27658	01 000	<reserved>	Reserved for future standardization.
27659	00 400	S_IRUSR	Read permission for file owner class.
27660	00 200	S_IWUSR	Write permission for file owner class.
27661	00 100	S_IXUSR	Execute/search permission for file owner class.
27662	00 040	S_IRGRP	Read permission for file group class.
27663	00 020	S_IWGRP	Write permission for file group class.
27664	00 010	S_IXGRP	Execute/search permission for file group class.
27665	00 004	S_IROTH	Read permission for file other class.
27666	00 002	S_IWOTH	Write permission for file other class.
27667	00 001	S_IXOTH	Execute/search permission for file other class.

27668 When appropriate privilege is required to set one of these mode bits, and the user restoring the
27669 files from the archive does not have the appropriate privilege, the mode bits for which the user
27670 does not have appropriate privilege shall be ignored. Some of the mode bits in the archive
27671 format are not mentioned elsewhere in this volume of IEEE Std. 1003.1-200x. If the
27672 implementation does not support those bits, they may be ignored.

27673 The *uid* and *gid* fields are the user and group ID of the owner and group of the file, respectively.

27674 The *size* field is the size of the file in octets. If the *typeflag* field is set to specify a file to be of type
27675 1 (a link) or 2 (reserved for symbolic links), the *size* field shall be specified as zero. If the *typeflag*
27676 field is set to specify a file of type 5 (directory), the *size* field shall be interpreted as described
27677 under the definition of that record type. No data blocks are stored for types 1, 2, or 5. If the
27678 *typeflag* field is set to 3 (character special file), 4 (block special file), or 6 (FIFO), the meaning of
27679 the *size* field is unspecified by this volume of IEEE Std. 1003.1-200x, and no data blocks shall be
27680 stored on the medium. Additionally, for type 6, the *size* field shall be ignored when reading. If
27681 the *typeflag* field is set to any other value, the number of blocks written following the header
27682 shall be $(size+511)/512$, ignoring any fraction in the result of the division.

27683 The *mtime* field shall be the modification time of the file at the time it was archived. It is the
27684 ISO/IEC 646:1991 standard representation of the octal value of the modification time obtained
27685 from the *stat()* function.

27686 The *chksum* field shall be the ISO/IEC 646:1991 standard IRV representation of the octal value of
27687 the simple sum of all octets in the header block. Each octet in the header shall be treated as an
27688 unsigned value. These values shall be added to an unsigned integer, initialized to zero, the
27689 precision of which is not less than 17 bits. When calculating the checksum, the *chksum* field is
27690 treated as if it were all spaces.

27691 The *typeflag* field specifies the type of file archived. If a particular implementation does not
27692 recognize the type, or the user does not have appropriate privilege to create that type, the file
27693 shall be extracted as if it were a regular file if the file type is defined to have a meaning for the
27694 *size* field that could cause data blocks to be written on the medium (see the previous description
27695 for *size*). If conversion to a regular file occurs, the *pax* utility shall produce an error indicating
27696 that the conversion took place. All of the *typeflag* fields shall be coded in the ISO/IEC 646:1991
27697 standard IRV:

27698	0	Represents a regular file. For backward compatibility, a <i>typeflag</i> value of binary zero (' <code>\0</code> ') should be recognized as meaning a regular file when extracting files from the archive. Archives written with this version of the archive file format create regular files with a <i>typeflag</i> value of the ISO/IEC 646:1991 standard IRV ' <code>0</code> '.
27699		
27700		
27701		
27702	1	Represents a file linked to another file, of any type, previously archived. Such files are identified by each file having the same device and file serial number. The linked-to name is specified in the <i>linkname</i> field with a NUL-character terminator if it is less than 100 octets in length.
27703		
27704		
27705		
27706	2	Represents a symbolic link. The contents of the symbolic link shall be stored in the <i>linkname</i> field.
27707		
27708	3, 4	Represent character special files and block special files respectively. In this case the <i>devmajor</i> and <i>devminor</i> fields shall contain information defining the device, the format of which is unspecified by this volume of IEEE Std. 1003.1-200x. Implementations may map the device specifications to their own local specification or may ignore the entry.
27709		
27710		
27711		
27712	5	Specifies a directory or subdirectory. On systems where disk allocation is performed on a directory basis, the <i>size</i> field shall contain the maximum number of octets (which may be rounded to the nearest disk block allocation unit) that the directory may hold. A <i>size</i> field of zero indicates no such limiting. Systems that do not support limiting in this manner should ignore the <i>size</i> field.
27713		
27714		
27715		
27716		
27717	6	Specifies a FIFO special file. Note that the archiving of a FIFO file archives the existence of this file and not its contents.
27718		
27719	7	Reserved to represent a file to which an implementation has associated some high-performance attribute. Implementations without such extensions should treat this file as a regular file (type 0).
27720		
27721		
27722	A-Z	The letters ' <code>A</code> ' to ' <code>Z</code> ', inclusive, are reserved for custom implementations. All other values are reserved for future revisions of IEEE Std. 1003.1-200x.
27723		
27724		The <i>magic</i> field is the specification that this archive was output in this archive format. If this field contains ustar (the five characters from the ISO/IEC 646:1991 standard IRV shown followed by NUL), the <i>uname</i> and <i>gname</i> fields shall contain the ISO/IEC 646:1991 standard IRV representation of the owner and group of the file, respectively (truncated to fit, if necessary). When the file is restored by a privileged, protection-preserving version of the utility, the user and group databases shall be scanned for these names. If found, the user and group IDs contained within these files shall be used rather than the values contained within the <i>uid</i> and <i>gid</i> fields.
27725		
27726		
27727		
27728		
27729		
27730		
27731		
27732		cpio Interchange Format
27733		The octet-oriented <i>cpio</i> archive format shall be a series of entries, each comprising a header that describes the file, the name of the file, and then the contents of the file.
27734		
27735		An archive may be recorded as a series of fixed-size blocks of octets. This blocking shall be used only to make physical I/O more efficient. The last group of blocks shall be always at the full size.
27736		
27737		
27738		For the octet-oriented <i>cpio</i> archive format, the individual entry information shall be in the order indicated and described by the following table; see also the <code><cpio.h></code> header.
27739		

27740

Table 4-15 Octet-Oriented *cpio* Archive Entry

27741

27742

27743

27744

27745

27746

27747

27748

27749

27750

27751

27752

27753

27754

27755

27756

Header Field Name	Length (in Octets)	Interpreted as
<i>c_magic</i>	6	Octal number
<i>c_dev</i>	6	Octal number
<i>c_ino</i>	6	Octal number
<i>c_mode</i>	6	Octal number
<i>c_uid</i>	6	Octal number
<i>c_gid</i>	6	Octal number
<i>c_nlink</i>	6	Octal number
<i>c_rdev</i>	6	Octal number
<i>c_mtime</i>	11	Octal number
<i>c_namesize</i>	6	Octal number
<i>c_filesize</i>	11	Octal number
File Name Field Name	Length	Interpreted as
<i>c_name</i>	<i>c_namesize</i>	Path name string
File Data Field Name	Length	Interpreted as
<i>c_filedata</i>	<i>c_filesize</i>	Data

27757

cpio Header

27758

27759

27760

27761

27762

27763

For each file in the archive, a header as defined previously shall be written. The information in the header fields is written as streams of the ISO/IEC 646: 1991 standard characters interpreted as octal numbers. The octal numbers shall be extended to the necessary length by appending the ISO/IEC 646: 1991 standard IRV zeros at the most-significant-digit end of the number; the result is written to the most-significant digit of the stream of octets first. The fields shall be interpreted as follows:

27764

27765

c_magic Identify the archive as being a transportable archive by containing the identifying value "070707".

27766

27767

27768

c_dev, c_ino Contains values that uniquely identify the file within the archive (that is, no files contain the same pair of *c_dev* and *c_ino* values unless they are links to the same file). The values shall be determined in an unspecified manner.

27769

c_mode Contains the file type and access permissions as defined in the following table.

27770

Table 4-16 Values for *cpio c_mode* Field

27771

27772

27773

27774

27775

27776

27777

27778

27779

27780

27781

27782

27783

27784

27785

27786

27787

27788

27789

27790

27791

27792

File Permissions Name	Value	Indicates
C_IRUSR	000 400	Read by owner
C_IWUSR	000 200	Write by owner
C_IXUSR	000 100	Execute by owner
C_IRGRP	000 040	Read by group
C_IWGRP	000 020	Write by group
C_IXGRP	000 010	Execute by group
C_IROTH	000 004	Read by others
C_IWOTH	000 002	Write by others
C_IXOTH	000 001	Execute by others
C_ISUID	004 000	Set <i>uid</i>
C_ISGID	002 000	Set <i>gid</i>
C_ISVTX	001 000	Reserved
File Type Name	Value	Indicates
C_ISDIR	040 000	Directory
C_ISFIFO	010 000	FIFO
C_ISREG	0100 000	Regular file
C_ISBLK	060 000	Block special file
C_ISCHR	020 000	Character special file
C_ISCTG	0110 000	Reserved
C_ISLNK	0120 000	Reserved
C_ISSOCK	0140 000	Reserved

27793

27794

27795

27796

27797

Directories, FIFOs, and regular files shall be supported on a system conforming to this volume of IEEE Std. 1003.1-200x; additional values defined previously are reserved for compatibility with existing systems. Additional file types may be supported; however, such files should not be written to archives intended to be transported to other systems.

27798

c_uid

Contains the user ID of the owner.

27799

c_gid

Contains the group ID of the group.

27800

27801

c_nlink

Contains the number of links referencing the file at the time the archive was created.

27802

c_rdev

Contains implementation-defined information for character or block special files.

27803

27804

c_mtime

Contains the latest time of modification of the file at the time the archive was created.

27805

c_namesize

Contains the length of the path name, including the terminating NUL character.

27806

27807

c_filesize

Contains the length of the file in octets. This shall be the length of the data section following the header structure.

27808	cpio File Name
27809 27810	The <i>c_name</i> field shall contain the path name of the file. The length of this field in octets is the value of <i>c_namesize</i> .
27811 MAN 27812 27813	If a file name is found on the medium that would create an invalid path name, it is implementation-defined whether the data from the file is stored on the file hierarchy and under what name it is stored.
27814 27815 27816 27817 27818 27819 27820 27821 27822 27823 27824	All characters shall be represented in the ISO/IEC 646:1991 standard IRV. For maximum portability between implementations, names should be selected from characters represented by the portable file name character set as octets with the most significant bit zero. If an implementation supports the use of characters outside the portable file name character set in names for files, users, and groups, one or more implementation-defined encodings of these characters shall be provided for interchange purposes. However, the <i>pax</i> utility shall never create file names on the local system that cannot be accessed via the procedures described previously in this volume of IEEE Std. 1003.1-200x. If a file name is found on the medium that would create an invalid file name, it is implementation-defined whether the data from the file is stored on the local file system and under what name it is stored. The <i>pax</i> utility may choose to ignore these files as long as it produces an error indicating that the file is being ignored.
27825	cpio File Data
27826 27827	Following <i>c_name</i> , there shall be <i>c_filesize</i> octets of data. Interpretation of such data occurs in a manner dependent on the file. If <i>c_filesize</i> is zero, no data shall be contained in <i>c_filedata</i> .
27828 MAN	When restoring from an archive:
27829 27830	<ul style="list-style-type: none"> • If the user does not have the appropriate privilege to create a file of the specified type, <i>pax</i> shall ignore the entry and write an error message to standard error.
27831 27832 27833	<ul style="list-style-type: none"> • Only regular files have data to be restored. Presuming a regular file meets any selection criteria that might be imposed on the format-reading utility by the user, such data shall be restored.
27834 27835 27836 27837 27838	<ul style="list-style-type: none"> • If a user does not have appropriate privilege to set a particular mode flag, the flag shall be ignored. Some of the mode flags in the archive format are not mentioned elsewhere in this volume of IEEE Std. 1003.1-200x. If the implementation does not support those flags, they may be ignored.
27839	cpio Special Entries
27840 27841 27842 27843 27844	FIFO special files, directories, and the trailer shall be recorded with <i>c_filesize</i> equal to zero. For other special files, <i>c_filesize</i> is unspecified by this volume of IEEE Std. 1003.1-200x. The header for the next file entry in the archive shall be written directly after the last octet of the file entry preceding it. A header denoting the file name TRAILER!!! indicates the end of the archive; the contents of octets in the last block of the archive following such a header are undefined.
27845	EXIT STATUS
27846	The following exit values shall be returned:
27847	0 All files were processed successfully.
27848	>0 An error occurred.

27849 CONSEQUENCES OF ERRORS

27850 If *pax* cannot create a file or a link when reading an archive or cannot find a file when writing an
 27851 archive, or cannot preserve the user ID, group ID, or file mode when the **-p** option is specified, a
 27852 diagnostic message shall be written to standard error and a non-zero exit status shall be
 27853 returned, but processing shall continue. In the case where *pax* cannot create a link to a file, *pax*
 27854 shall not, by default, create a second copy of the file.

27855 If the extraction of a file from an archive is prematurely terminated by a signal or error, *pax* may
 27856 have only partially extracted the file or (if the **-n** option was not specified) may have extracted a
 27857 file of the same name as that specified by the user, but which is not the file the user wanted.
 27858 Additionally, the file modes of extracted directories may have additional bits from the S_IRWXU
 27859 mask set as well as incorrect modification and access times.

27860 APPLICATION USAGE

27861 The **-p** (privileges) option was invented to reconcile differences between historical *tar* and *cpio*
 27862 implementations. In particular, the two utilities use **-m** in diametrically opposed ways. The **-p**
 27863 option also provides a consistent means of extending the ways in which future file attributes can
 27864 be addressed, such as for enhanced security systems or high-performance files. Although it may
 27865 seem complex, there are really two modes that are most commonly used:

27866 **-p e** “Preserve everything”. This would be used by the historical superuser, someone with
 27867 all the appropriate privileges, to preserve all aspects of the files as they are recorded in
 27868 the archive. The **e** flag is the sum of **o** and **p**, and other implementation-defined
 27869 attributes.

27870 **-p p** “Preserve” the file mode bits. This would be used by the user with regular privileges
 27871 who wished to preserve aspects of the file other than the ownership. The file times are
 27872 preserved by default, but two other flags are offered to disable these and use the time
 27873 of extraction.

27874 The one path name per line format of standard input precludes path names containing
 27875 <newline> characters. Although such path names violate the portable file name guidelines, they
 27876 may exist and their presence may inhibit usage of *pax* within shell scripts. This problem is
 27877 inherited from historical archive programs. The problem can be avoided by listing file name
 27878 arguments on the command line instead of on standard input.

27879 It is almost certain that appropriate privileges are required for *pax* to accomplish parts of this
 27880 volume of IEEE Std. 1003.1-200x. Specifically, creating files of type block special or character
 27881 special, restoring file access times unless the files are owned by the user (the **-t** option), or
 27882 preserving file owner, group, and mode (the **-p** option) all probably require appropriate
 27883 privileges.

27884 In **read** mode, implementations are permitted to overwrite files when the archive has multiple
 27885 members with the same name. This may fail if permissions on the first version of the file do not
 27886 permit it to be overwritten.

27887 The **cpio** and **ustar** formats can only support files up to 8 gigabytes in size.

27888 EXAMPLES

27889 The following command:

```
27890 pax -w -f /dev/rmt/1m .
```

27891 copies the contents of the current directory to tape drive 1, medium density (assuming historical
 27892 System V device naming procedures. The historical BSD device name would be **/dev/rmt9**).

27893 The following commands:

27894 `mkdir newdir`
 27895 `pax -rw olddir newdir`

27896 copy the *olddir* directory hierarchy to *newdir*.

27897 `pax -r -s ',^//*usr//*,,' -f a.pax`

27898 reads the archive **a.pax**, with all files rooted in **/usr** in the archive extracted relative to the current
 27899 directory.

27900 Using the option:

27901 `-o listopt="%M %(atime)T %(size)D %(name)s"`

27902 overrides the default output description in Standard Output and instead writes:

27903 `-rw-rw--- Jan 12 15:53 1492 /usr/foo/bar`

27904 Using the options:

27905 `-o listopt='%L\t%(size)D\n%.7' \`
 27906 `-o listopt='(name)s\n%(ctime)T\n%T'`

27907 overrides the default output description in Standard Output and instead writes:

27908 `/usr/foo/bar -> /tmp 1492`
 27909 `/usr/fo`
 27910 `Jan 12 1991`
 27911 `Jan 31 15:53`

27912 **RATIONALE**

27913 The *pax* utility was new, commissioned for the ISO POSIX-2:1993 standard. It represents a
 27914 peaceful compromise between advocates of the historical *tar* and *cpio* utilities.

27915 A fundamental difference between *cpio* and *tar* was in the way directories were treated. The *cpio*
 27916 utility did not treat directories differently from other files, and to select a directory and its
 27917 contents required that each file in the hierarchy be explicitly specified. For *tar*, a directory
 27918 matched every file in the file hierarchy it rooted.

27919 The *pax* utility offers both interfaces; by default, directories map into the file hierarchy they root.
 27920 The **-d** option causes *pax* to skip any file not explicitly referenced, as *cpio* historically did. The *tar*
 27921 **-style** behavior was chosen as the default because it was believed that this was the more
 27922 common usage and because *tar* is the more commonly available interface, as it was historically
 27923 provided on both System V and BSD implementations.

27924 The data interchange format specification in this volume of IEEE Std. 1003.1-200x requires that
 27925 processes with “appropriate privileges” shall always restore the ownership and permissions of
 27926 extracted files exactly as archived. If viewed from the historic equivalence between superuser
 27927 and “appropriate privileges”, there are two problems with this requirement. First, users running
 27928 as superusers may unknowingly set dangerous permissions on extracted files. Second, it is
 27929 needlessly limiting, in that superusers cannot extract files and own them as superuser unless the
 27930 archive was created by the superuser. (It should be noted that restoration of ownerships and
 27931 permissions for the superuser, by default, is historical practice in *cpio*, but not in *tar*.) In order to
 27932 avoid these two problems, the *pax* specification has an additional “privilege” mechanism, the **-p**
 27933 option. Only a *pax* invocation with the privileges needed, and which has the **-p** option set using
 27934 the **e** specification character, has the “appropriate privilege” to restore full ownership and
 27935 permission information.

27936 Note also that this volume of IEEE Std. 1003.1-200x requires that the file ownership and access
 27937 permissions shall be set, on extraction, in the same fashion as the *creat()* function when provided

- 27938 the mode stored in the archive. This means that the file creation mask of the user is applied to
27939 the file permissions.
- 27940 Users should note that directories may be created by *pax* while extracting files with permissions
27941 that are different from those that existed at the time the archive was created. When extracting
27942 sensitive information into a directory hierarchy that no longer exists, users are encouraged to set
27943 their file creation mask appropriately to protect these files during extraction.
- 27944 The table of contents output is written to standard output to facilitate pipeline processing.
- 27945 An early proposal had hard links displaying for all path names. This was removed because it
27946 complicates the output of the case where `-v` is not specified and does not match historical *cpio*
27947 usage. The hard-link information is available in the `-v` display.
- 27948 The archive formats inherited from the POSIX.1-1990 standard have certain restrictions that
27949 have been brought along from historical usage. For example, there are restrictions on the length
27950 of path names stored in the archive. When *pax* is used in `copy(-rw)` mode (copying directory
27951 hierarchies), the ability to use extensions from the `-xpax` format overcomes these restrictions.
- 27952 The default *blocksize* value of 5 120 bytes for *cpio* was selected because it is one of the standard
27953 block-size values for *cpio*, set when the `-B` option is specified. (The other default block-size value
27954 for *cpio* is 512 bytes, and this was considered to be too small.) The default block value of 10 240
27955 bytes for *tar* was selected because that is the standard block-size value for BSD *tar*. The
27956 maximum block size of 32 256 bytes (2^{15} -512 bytes) is the largest multiple of 512 bytes that fits
27957 into a signed 16-bit tape controller transfer register. There are known limitations in some
27958 historical systems that would prevent larger blocks from being accepted. Historical values were
27959 chosen to improve compatibility with historical scripts using *dd* or similar utilities to manipulate
27960 archives. Also, default block sizes for any file type other than character special file has been
27961 deleted from this volume of IEEE Std. 1003.1-200x as unimportant and not likely to affect the
27962 structure of the resulting archive.
- 27963 Implementations are permitted to modify the block-size value based on the archive format or
27964 the device to which the archive is being written. This is to provide implementations with the
27965 opportunity to take advantage of special types of devices, and it should not be used without a
27966 great deal of consideration as it almost certainly decreases archive portability.
- 27967 The intended use of the `-n` option was to permit extraction of one or more files from the archive
27968 without processing the entire archive. This was viewed by the standard developers as offering
27969 significant performance advantages over historical implementations. The `-n` option in early
27970 proposals had three effects; the first was to cause special characters in patterns to not be treated
27971 specially. The second was to cause only the first file that matched a pattern to be extracted. The
27972 third was to cause *pax* to write a diagnostic message to standard error when no file was found
27973 matching a specified pattern. Only the second behavior is retained by this volume of
27974 IEEE Std. 1003.1-200x, for many reasons. First, it is in general not acceptable for a single option to
27975 have multiple effects. Second, the ability to make pattern matching characters act as normal
27976 characters is useful for parts of *pax* other than file extraction. Third, a finer degree of control over
27977 the special characters is useful because users may wish to normalize only a single special
27978 character in a single file name. Fourth, given a more general escape mechanism, the previous
27979 behavior of the `-n` option can be easily obtained using the `-s` option or a *sed* script. Finally,
27980 writing a diagnostic message when a pattern specified by the user is unmatched by any file is
27981 useful behavior in all cases.
- 27982 In this version, the `-n` was removed from the `copy` mode synopsis of *pax*; it is inapplicable
27983 because there are no pattern operands specified in this mode.
- 27984 There is another method than *pax* for copying subtrees in IEEE Std. 1003.1-200x described as part
27985 of the *cp* utility. Both methods are historical practice: *cp* provides a simpler, more intuitive

27986 interface, while *pax* offers a finer granularity of control. Each provides additional functionality to
27987 the other; in particular, *pax* maintains the hard-link structure of the hierarchy while *cp* does not.
27988 It is the intention of the standard developers that the results be similar (using appropriate option
27989 combinations in both utilities). The results are not required to be identical; there seemed
27990 insufficient gain to applications to balance the difficulty of implementations having to guarantee
27991 that the results would be exactly identical.

27992 A single archive may span more than one file. It is suggested that implementations provide
27993 informative messages to the user on standard error whenever the archive file is changed.

27994 The **-d** option (do not create intermediate directories not listed in the archive) found in early
27995 proposals was originally provided as a complement to the historic **-d** option of *cpio*. It has been
27996 deleted.

27997 The **-s** option in early proposals specified a subset of the substitution command from the *ed*
27998 utility. As there was no reason for only a subset to be supported, the **-s** option is now
27999 compatible with the current *ed* specification. Since the delimiter can be any non-null character,
28000 the following usage with single spaces is valid:

```
28001 pax -s " foo bar " ...
```

28002 The **-t** option (specify an implementation-defined identifier naming an input or output device)
28003 found in early proposals has been deleted because it is not historical practice and is of limited
28004 utility. In particular, historic versions of neither *cpio* nor *tar* had the concept of devices that were
28005 not mapped into the file system; if the devices are mapped into the file system, the **-f** option is
28006 sufficient.

28007 The default behavior of *pax* with regard to file modification times is the same as historical
28008 implementations of *tar*. It is not the historical behavior of *cpio*.

28009 Because the **-i** option uses **/dev/tty**, utilities without a controlling terminal are not able to use
28010 this option.

28011 The **-y** option, found in early proposals, has been deleted because a line containing a single
28012 period for the **-i** option has equivalent functionality. The special lines for the **-i** option (a single
28013 period and the empty line) are historical practice in *cpio*.

28014 In early drafts, an **-echarmap** option was included to increase portability of files between systems
28015 using different coded character sets. This option was omitted because it was apparent that
28016 consensus could not be formed for it. In this version, the use of UTF-8 should be an adequate
28017 substitute.

28018 The **-k** option was added to address international concerns about the dangers involved in the
28019 character set transformations of **-e** (if the target character set were different than the source, the
28020 file names might be transformed into names matching existing files) and also was made more
28021 general to protect files transferred between file systems with different {NAME_MAX} values
28022 (truncating a file name on a smaller system might also inadvertently overwrite existing files). As
28023 stated, it prevents any overwriting, even if the target file is older than the source. This version
28024 adds more granularity of options to solve this problem by introducing the **-oinvalid=** option—
28025 specifically the UTF-8 action. (Note that an existing file that is named with a UTF-8 encoding is
28026 still subject to overwriting in this case. The **-k** option closes that loophole.)

28027 Some of the file characteristics referenced in this volume of IEEE Std. 1003.1-200x might not be
28028 supported by some archive formats. For example, neither the *tar* nor *cpio* formats contain the file
28029 access time. For this reason, the **e** specification character has been provided, intended to cause all
28030 file characteristics specified in the archive to be retained.

28031 It is required that extracted directories, by default, have their access and modification times and
 28032 permissions set to the values specified in the archive. This has obvious problems in that the
 28033 directories are almost certainly modified after being extracted and that directory permissions
 28034 may not permit file creation. One possible solution is to create directories with the mode
 28035 specified in the archive, as modified by the *umask* of the user, with sufficient permissions to
 28036 allow file creation. After all files have been extracted, *pax* would then reset the access and
 28037 modification times and permissions as necessary.

28038 The list-mode formatting description borrows heavily from the one defined by the *printf* utility.
 28039 However, since there is no separate operand list to get conversion arguments, the format was
 28040 extended to allow specifying the name of the conversion argument as part of the conversion
 28041 specification.

28042 The **T** specifier allows time fields to be displayed in any of the date formats. Unlike the *ls* utility,
 28043 *pax* does not adjust the format when the date is less than six months in the past. This makes
 28044 parsing the output more predictable.

28045 The **D** specifier handles the ability to display the major/minor or file size, as with *ls*, by using
 28046 **%-8(size)D**.

28047 The **L** specifier handles the *ls* display for symbolic links.

28048 Conversion specifiers were added to generate existing known types used for *ls*.

28049 **pax Interchange Format**

28050 The new POSIX data interchange format was developed primarily to satisfy international
 28051 concerns that the **ustar** and *cpio* formats did not provide for file, user, and group names encoded
 28052 in characters outside a subset of the ISO/IEC 646:1991 standard. The standard developers
 28053 realized that this new POSIX data interchange format should be very extensible because there
 28054 were other requirements they foresaw in the near future:

- 28055 • Support international character encodings and locale information
- 28056 • Support security information (ACLs, and so on)
- 28057 • Support future file types, such as realtime or contiguous files
- 28058 • Include data areas for implementation use
- 28059 • Support systems with words larger than 32 bits and timers with subsecond granularity

28060 The following were not goals for this format because these are better handled by separate
 28061 utilities or are inappropriate for a portable format:

- 28062 • Encryption
- 28063 • Compression
- 28064 • Data translation between locales and codesets
- 28065 • *inode* storage

28066 The format chosen to support the goals is an extension of the **ustar** format. Of the two formats
 28067 previously available, only the **ustar** format was selected for extensions because:

- 28068 • It was easier to extend in an upward-compatible way. It offered version flags and header
 28069 block type fields with room for future standardization. The *cpio* format, while possessing a
 28070 more flexible file naming methodology, could not be extended without breaking some
 28071 theoretical implementation or using a dummy file name that could be a legitimate file name.

28072 • Industry experience since the original “tar wars” fought in developing the ISO POSIX-1
28073 standard has clearly been in favor of the **ustar** format, which is generally the default output
28074 format selected for *pax* implementations on new systems.

28075 The new format was designed with one additional goal in mind: reasonable behavior when an
28076 older *tar* or *pax* utility happened to read an archive. Since the POSIX.1-1990 standard mandated
28077 that a “format-reading utility” had to treat unrecognized *typeflag* values as regular files, this
28078 allowed the format to include all the extended information in a pseudo-regular file that preceded
28079 each real file. An option is given that allows the archive creator to set up reasonable names for
28080 these files on the older systems. Also, the normative text suggests that reasonable file access
28081 values be used for this **ustar** header block. Making these header files inaccessible for convenient
28082 reading and deleting would not be reasonable. File permissions of 600 or 700 are suggested.

28083 The **ustar** *typeflag* field was used to accommodate the additional functionality of the new format
28084 rather than magic or version because the POSIX.1-1990 standard (and, by reference, the previous
28085 version of *pax*), mandated the behavior of the format-reading utility when it encountered an
28086 unknown *typeflag*, but was silent about the other two fields.

28087 Early proposals of the first revision to IEEE Std. 1003.1-200x contained a proposed archive
28088 format that was based on compatibility with the standard for tape files (ISO 1001, similar to the
28089 format used historically on many mainframes and minicomputers). This format was overly
28090 complex and required considerable overhead in volume and header records. Furthermore, the
28091 standard developers felt that it would not be acceptable to the community of POSIX developers,
28092 so it was later changed to be a format more closely related to historical practice on POSIX
28093 systems.

28094 The prefix and name split of path names in **ustar** was replaced by the single path extended
28095 header record for simplicity.

28096 The concept of a global extended header (*typeflag g*) was controversial. If this were applied to an
28097 archive being recorded on magnetic tape, a few unreadable blocks at the beginning of the tape
28098 could be a serious problem; a utility attempting to extract as many files as possible from a
28099 damaged archive could lose a large percentage of file header information in this case. However,
28100 if the archive were on a reliable medium, such as a CD-ROM, the global extended header offers
28101 considerable potential size reductions by eliminating redundant information. Thus, the text
28102 warns against using the global method for unreliable media and provides a method for
28103 implanting global information in the extended header for each file, rather than in the *typeflag g*
28104 records.

28105 No facility for data translation or filtering on a per-file basis is included because the standard
28106 developers could not invent an interface that would allow this in an efficient manner. If a filter,
28107 such as encryption or compression, is to be applied to all the files, it is more efficient to apply the
28108 filter to the entire archive as a single file. The standard developers considered interfaces that
28109 would invoke a shell script for each file going into or out of the archive, but the system overhead
28110 in this approach was considered to be too high.

28111 One such approach would be to have **filter=** records that give a path name for an executable.
28112 When the program is invoked, the file and archive would be open for standard input/output
28113 and all the header fields would be available as environment variables or command-line
28114 arguments. The standard developers did discuss such schemes, but they were omitted from
28115 IEEE Std. 1003.1-200x due to concerns about excessive overhead. Also, the program itself would
28116 need to be in the archive if it were to be used portably.

28117 There is currently no portable means of identifying the character set(s) used for a file in the file
28118 system. Therefore, *pax* has not been given a mechanism to generate charset records
28119 automatically. The only portable means of doing this is for the user to write the archive using the

28120 –**charset=string** command line option. This assumes that all of the files in the archive use the
 28121 same encoding. The “implementation-defined” text is included to allow for a system that can
 28122 identify the encodings used for each of its files.

28123 The table of standards that accompanies the charset record description is acknowledged to be
 28124 very limited. Only a limited number of character set standards is reasonable for maximal
 28125 interchange. Any character set is, of course, possible by prior agreement. It was suggested that
 28126 EBCDIC be listed, but it was omitted because it is not defined by a formal standard. Formal
 28127 standards, and then only those with reasonably large followings, can be included here, simply as
 28128 a matter of practicality. The <value>s represent names of officially registered character sets in the
 28129 format required by the ISO 2375:1985 standard.

28130 The normal comma or <blank>-separated list rules are not followed in the case of keyword
 28131 options to allow ease of argument parsing for *getopts*.

28132 Further information on character encodings is in **pax Archive Character Set Encoding/Decoding**
 28133 (on page 2941).

28134 The standard developers have reserved keyword name space for vendor extensions. It is
 28135 suggested that the format to be used is:

28136 *VENDOR.keyword*

28137 where *VENDOR* is the name of the vendor or organization in all uppercase letters. It is further
 28138 suggested that the keyword following the period be named differently than any of the standard
 28139 keywords so that it could be used for future standardization, if appropriate, by omitting the
 28140 *VENDOR* prefix.

28141 The <length> field in the extended header record was included to make it simpler to step
 28142 through the records, even if a record contains an unknown format (to a particular *pax*) with
 28143 complex interactions of special characters. It also provides a minor integrity checkpoint within
 28144 the records to aid a program attempting to recover files from a damaged archive.

28145 There are no extended header versions of the *devmajor* and *devminor* fields because the
 28146 unspecified format **ustar** header field should be sufficient. If they are not, vendor-specific
 28147 extended keywords (such as *VENDOR.devmajor*) should be used.

28148 Device and *i*-number labeling of files was not adopted from *cpio*; files are interchanged strictly
 28149 on a symbolic name basis, as in **ustar**.

28150 Just as with the **ustar** format descriptions, the new format makes no special arrangements for
 28151 multi-volume archives. Each of the *pax* archive types is assumed to be inside a single POSIX file
 28152 and splitting that file over multiple volumes (diskettes, tape cartridges, and so on), processing
 28153 their labels, and mounting each in the proper sequence are considered to be implementation
 28154 details that cannot be described portably.

28155 The *pax* format is intended for interchange, not only for backup on a single (family of) systems. It
 28156 is not as densely packed as might be possible for backup:

28157

- It contains information as coded characters that could be coded in binary.
- It identifies extended records with name fields that could be omitted in favor of a fixed-field layout.
- It translates names into a portable character set and identifies locale-related information, both of which are probably unnecessary for backup.

28162 The requirements on restoring from an archive are slightly different from the historical wording,
 28163 allowing for non-monolithic privilege to bring forward as much as possible. In particular,
 28164 attributes such as “high performance file” might be broadly but not universally granted while

28165 set-user-ID or *chown()* might be much more restricted. There is no implication in
28166 IEEE Std. 1003.1-200x that the security information be honored after it is restored to the file
28167 hierarchy, in spite of what might be improperly inferred by the silence on that topic. That is a
28168 topic for another standard.

28169 Links are recorded in the fashion described here because a link can be to any file type. It is
28170 desirable in general to be able to restore part of an archive selectively and restore all of those
28171 files completely. If the data is not associated with each link, it is not possible to do this.
28172 However, the data associated with a file can be large, and when selective restoration is not
28173 needed, this can be a significant burden. The archive is structured so that files that have no
28174 associated data can always be restored by the name of any link name of any link, and the user
28175 may choose whether data is recorded with each instance of a file that contains data. The format
28176 permits mixing of both types of links in a single archive; this can be done for special needs, and
28177 *pax* is expected to interpret such archives on input properly, despite the fact that there is no *pax*
28178 option that would force this mixed case on output. (When **-o linkdata** is used, the output must
28179 contain the duplicate data, but the implementation is free to include it or omit it when **-o**
28180 **linkdata** is not used.)

28181 The time values are included as extended header records for those implementations needing
28182 more than the eleven octal digits allowed by the **ustar** format. Portable file timestamps cannot be
28183 negative. If *pax* encounters a file with a negative timestamp in **copy** or **write** mode, it can reject
28184 the file, substitute a non-negative timestamp, or generate a non-portable timestamp with a
28185 leading ' - '. Even though some implementations can support finer file-time granularities than
28186 seconds, the normative text requires support only for seconds since the Epoch because the
28187 ISO POSIX-1 standard states them that way. The **ustar** format includes only *mtime*; the new
28188 format adds *atime* and *ctime* for symmetry. The *atime* access time restored to the file system will
28189 be affected by the **-p a** and **-p e** options. The *ctime* creation time (actually *inode* modification
28190 time) is described with “appropriate privilege” so that it can be ignored when writing to the file
28191 system. POSIX does not provide a portable means to change file creation time. Nothing is
28192 intended to prevent a non-portable implementation of *pax* from restoring the value.

28193 The *gid*, *size*, and *uid* extended header records were included to allow expansion beyond the
28194 sizes specified in the regular *tar* header. New file system architectures are emerging that will
28195 exhaust the 12-digit size field. There are probably not many systems requiring more than 8 digits
28196 for user and group IDs, but the extended header values were included for completeness,
28197 allowing overrides for all of the decimal values in the *tar* header.

28198 The standard developers intended to describe the effective results of *pax* with regard to file
28199 ownerships and permissions; implementations are not restricted in timing or sequencing the
28200 restoration of such, provided the results are as specified.

28201 Much of the text describing the extended headers refers to use in “**write** or **copy** modes”. The
28202 **copy** mode references are due to the normative text: “The effect of the copy shall be as if the
28203 copied files were written to an archive file and then subsequently extracted ...”. There is
28204 certainly no way to test whether *pax* is actually generating the extended headers in **copy** mode,
28205 but the effects must be as if it had.

28206 pax Archive Character Set Encoding/Decoding

28207 There is a need to exchange archives of files between systems of different native codesets. File
28208 names, group names, and user names must be preserved to the fullest extent possible when an
28209 archive is read on the receiving platform. Translation of the contents of files is not within the
28210 scope of the *pax* utility.

28211 There will also be the need to represent glyphs that are not available on the receiving platform.
28212 (A *glyph* is commonly called a character, but without any reference to a specific encoding of that
28213 character. The term *glyph* refers to the symbol itself.) These unsupported glyphs cannot be
28214 automatically folded to the local set of glyphs due to the chance of collisions. This could result in
28215 overwriting previous extracted files from the archive or pre-existing files on the system.

28216 For these reasons, the codeset used to represent glyphs within the extended header records of
28217 the *pax* archive must be sufficiently rich to handle all commonly used character sets. The fields
28218 requiring translation include, at a minimum, file names, user names, group names, and link path
28219 names. Implementations may wish to have localized extended keywords that use non-portable
28220 characters.

28221 The standard developers considered the following options:

- 28222 • The archive creator specifies the well-defined name of the source codeset. The receiver must
28223 then recognize the codeset name and perform the appropriate translations to the destination
28224 codeset.
- 28225 • The archive creator includes within the archive the character mapping table for the source
28226 codeset used to encode extended header records. The receiver must then read the character
28227 mapping table and perform the appropriate translations to the destination codeset.
- 28228 • The archive creator translates the extended header records in the source codeset into a
28229 canonical form. The receiver must then perform the appropriate translations to the
28230 destination codeset.

28231 The approach that incorporates the name of the source codeset poses the problem of codeset
28232 name registration, and makes the archive useless to *pax* archive decoders that do not recognize
28233 that codeset.

28234 Because parts of an archive may be corrupted, the standard developers felt that including the
28235 character map of the source codeset was too fragile. The loss of this one key component could
28236 result in making the entire archive useless. (The difference between this and the global extended
28237 header decision was that the latter has a workaround—duplicating extended header records on
28238 unreliable media—but this would be too burdensome for large character set maps.)

28239 Both of the above approaches also put an undue burden on the *pax* archive receiver to handle the
28240 cross-product of all source and destination codesets.

28241 To simplify the translation from the source codeset to the canonical form and from the canonical
28242 form to the destination codeset, the standard developers decided that the internal representation
28243 should be a stateless encoding. A stateless encoding is one where each codepoint has the same
28244 meaning, without regard to the decoder being in a specific state. An example of a stateful
28245 encoding would be the Japanese Shift-JIS; an example of a stateless encoding would be the
28246 ISO/IEC 646:1991 standard (equivalent to 7-bit ASCII).

28247 For these reasons, the standard developers decided to adopt a canonical format for the
28248 representation of file information strings. The obvious, well-endorsed candidate is the
28249 ISO/IEC 10646-1:1993 standard (based in part on Unicode), which can be used to represent the
28250 glyphs of virtually all standardized character sets. The standard developers initially agreed upon
28251 using UCS2 (16-bit Unicode) as the internal representation. This repertoire of glyphs provides a

28252 sufficiently rich set to represent all commonly-used codesets.

28253 However, the standard developers found that the 16-bit Unicode representation had some
 28254 problems. It forced the issue of standardizing byte ordering. The 2-byte length of each character
 28255 made the extended header records twice as long for the case of strings coded entirely from
 28256 historical 7-bit ASCII. For these reasons, the standard developers chose the UTF-8 defined in the
 28257 ISO/IEC 10646-1:1993 standard. This multi-byte representation encodes UCS2 or UCS4
 28258 characters reliably and deterministically, eliminating the need for a canonical byte ordering. In
 28259 addition, NUL octets and other characters possibly confusing to POSIX file systems do not
 28260 appear, except to represent themselves. It was realized that certain national codesets take up
 28261 more space after the encoding, due to their placement within the UCS range; it was felt that the
 28262 usefulness of the encoding of the names outweighs the disadvantage of size increase for file,
 28263 user, and group names.

28264 The encoding of UTF-8 is as follows:

28265	UCS4 Hex Encoding	UTF-8 Binary Encoding
28266	00000000-0000007F	0xxxxxxx
28267	00000080-000007FF	110xxxxx 10xxxxxx
28268	00000800-0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
28269	00010000-001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
28270	00200000-03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
28271	04000000-7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

28272 where each 'x' represents a bit value from the character being translated.

28273 **ustar Interchange Format**

28274 The description of the **ustar** format reflects numerous enhancements over pre-1988 versions of
 28275 the historical *tar* utility. The goal of these changes was not only to provide the functional
 28276 enhancements desired, but also to retain compatibility between new and old versions. This
 28277 compatibility has been retained. Archives written using the old archive format are compatible
 28278 with the new format.

28279 Implementors should be aware that the previous file format did not include a mechanism to
 28280 archive directory type files. For this reason, the convention of using a file name ending with
 28281 slash was adopted to specify a directory on the archive.

28282 The total size of the *name* and *prefix* fields have been set to meet the minimum requirements for
 28283 {PATH_MAX}. If a path name will fit within the *name* field, it is recommended that the path
 28284 name be stored there without the use of the *prefix* field. Although the name field is known to be
 28285 too small to contain {PATH_MAX} characters, the value was not changed in this version of the
 28286 archive file format to retain backward compatibility, and instead the prefix was introduced.
 28287 Also, because of the earlier version of the format, there is no way to remove the restriction on the
 28288 *linkname* field being limited in size to just that of the *name* field.

28289 The *size* field is required to be meaningful in all implementation extensions, although it could be
 28290 zero. This is required so that the data blocks can always be properly counted.

28291 It is suggested that if device special files need to be represented that cannot be represented in the
 28292 standard format that one of the extension types (A-Z) be used, and that the additional
 28293 information for the special file be represented as data and be reflected in the *size* field.

28294 Attempting to restore a special file type, where it is converted to ordinary data and conflicts
 28295 with an existing file name, need not be specially detected by the utility. If run as an ordinary
 28296 user, *pax* should not be able to overwrite the entries in, for example, */dev* in any case (whether
 28297 the file is converted to another type or not). If run as a privileged user, it should be able to do so,

28298 and it would be considered a bug if it did not. The same is true of ordinary data files and
28299 similarly named special files; it is impossible to anticipate the needs of the user (who could
28300 really intend to overwrite the file), so the behavior should be predictable (and thus regular) and
28301 rely on the protection system as required.

28302 The value 7 in the *typeflag* field is intended to define how contiguous files can be stored in a
28303 **ustar** archive. IEEE Std. 1003.1-200x does not require the contiguous file extension, but does
28304 define a standard way of archiving such files so that all conforming systems can interpret these
28305 file types in a meaningful and consistent manner. On a system that does not support extended
28306 file types, the *pax* utility should do the best it can with the file and go on to the next.

28307 The file protection modes are those conventionally used by the *ls* utility. This is extended
28308 beyond the usage in the ISO POSIX-2 standard to support the “shared text” or “sticky” bit. It is
28309 intended that the conformance document should not document anything beyond the existence
28310 of and support of such a mode. Further extensions are expected to these bits, particularly with
28311 overloading the set-user-ID and set-group-ID flags.

28312 **cpio Interchange Format**

28313 The reference to appropriate privilege in the *cpio* format refers to an error on standard output;
28314 the **ustar** format does not make comparable statements.

28315 The model for this format was the historical System V *cpio-c* data interchange format. This
28316 model documents the portable version of the *cpio* format and not the binary version. It has the
28317 flexibility to transfer data of any type described within IEEE Std. 1003.1-200x, yet is extensible to
28318 transfer data types specific to extensions beyond IEEE Std. 1003.1-200x (for example, contiguous
28319 files). Because it describes existing practice, there is no question of maintaining upward
28320 compatibility.

28321 **cpio Header**

28322 There has been some concern that the size of the *c_ino* field of the header is too small to handle
28323 those systems that have very large *inode* numbers. However, the *c_ino* field in the header is used
28324 strictly as a hard-link resolution mechanism for archives. It is not necessarily the same value as
28325 the *inode* number of the file in the location from which that file is extracted.

28326 The name *c_magic* is based on historical usage.

28327 **cpio File Name**

28328 For most historical implementations of the *cpio* utility, {PATH_MAX} octets can be used to
28329 describe the path name without the addition of any other header fields (the NUL character
28330 would be included in this count). {PATH_MAX} is the minimum value for path name size,
28331 documented as 256 bytes. However, an implementation may use *c_namesize* to determine the
28332 exact length of the path name. With the current description of the <**cpio.h**> header, this path
28333 name size can be as large as a number that is described in six octal digits.

28334 Two values are documented under the *c_mode* field values to provide for extensibility for known
28335 file types:

28336 **Notes to Reviewers**28337 *This section with side shading will not appear in the final copy. - Ed.*28338 Note that the sockets extension below needs to be integrated, now that sockets have been
28339 merged28340 **0110 000** Reserved for contiguous files. The implementation may treat the rest of the
28341 information for this archive like a regular file. If this file type is undefined, the
28342 implementation may create the file as a regular file.28343 **0140 000** Reserved for sockets. If this type is undefined on the target system, the
28344 implementation may decide to ignore this file type and output a warning message.28345 This provides for extensibility of the *cpio* format while allowing for the ability to read old
28346 archives. Files of an unknown type may be read as “regular files” on some implementations. On
28347 a system that does not support extended file types, the *pax* utility should do the best it can with
28348 the file and go on to the next.28349 **FUTURE DIRECTIONS**

28350 None.

28351 **SEE ALSO**28352 *cp*, *ed*, *getopts*, *printf*, the Base Definitions volume of IEEE Std. 1003.1-200x, **<cpio.h>**, the System
28353 Interfaces volume of IEEE Std. 1003.1-200x, *chown()*, *creat()*, *mkdir()*, *stat()*, *write()*28354 **CHANGE HISTORY**

28355 First released in Issue 4.

28356 **Issue 5**28357 A note is added to the APPLICATION USAGE indicating that the *cpio* and *tar* formats can only
28358 support files up to 8 gigabytes in size.28359 **Issue 6**28360 The *pax* utility is aligned with the IEEE P1003.2b draft standard:

- 28361
- Support has been added for symbolic links in the options and interchange formats.
 - A new format has been devised, based on extensions to ustar.
 - References to the “extended” *tar* and *cpio* formats derived from the POSIX.1-1990 standard have been changed to remove the “extended” adjective because this could cause confusion with the extended *tar* header added in this revision. (All references to *tar* are actually to **ustar**).

28367 IEEE PASC Interpretation 1003.2 #168 is applied clarifying that *mkdir()* and *mkfifo()* calls can
28368 ignore an [EEXIST] error when extracting an archive.

28369 NAME

28370 pr — print files

28371 SYNOPSIS

```
28372 pr [+page][-column][-adFmrt][-e[char][gap]][-h header][-i[char][gap]]
28373 xSI [-l lines][-n[char][width]][-o offset][-s[char]][-w width][-fp]
28374 [file...]
```

28375 DESCRIPTION

28376 The *pr* utility is a printing and pagination filter. If multiple input files are specified, each shall be
 28377 read, formatted, and written to standard output. By default, the input shall be separated into 66-
 28378 line pages, each with:

- 28379 • A 5-line header that includes the page number, date, time, and the path name of the file
- 28380 • A 5-line trailer consisting of blank lines

28381 If standard output is associated with a terminal, diagnostic messages shall be deferred until the
 28382 *pr* utility has completed processing.

28383 When options specifying multi-column output are specified, output text columns shall be of
 28384 equal width; input lines that do not fit into a text column shall be truncated. By default, text
 28385 columns shall be separated with at least one <blank> character.

28386 OPTIONS

28387 The *pr* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
 28388 Utility Syntax Guidelines, except that: the *page* option has a '+' delimiter; *page* and *column* can
 28389 be multi-digit numbers; some of the option-arguments are optional; and some of the option-
 28390 arguments cannot be specified as separate arguments from the preceding option letter. In
 28391 particular, the *-s* option does not allow the option letter to be separated from its argument, and
 28392 the options *-e*, *-i*, and *-n* require that both arguments, if present, not be separated from the
 28393 option letter.

28394 The following options shall be supported. In the following option descriptions, *column*, *lines*,
 28395 *offset*, *page*, and *width* are positive decimal integers; *gap* is a non-negative decimal integer.

- 28396 *+page* Begin output at page number *page* of the formatted input.
- 28397 *-column* Produce multi-column output that is arranged in *column* columns (the default shall
 28398 be 1) and is written down each column in the order in which the text is received
 28399 from the input file. This option should not be used with *-m*. The options *-e* and *-i*
 28400 shall be assumed for multiple text-column output. Whether or not text columns
 28401 are produced with identical vertical lengths is unspecified, but a text column shall
 28402 never exceed the length of the page (see the *-l* option). When used with *-t*, use the
 28403 minimum number of lines to write the output.
- 28404 *-a* Modify the effect of the *-column* option so that the columns are filled across the
 28405 page in a round-robin order (for example, when *column* is 2, the first input line
 28406 heads column 1, the second heads column 2, the third is the second line in column
 28407 1, and so on).
- 28408 *-d* Produce output that is double-spaced; append an extra <newline> character
 28409 following every <newline> character found in the input.
- 28410 *-e[*char*][*gap*]*
 28411 Expand each input <tab> character to the next greater column position specified
 28412 by the formula $n * gap + 1$, where *n* is an integer > 0. If *gap* is zero or is omitted, it
 28413 shall default to 8. All <tab> characters in the input shall be expanded into the
 28414 appropriate number of <space> characters. If any non-digit character, *char*, is

- 28415 specified, it shall be used as the input <tab> character.
- 28416 XSI **-f** Use a <form-feed> character for new pages, instead of the default behavior that
28417 uses a sequence of <newline> characters. Pause before beginning the first page if
28418 the standard output is associated with a terminal.
- 28419 **-F** Use a <form-feed> character for new pages, instead of the default behavior that
28420 uses a sequence of <newline> characters.
- 28421 **-h header** Use the string *header* to replace the contents of the *file* operand in the page header.
- 28422 **-i[char][gap]**
28423 In output, replace multiple <space> characters with <tab> characters wherever
28424 two or more adjacent <space> characters reach column positions *gap*+1, 2* *gap*+1,
28425 3* *gap*+1, and so on. If *gap* is zero or is omitted, default tab settings at every eighth
28426 column position shall be assumed. If any non-digit character, *char*, is specified, it
28427 shall be used as the output <tab> character.
- 28428 **-l lines** Override the 66-line default and reset the page length to *lines*. If *lines* is not greater
28429 than the sum of both the header and trailer depths (in lines), the *pr* utility shall
28430 suppress both the header and trailer, as if the **-t** option were in effect.
- 28431 **-m** Merge files. Standard output shall be formatted so the *pr* utility writes one line
28432 from each file specified by a *file* operand, side by side into text columns of equal
28433 fixed widths, in terms of the number of column positions. Implementations shall
28434 support merging of at least nine *file* operands.
- 28435 **-n[char][width]**
28436 Provide *width*-digit line numbering (default for *width* shall be 5). The number shall
28437 occupy the first *width* column positions of each text column of default output or
28438 each line of **-m** output. If *char* (any non-digit character) is given, it shall be
28439 appended to the line number to separate it from whatever follows (default for *char*
28440 is a <tab> character).
- 28441 **-o offset** Each line of output shall be preceded by offset <space>*s*. If the **-o** option is not
28442 specified, the default offset shall be zero. The space taken is in addition to the
28443 output line width (see the **-w** option below).
- 28444 **-p** Pause before beginning each page if the standard output is directed to a terminal
28445 (*pr* shall write an <alert> character to standard error and wait for a <carriage-
28446 return> character to be read on */dev/tty*).
- 28447 **-r** Write no diagnostic reports on failure to open files.
- 28448 **-s[char]** Separate text columns by the single character *char* instead of by the appropriate
28449 number of <space> characters (default for *char* shall be the <tab> character).
- 28450 **-t** Write neither the five-line identifying header nor the five-line trailer usually
28451 supplied for each page. Quit writing after the last line of each file without spacing
28452 to the end of the page.
- 28453 **-w width** Set the width of the line to *width* column positions for multiple text-column output
28454 only. If the **-w** option is not specified and the **-s** option is not specified, the default
28455 width shall be 72. If the **-w** option is not specified and the **-s** option is specified,
28456 the default width shall be 512.
- 28457 For single column output, input lines shall not be truncated.

28458 **OPERANDS**

28459 The following operand shall be supported:

28460 *file* A path name of a file to be written. If no *file* operands are specified, or if a *file*
 28461 operand is '-', the standard input shall be used.

28462 **STDIN**

28463 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
 28464 See the INPUT FILES section.

28465 **INPUT FILES**

28466 The input files shall be text files.

28467 The file `/dev/tty` is used to read responses required by the `-p` option.

28468 **ENVIRONMENT VARIABLES**

28469 The following environment variables shall affect the execution of *pr*:

28470 *LANG* Provide a default value for the internationalization variables that are unset or null.
 28471 If *LANG* is unset or null, the corresponding value from the implementation-
 28472 defined default locale shall be used. If any of the internationalization variables
 28473 contains an invalid setting, the utility shall behave as if none of the variables had
 28474 been defined.

28475 *LC_ALL* If set to a non-empty string value, override the values of all the other
 28476 internationalization variables.

28477 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 28478 characters (for example, single-byte as opposed to multi-byte characters in
 28479 arguments and input files) and which characters are defined as printable (character
 28480 class **print**). Non-printable characters are still written to standard output, but are
 28481 not counted for the purpose for column-width and line-length calculations.

28482 *LC_MESSAGES*

28483 Determine the locale that should be used to affect the format and contents of
 28484 diagnostic messages written to standard error.

28485 *LC_TIME* Determine the format of the date and time for use in writing header lines.

28486 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

28487 *TZ* Determine the timezone for use in writing header lines.

28488 **ASYNCHRONOUS EVENTS**

28489 If *pr* receives an interrupt while writing to a terminal, it shall flush all accumulated error
 28490 messages to the screen before terminating.

28491 **STDOUT**

28492 The *pr* utility output shall be a paginated version of the original file (or files). This pagination
 28493 shall be accomplished using either `<form-feed>` characters or a sequence of `<newline>`
 28494 XSI characters, as controlled by the `-F` or `-f` option. Page headers shall be generated unless the `-t`
 28495 option is specified. The page headers shall be of the form:

28496 `"\n\n%s %s Page %d\n\n\n", <output of date>, <file>, <page number>`

28497 In the POSIX locale, the `<output of date>` field, representing the date and time of last modification
 28498 of the input file (or the current date and time if the input file is standard input), shall be
 28499 equivalent to the output of the following command as it would appear if executed at the given
 28500 time:

- 28501 date "+%b %e %H:%M %Y"
- 28502 without the trailing <newline> character, if the page being written is from standard input. If the
28503 page being written is not from standard input, in the POSIX locale, the same format shall be
28504 used, but the time used shall be the modification time of the file corresponding to *file* instead of
28505 the current time. When the *LC_TIME* locale category is not set to the POSIX locale, a different
28506 format and order of presentation of this field may be used.
- 28507 If the standard input is used instead of a *file* operand, the <*file*> field shall be replaced by a null
28508 string.
- 28509 If the **-h** option is specified, the <*file*> field shall be replaced by the *header* argument.
- 28510 **STDERR**
- 28511 Used for diagnostic messages and for alerting the terminal when **-p** is specified.
- 28512 **OUTPUT FILES**
- 28513 None.
- 28514 **EXTENDED DESCRIPTION**
- 28515 None.
- 28516 **EXIT STATUS**
- 28517 The following exit values shall be returned:
- 28518 0 Successful completion.
- 28519 >0 An error occurred.
- 28520 **CONSEQUENCES OF ERRORS**
- 28521 Default.
- 28522 **APPLICATION USAGE**
- 28523 None.
- 28524 **EXAMPLES**
- 28525 1. Print a numbered list of all files in the current directory:
- 28526 ls -a | pr -n -h "Files in \$(pwd)."
- 28527 2. Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":
- 28528 pr -3d -h "file list" file1 file2
- 28529 3. Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, ...:
- 28530 pr -e9 -t <file1 >file2
- 28531 **RATIONALE**
- 28532 This utility is one of those that does not follow the Utility Syntax Guidelines because of its
28533 historical origins. The standard developers could have added new options that obeyed the
28534 guidelines (and marked the old options *obsolescent*) or devised an entirely new utility; there are
28535 examples of both actions in this volume of IEEE Std. 1003.1-200x. Because of its widespread use
28536 by historical applications, the standard developers decided to exempt this version of *pr* from
28537 many of the guidelines.
- 28538 Implementations are required to accept option-arguments to the **-h**, **-l**, **-o**, and **-w** options
28539 whether presented as part of the same argument or as a separate argument to *pr*, as suggested by
28540 the Utility Syntax Guidelines. The **-n** and **-s** options, however, are specified as in historical
28541 practice because they are frequently specified without their optional arguments. If a <blank>
28542 were allowed before the option-argument in these cases, a *file* operand could mistakenly be

- 28543 interpreted as an option-argument in historical applications.
- 28544 The text about the minimum number of lines in multi-column output was included to ensure
28545 that a best effort is made in balancing the length of the columns. There are known historical
28546 implementations in which, for example, 60-line files are listed by *pr -2* as one column of 56 lines
28547 and a second of 4. Although this is not a problem when a full page with headers and trailers is
28548 produced, it would be relatively useless when used with *-t*.
- 28549 Historical implementations of the *pr* utility have differed in the action taken for the *-f* option.
28550 BSD uses it as described here for the *-F* option; System V uses it to change trailing *<newline>s*
28551 on each page to a *<form-feed>* and, if standard output is a TTY device, sends an *<alert>*
28552 to standard error and reads a line from */dev/tty* before the first page. There were strong arguments
28553 from both sides of this issue concerning historical practice and additional arguments against the
28554 System V *-f* behavior, on the grounds that having the behavior of an option change depending
28555 on where output is directed was not a modular design. Therefore, the *-f* option is not specified
28556 and the *-F* option has been added.
- 28557 The *<output of date>* field in the *-I* format is specified only for the POSIX locale. As noted, the
28558 format can be different in other locales. No mechanism for defining this is present in this volume
28559 of IEEE Std. 1003.1-200x, as the appropriate vehicle is a message catalog; that is, the format
28560 should be specified as a “message”.
- 28561 **FUTURE DIRECTIONS**
- 28562 It is possible that a new interface that conforms to the Utility Syntax Guidelines will be
28563 introduced.
- 28564 **SEE ALSO**
- 28565 *expand, lp*
- 28566 **CHANGE HISTORY**
- 28567 First released in Issue 2.
- 28568 **Issue 4**
- 28569 Aligned with the ISO/IEC 9945-2:1993 standard.
- 28570 **Issue 6**
- 28571 The following new requirements on POSIX implementations derive from alignment with the
28572 Single UNIX Specification:
- 28573
 - The *-p* option is added.
- 28574 The normative text is reworded to avoid use of the term “must” for application requirements.

28575 **NAME**

28576 printf — write formatted output

28577 **SYNOPSIS**28578 printf *format*[*argument...*]28579 **DESCRIPTION**28580 The *printf* utility shall write formatted operands to the standard output. The *argument* operands
28581 shall be formatted under control of the *format* operand.28582 **OPTIONS**

28583 None.

28584 **OPERANDS**

28585 The following operands shall be supported:

28586 *format* A string describing the format to use to write the remaining operands. See the
28587 EXTENDED DESCRIPTION section.28588 *argument* The strings to be written to standard output, under the control of *format*. See the
28589 EXTENDED DESCRIPTION section.28590 **STDIN**

28591 Not used.

28592 **INPUT FILES**

28593 None.

28594 **ENVIRONMENT VARIABLES**28595 The following environment variables shall affect the execution of *printf*:28596 *LANG* Provide a default value for the internationalization variables that are unset or null.
28597 If *LANG* is unset or null, the corresponding value from the implementation-
28598 defined default locale shall be used. If any of the internationalization variables
28599 contains an invalid setting, the utility shall behave as if none of the variables had
28600 been defined.28601 *LC_ALL* If set to a non-empty string value, override the values of all the other
28602 internationalization variables.28603 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
28604 characters (for example, single-byte as opposed to multi-byte characters in
28605 arguments).28606 *LC_MESSAGES*28607 Determine the locale that should be used to affect the format and contents of
28608 diagnostic messages written to standard error.28609 *LC_NUMERIC*28610 Determine the locale for numeric formatting. It shall affect the format of numbers
28611 written using the *e*, *E*, *f*, *g*, and *G* conversion characters (if supported).28612 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.28613 **ASYNCHRONOUS EVENTS**

28614 Default.

28615 **STDOUT**

28616 See the EXTENDED DESCRIPTION section.

28617 **STDERR**

28618 Used only for diagnostic messages.

28619 **OUTPUT FILES**

28620 None.

28621 **EXTENDED DESCRIPTION**28622 The *format* operand shall be used as the *format* string described in the Base Definitions volume of
28623 IEEE Std. 1003.1-200x, Chapter 5, File Format Notation with the following exceptions:

- 28624 1. A <space> character in the format string, in any context other than a flag of a conversion
28625 specification, shall be treated as an ordinary character that is copied to the output.
- 28626 2. A ' Δ ' character in the format string shall be treated as a ' Δ ' character, not as a <space>
28627 character.
- 28628 3. In addition to the escape sequences shown in the Base Definitions volume of
28629 IEEE Std. 1003.1-200x, Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n',
28630 '\r', '\t', '\v'), "\ddd", where *ddd* is a one, two, or three-digit octal number, shall be
28631 written as a byte with the numeric value specified by the octal number.
- 28632 4. The implementation shall not precede or follow output from the *d* or *u* conversion
28633 specifications with <blank> characters not specified by the *format* operand.
- 28634 5. The implementation shall not precede output from the *o* conversion specification with
28635 zeros not specified by the *format* operand.
- 28636 6. The *e*, *E*, *f*, *g*, and *G* conversion specifications need not be supported.
- 28637 7. An additional conversion character, *b*, shall be supported as follows. The argument shall
28638 be taken to be a string that may contain backslash-escape sequences. The following
28639 backslash-escape sequences shall be supported:
- 28640 — The escape sequences listed in the Base Definitions volume of IEEE Std. 1003.1-200x,
28641 Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'),
28642 which shall be converted to the characters they represent
 - 28643 — "\0ddd", where *ddd* is a zero, one, two, or three-digit octal number that shall be
28644 converted to a byte with the numeric value specified by the octal number
 - 28645 — '\c', which shall not be written and shall cause *printf* to ignore any remaining
28646 characters in the string operand containing it, any remaining string operands, and any
28647 additional characters in the *format* operand
- 28648 The interpretation of a backslash followed by any other sequence of characters is
28649 unspecified.
- 28650 Bytes from the converted string shall be written until the end of the string or the number of
28651 bytes indicated by the precision specification is reached. If the precision is omitted, it shall
28652 be taken to be infinite, so all bytes up to the end of the converted string shall be written.
- 28653 8. For each specification that consumes an argument, the next argument operand shall be
28654 evaluated and converted to the appropriate type for the conversion as specified below.
- 28655 9. The *format* operand shall be reused as often as necessary to satisfy the argument operands.
28656 Any extra *c* or *s* conversion specifications shall be evaluated as if a null string argument
28657 were supplied; other extra conversion specifications shall be evaluated as if a zero
28658 argument were supplied. If the *format* operand contains no conversion specifications and

- 28659 *argument* operands are present, the results are unspecified.
- 28660 10. If a character sequence in the *format* operand begins with a '%' character, but does not
28661 form a valid conversion specification, the behavior is unspecified.
- 28662 The *argument* operands shall be treated as strings if the corresponding conversion character is *b*,
28663 *c*, or *s*; otherwise, it shall be evaluated as a C constant, as described by the ISO C standard, with
28664 the following extensions:
- 28665 • A leading plus or minus sign shall be allowed.
 - 28666 • If the leading character is a single-quote or double-quote, the value shall be the numeric
28667 value in the underlying codeset of the character following the single-quote or double-quote.
- 28668 If an argument operand cannot be completely converted into an internal value appropriate to
28669 the corresponding conversion specification, a diagnostic message shall be written to standard
28670 error and the utility shall not exit with a zero exit status, but shall continue processing any
28671 remaining operands and shall write the value accumulated at the time the error was detected to
28672 standard output.
- 28673 It is not considered an error if an argument operand is not completely used for a *c* or *s*
28674 conversion or if a string operand's first or second character is used to get the numeric value of a
28675 character.
- 28676 **EXIT STATUS**
- 28677 The following exit values shall be returned:
- 28678 0 Successful completion.
 - 28679 >0 An error occurred.
- 28680 **CONSEQUENCES OF ERRORS**
- 28681 Default.
- 28682 **APPLICATION USAGE**
- 28683 The floating-point formatting conversion specifications of *printf()* are not required because all
28684 arithmetic in the shell is integer arithmetic. The *awk* utility performs floating-point calculations
28685 and provides its own **printf** function. The *bc* utility can perform arbitrary-precision floating-
28686 point arithmetic, but does not provide extensive formatting capabilities. (This *printf* utility
28687 cannot really be used to format *bc* output; it does not support arbitrary precision.)
28688 Implementations are encouraged to support the floating-point conversions as an extension.
- 28689 Note that this *printf* utility, like the *printf()* function defined in the System Interfaces volume of
28690 IEEE Std. 1003.1-200x on which it is based, makes no special provision for dealing with multi-
28691 byte characters when using the *%c* conversion specification or when a precision is specified in a
28692 *%b* or *%s* conversion specification. Applications should be extremely cautious using either of
28693 these features when there are multi-byte characters in the character set.
- 28694 No provision is made in this volume of IEEE Std. 1003.1-200x which allows field widths and
28695 precisions to be specified as '*' since the '*' can be replaced directly in the *format* operand
28696 using shell variable substitution. Implementations can also provide this feature as an extension
28697 if they so choose.
- 28698 Hexadecimal character constants as defined in the ISO C standard are not recognized in the
28699 *format* operand because there is no consistent way to detect the end of the constant. Octal
28700 character constants are limited to, at most, three octal digits, but hexadecimal character
28701 constants are only terminated by a non-hex-digit character. In the ISO C standard, the "##"
28702 concatenation operator can be used to terminate a constant and follow it with a hexadecimal
28703 character to be written. In the shell, concatenation occurs before the *printf* utility has a chance to

28704 parse the end of the hexadecimal constant.

28705 The `%b` conversion specification is not part of the ISO C standard; it has been added here as a
 28706 portable way to process backslash escapes expanded in string operands as provided by the `echo`
 28707 utility. See also the APPLICATION USAGE section of `echo` (on page 2543) for ways to use `printf`
 28708 as a replacement for all of the traditional versions of the `echo` utility.

28709 If an argument cannot be parsed correctly for the corresponding conversion specification, the
 28710 `printf` utility is required to report an error. Thus, overflow and extraneous characters at the end
 28711 of an argument being used for a numeric conversion shall be reported as errors.

28712 EXAMPLES

28713 To alert the user and then print and read a series of prompts:

```
28714 printf "\aPlease fill in the following: \nName: "
28715 read name
28716 printf "Phone number: "
28717 read phone
```

28718 To read out a list of right and wrong answers from a file, calculate the percentage correctly, and
 28719 print them out. The numbers are right-justified and separated by a single `<tab>` character. The
 28720 percentage is written to one decimal place of accuracy:

```
28721 while read right wrong ; do
28722     percent=$(echo "scale=1;($right*100)/($right+$wrong)" | bc)
28723     printf "%2d right\t%2d wrong\t(%%s%%)\n" \
28724         $right $wrong $percent
28725 done < database_file
```

28726 The command:

```
28727 printf "%5d%4d\n" 1 21 321 4321 54321
```

28728 produces:

```
28729     1  21
28730     3214321
28731 54321   0
```

28732 Note that the `format` operand is used three times to print all of the given strings and that a `'0'`
 28733 was supplied by `printf` to satisfy the last `%4d` conversion specification.

28734 The `printf` utility is required to notify the user when conversion errors are detected while
 28735 producing numeric output; thus, the following results would be expected on an implementation
 28736 with 32-bit twos-complement integers when `%d` is specified as the `format` operand:

Argument	Standard Output	Diagnostic Output
5a	5	<code>printf: "5a" not completely converted</code>
9999999999	2147483647	<code>printf: "9999999999" arithmetic overflow</code>
-9999999999	-2147483648	<code>printf: "-9999999999" arithmetic overflow</code>
ABC	0	<code>printf: "ABC" expected numeric value</code>

28743 The diagnostic message format is not specified, but these examples convey the type of
 28744 information that should be reported. Note that the value shown on standard output is what
 28745 would be expected as the return value from the `strtol()` function as defined in the System
 28746 Interfaces volume of IEEE Std. 1003.1-200x. A similar correspondence exists between `%u` and
 28747 `strtoul()` and `%e`, `%f`, and `%g` (if the implementation supports floating-point conversions) and
 28748 `strtod()`.

- 28749 In a locale using the ISO/IEC 646: 1991 standard as the underlying codeset, the command:
- 28750 `printf "%d\n" 3 +3 -3 \'3 \"+3 "'-3"`
- 28751 produces:
- 28752 3 Numeric value of constant 3
- 28753 3 Numeric value of constant 3
- 28754 -3 Numeric value of constant -3
- 28755 51 Numeric value of the character '3' in the ISO/IEC 646: 1991 standard codeset
- 28756 43 Numeric value of the character '+' in the ISO/IEC 646: 1991 standard codeset
- 28757 45 Numeric value of the character '-' in the ISO/IEC 646: 1991 standard codeset
- 28758 Note that in a locale with multi-byte characters, the value of a character is intended to be the
- 28759 value of the equivalent of the `wchar_t` representation of the character as described in the System
- 28760 Interfaces volume of IEEE Std. 1003.1-200x.
- 28761 **RATIONALE**
- 28762 The *printf* utility was added to provide functionality that has historically been provided by *echo*.
- 28763 However, due to irreconcilable differences in the various versions of *echo* extant, the version has
- 28764 few special features, leaving those to this new *printf* utility, which is based on one in the Ninth
- 28765 Edition system.
- 28766 The EXTENDED DESCRIPTION section almost exactly matches the *printf()* function in the
- 28767 ISO C standard, although it is described in terms of the file format notation in the Base
- 28768 Definitions volume of IEEE Std. 1003.1-200x, Chapter 5, File Format Notation.
- 28769 **FUTURE DIRECTIONS**
- 28770 None.
- 28771 **SEE ALSO**
- 28772 *awk*, *bc*, *echo*, the System Interfaces volume of IEEE Std. 1003.1-200x, *printf()*
- 28773 **CHANGE HISTORY**
- 28774 First released in Issue 4.

28775 **NAME**28776 prs — print an SCCS file (**DEVELOPMENT**)28777 **SYNOPSIS**28778 xSI prs [-a][-d *dataspec*][-r[*SID*]] *file...*28779 xSI prs [-e|-l] -c *cutoff* [-d *dataspec*] *file...*28780 xSI prs [-e|-l] -r[*SID*][-d *dataspec*]*file...*

28781

28782 **DESCRIPTION**28783 The *prs* utility shall write to standard output parts or all of an SCCS file in a user-supplied
28784 format.28785 **OPTIONS**28786 The *prs* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
28787 12.2, Utility Syntax Guidelines, except that the *-r* option has an optional option-argument. This
28788 optional option-argument cannot be presented as a separate argument. The following options
28789 shall be supported:28790 *-d dataspec* Specify the output data specification. The *dataspec* shall be a string consisting of
28791 SCCS file *data keywords* (see **Data Keywords** (on page 2956)) interspersed with
28792 optional user-supplied text.28793 *-r[SID]* Specify the SCCS identification string (SID) of a delta for which information is
28794 desired. If no *SID* option-argument is specified, the SID of the most recently
28795 created delta is assumed.28796 *-e* Request information for all deltas created earlier than and including the delta
28797 designated via the *-r* option or the date-time given by the *-c* option.28798 *-l* Request information for all deltas created later than and including the delta
28799 designated via the *-r* option or the date-time given by the *-c* option.28800 *-c cutoff* Indicate the *cutoff* date-time, in the form:

28801 YY[MM[DD[HH[MM[SS]]]]]

28802 For the YY component, values in the range [69-99] shall refer to years in the
28803 twentieth century (1969 to 1999 inclusive); values in the range [00-68] shall refer to
28804 years in the twenty-first century (2000 to 2068 inclusive).28805 No changes (deltas) to the SCCS file that were created after the specified *cutoff*
28806 date-time shall be included in the output. Units omitted from the date-time default
28807 to their maximum possible values; for example, *-c 7502* is equivalent to
28808 *-c 750228235959*.28809 *-a* Request writing of information for both removed, that is, *delta type=R* (see *rm*
28810 *del* (on page 3037)) and existing, that is, *delta type=D*, deltas. If the *-a* option is not
28811 specified, information for existing deltas only shall be provided.28812 **OPERANDS**

28813 The following operand shall be supported:

28814 *file* A path name of an existing SCCS file or a directory. If *file* is a directory, the *prs*
28815 utility shall behave as though each file in the directory were specified as a named
28816 file, except that non-SCCS files (last component of the path name does not begin
28817 with *s.*) and unreadable files shall be silently ignored.

28818 If a single instance *file* is specified as *'-'*, the standard input shall be read; each
28819 line of the standard input shall be taken to be the name of an SCCS file to be
28820 processed. Non-SCCS files and unreadable files shall be silently ignored.

28821 **STDIN**

28822 The standard input shall be a text file used only when the *file* operand is specified as *'-'*. Each
28823 line of the text file shall be interpreted as an SCCS path name.

28824 **INPUT FILES**

28825 Any SCCS files displayed are files of an unspecified format.

28826 **ENVIRONMENT VARIABLES**

28827 The following environment variables shall affect the execution of *prs*:

28828 *LANG* Provide a default value for the internationalization variables that are unset or null.
28829 If *LANG* is unset or null, the corresponding value from the implementation-
28830 defined default locale shall be used. If any of the internationalization variables
28831 contains an invalid setting, the utility shall behave as if none of the variables had
28832 been defined.

28833 *LC_ALL* If set to a non-empty string value, override the values of all the other
28834 internationalization variables.

28835 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
28836 characters (for example, single-byte as opposed to multi-byte characters in
28837 arguments and input files).

28838 *LC_MESSAGES*

28839 Determine the locale that should be used to affect the format and contents of
28840 diagnostic messages written to standard error.

28841 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

28842 **ASYNCHRONOUS EVENTS**

28843 Default.

28844 **STDOUT**

28845 The standard output shall be a text file whose format is dependent on the data keywords
28846 specified with the *-d* option.

28847 **Data Keywords**

28848 Data keywords specify which parts of an SCCS file shall be retrieved and output. All parts of an
28849 SCCS file have an associated data keyword. A data keyword may appear in a *dataspec* multiple
28850 times.

28851 The information written by *prs* consists of:

- 28852 1. The user-supplied text
- 28853 2. Appropriate values (extracted from the SCCS file) substituted for the recognized data
28854 keywords in the order of appearance in the *dataspec*

28855 The format of a data keyword value shall either be simple (*'S'*), in which keyword substitution
28856 is direct, or multi-line (*'M'*).

28857 User-supplied text shall be any text other than recognized data keywords. A *<tab>* character
28858 shall be specified by *'\t'* and *<newline>* by *'\n'*. When the *-r* option is not specified, the
28859 default *dataspec* shall be:

28860 :PN: :\n\n

28861 and the following *dataspec* shall be used for each selected delta:

28862 :Dt: \t:DL: \nMRs: \n:MR: COMMENTS: \n:C: \n

28863

28864

28865

28866

28867

28868

28869

28870

28871

28872

28873

28874

28875

28876

28877

28878

28879

28880

28881

28882

28883

28884

28885

28886

28887

28888

28889

28890

28891

28892

28893

28894

28895

28896

28897

28898

28899

28900

28901

28902

28903

28904

28905

28906

SCCS File Data Keywords				
Keyword	Data Item	File Section	Value	Format
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/Ld:/Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	D or R	S
:I:	SCCS ID string (SID)	"	See below**	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date delta created	"	:Dy:/Dm:/Dd:	S
:Dy:	Year delta created	"	nn	S
:Dm:	Month delta created	"	nn	S
:Dd:	Day delta created	"	nn	S
:T:	Time delta created	"	:Th::Tm::Ts:	S
:Th:	Hour delta created	"	nn	S
:Tm:	Minutes delta created	"	nn	S
:Ts:	Seconds delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta sequence number	"	nnnn	S
:DI:	Sequence number of deltas included, excluded or ignored	"	:Dn:/Dx:/Dg:	S
:Dn:	Deltas included (sequence #)	"	:DS: :DS: ...	S
:Dx:	Deltas excluded (sequence #)	"	:DS: :DS: ...	S
:Dg:	Deltas ignored (sequence #)	"	:DS: :DS: ...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	yes or no	S
:MP:	MR validation program name	"	text	S
:KF:	Keyword error, warning flag	"	yes or no	S
:KV:	Keyword validation string	"	text	S
:BF:	Branch flag	"	yes or no	S
:J:	Joint edit flag	"	yes or no	S
:LK:	Locked releases	"	:R: ...	S
:Q:	User-defined keyword	"	text	S

28907
28908
28909
28910
28911
28912
28913
28914
28915
28916
28917
28918
28919
28920
28921
28922

SCCS File Data Keywords				
Keyword	Data Item	File Section	Value	Format
:M:	Module name	"	<i>text</i>	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	yes or no	S
:FD:	File descriptive text	Comments	<i>text</i>	M
:BD:	Body	Body	<i>text</i>	M
:GB:	Gotten body	"	<i>text</i>	M
:W:	A form of <i>what</i> string	N/A	:Z::M:\t:I:	S
:A:	A form of <i>what</i> string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	<i>what</i> string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	<i>text</i>	S
:PN:	SCCS file path name	N/A	<i>text</i>	S

28923 * :Dt::DT: :I: :D: :T: :P: :DS: :DP:
28924 ** :R::L::B::S: if the delta is a branch delta (:BF:= =yes)
28925 :R::L: if the delta is not a branch delta (:BF:= =no)

28926 **STDERR**

28927 Used only for diagnostic messages.

28928 **OUTPUT FILES**

28929 None.

28930 **EXTENDED DESCRIPTION**

28931 None.

28932 **EXIT STATUS**

28933 The following exit values shall be returned:

28934 0 Successful completion.

28935 >0 An error occurred.

28936 **CONSEQUENCES OF ERRORS**

28937 Default.

28938 **APPLICATION USAGE**

28939 None.

28940 **EXAMPLES**

28941 1. The following example:

28942 prs -d "User Names for :F: are:\n:UN:" s.file

28943 may write to standard output:

28944 User Names for s.file are:

28945 xyz

28946 131

28947 abc

28948 2. The following example:

28949 prs -d "Delta for pgm :M:: :I: - :D: By :P:" -r s.file

28950 may write to standard output:
 28951 Delta for pgm main.c: 3.7 - 77/12/01 By cas
 28952 3. As a special case:
 28953 prs s.file
 28954 may write to standard output:
 28955 s.file:
 28956 <blank line>
 28957 D 1.1 77/12/01 00:00:00 cas 1 000000/00000/00000
 28958 MRs:
 28959 b178-12345
 28960 b179-54321
 28961 COMMENTS:
 28962 this is the comment line for s.file initial delta
 28963 <blank line>
 28964 for each delta table entry of the **D** type. The only option allowed to be used with this
 28965 special case is the **-a** option.

28966 **RATIONALE**
 28967 None.

28968 **FUTURE DIRECTIONS**
 28969 None.

28970 **SEE ALSO**
 28971 *admin, delta, get, what*

28972 **CHANGE HISTORY**
 28973 First released in Issue 2.

28974 **Issue 4**
 28975 Format reorganized.
 28976 Exceptions to Utility Syntax Guidelines conformance noted.
 28977 Internationalized environment variable support mandated.

28978 **Issue 5**
 28979 The phrase “in which keyword substitution is followed by a <newline>” is deleted from the end
 28980 of the second paragraph of **Data Keywords** (on page 2956).
 28981 The interpretation of the **YY** component of the **-c cutoff** argument is noted.

28982 **Issue 6**
 28983 The normative text is reworded to emphasise the term “shall” for implementation requirements.

28984 NAME

28985 ps — report process status

28986 SYNOPSIS

28987 UP XSI ps [-aA][-defl][-G *grouplist*][-o *format*]...[-p *proclist*][-t *termlist*]
 28988 [-U *userlist*][-g *grouplist*][-n *namelist*][-u *userlist*]
 28989

28990 DESCRIPTION

28991 The *ps* utility shall write information about processes, subject to having the appropriate
 28992 privileges to obtain information about those processes.

28993 By default, *ps* selects all processes with the same effective user ID as the current user and the
 28994 same controlling terminal as the invoker.

28995 OPTIONS

28996 The *ps* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
 28997 Utility Syntax Guidelines.

28998 The following options shall be supported:

28999 **-a** Write information for all processes associated with terminals. Implementations
 29000 may omit session leaders from this list.

29001 **-A** Write information for all processes.

29002 XSI **-d** Write information for all processes, except session leaders.

29003 XSI **-e** Write information for all processes. (Equivalent to **-A**.)

29004 XSI **-f** Generate a **full** listing. (See the STDOUT section for the contents of a **full** listing.)

29005 XSI **-g *grouplist*** Write information for processes whose session leaders are given in *grouplist*. The
 29006 application shall ensure that the *grouplist* is a single argument in the form of a
 29007 <blank> or comma-separated list.

29008 **-G *grouplist*** Write information for processes whose real group ID numbers are given in
 29009 *grouplist*. The application shall ensure that the *grouplist* is a single argument in the
 29010 form of a <blank> or comma-separated list.

29011 XSI **-l** Generate a **long** listing. (See the STDOUT section for the contents of a **long** listing.)
 29012

29013 XSI **-n *namelist*** Specify the name of an alternative system *namelist* file in place of the default. The
 29014 name of the default file and the format of a *namelist* file are unspecified.

29015 **-o *format*** Write information according to the format specification given in *format*. This is
 29016 fully described in the STDOUT section. Multiple **-o** options can be specified; the
 29017 format specification shall be interpreted as the <space> character-separated
 29018 concatenation of all the *format* option-arguments.

29019 **-p *proclist*** Write information for processes whose process ID numbers are given in *proclist*.
 29020 The application shall ensure that the *proclist* is a single argument in the form of a
 29021 <blank> or comma-separated list.

29022 **-t *termlist*** Write information for processes associated with terminals given in *termlist*. The
 29023 application shall ensure that the *termlist* is a single argument in the form of a
 29024 <blank> or comma-separated list. Terminal identifiers shall be given in an
 29025 XSI implementation-defined format. On XSI-conformant systems, they shall be given
 29026 in one of two forms: the device's file name (for example, **tty04**) or, if the device's

- 29027 file name starts with **tty**, just the identifier following the characters **tty** (for
29028 example, "04").
- 29029 XSI **-u *userlist*** Write information for processes whose user ID numbers or login names are given
29030 in *userlist*. The application shall ensure that the *userlist* is a single argument in the
29031 form of a <blank> or comma-separated list. In the listing, the numerical user ID is
29032 written unless the **-f** option is used, in which case the login name is written.
- 29033 **-U *userlist*** Write information for processes whose real user ID numbers or login names are
29034 given in *userlist*. The application shall ensure that the *userlist* is a single argument
29035 in the form of a <blank> or comma-separated list.
- 29036 With the exception of **-o *format***, all of the options shown are used to select processes. If any are
29037 specified, the default list shall be ignored and *ps* shall select the processes represented by the
29038 bitwise-inclusive OR of all the selection-criteria options.
- 29039 **OPERANDS**
- 29040 None.
- 29041 **STDIN**
- 29042 Not used.
- 29043 **INPUT FILES**
- 29044 None.
- 29045 **ENVIRONMENT VARIABLES**
- 29046 The following environment variables shall affect the execution of *ps*:
- 29047 **COLUMNS** Override the system-selected horizontal screen size, used to determine the number
29048 of text columns to display. See the Base Definitions volume of
29049 IEEE Std. 1003.1-200x, Chapter 8, Environment Variables for valid values and
29050 results when it is unset or null.
- 29051 **LANG** Provide a default value for the internationalization variables that are unset or null.
29052 If *LANG* is unset or null, the corresponding value from the implementation-
29053 defined default locale shall be used. If any of the internationalization variables
29054 contains an invalid setting, the utility shall behave as if none of the variables had
29055 been defined.
- 29056 **LC_ALL** If set to a non-empty string value, override the values of all the other
29057 internationalization variables.
- 29058 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
29059 characters (for example, single-byte as opposed to multi-byte characters in
29060 arguments).
- 29061 **LC_MESSAGES**
- 29062 Determine the locale that should be used to affect the format and contents of
29063 diagnostic messages written to standard error and informative messages written to
29064 standard output.
- 29065 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 29066 **LC_TIME** Determine the format and contents of the date and time strings displayed.
- 29067 **ASYNCHRONOUS EVENTS**
- 29068 Default.

29069 **STDOUT**

29070 When the **-o** option is not specified, the standard output format is unspecified.

29071 XSI On XSI-conformant systems, the output format is as follows. The column headings and
 29072 descriptions of the columns in a *ps* listing are given below. The precise meanings of these fields
 29073 are implementation-defined. The letters 'f' and 'l' (below) indicate the option (**full** or **long**)
 29074 that shall cause the corresponding heading to appear; **all** means that the heading always
 29075 appears. Note that these two options determine only what information is provided for a process;
 29076 they do not determine which processes are listed.

29077	F	(l)	Flags (octal and additive) associated with the process.
29078	S	(l)	The state of the process.
29079	UID	(f,l)	The user ID number of the process owner; the login name is printed under the -f option.
29081	PID	(all)	The process ID of the process; it is possible to kill a process if this datum is known.
29082			
29083	PPID	(f,l)	The process ID of the parent process.
29084	C	(f,l)	Processor utilization for scheduling.
29085	PRI	(l)	The priority of the process; higher numbers mean lower priority.
29086	NI	(l)	Nice value; used in priority computation.
29087	ADDR	(l)	The address of the process.
29088	SZ	(l)	The size in blocks of the core image of the process.
29089	WCHAN	(l)	The event for which the process is waiting or sleeping; if blank, the process is running.
29090			
29091	STIME	(f)	Starting time of the process.
29092	TTY	(all)	The controlling terminal for the process.
29093	TIME	(all)	The cumulative execution time for the process.
29094	CMD	(all)	The command name; the full command name and its arguments are written under the -f option.
29095			

29096 A process that has exited and has a parent, but has not yet been waited for by the parent, is
 29097 marked **defunct**.

29098 Under the option **-f**, *ps* tries to determine the command name and arguments given when the
 29099 process was created by examining memory or the swap area. Failing this, the command name, as
 29100 it would appear without the option **-f**, is written in square brackets.

29101 The **-o** option allows the output format to be specified under user control.

29102 The application shall ensure that the format specification is a list of names presented as a single
 29103 argument, <blank> or comma-separated. Each variable has a default header. The default header
 29104 can be overridden by appending an equals sign and the new text of the header. The rest of the
 29105 characters in the argument shall be used as the header text. The fields specified shall be written
 29106 in the order specified on the command line, and should be arranged in columns in the output.
 29107 The field widths shall be selected by the system to be at least as wide as the header text (default
 29108 or overridden value). If the header text is null, such as **-o user=**, the field width shall be at least
 29109 as wide as the default header text. If all header text fields are null, no header line shall be
 29110 written.

29111 The following names are recognized in the POSIX locale:

29112 **ruser** The real user ID of the process. This shall be the textual user ID, if it can be obtained
 29113 and the field width permits, or a decimal representation otherwise.

29114	user	The effective user ID of the process. This shall be the textual user ID, if it can be
29115		obtained and the field width permits, or a decimal representation otherwise.
29116	rgroup	The real group ID of the process. This shall be the textual group ID, if it can be obtained
29117		and the field width permits, or a decimal representation otherwise.
29118	group	The effective group ID of the process. This shall be the textual group ID, if it can be
29119		obtained and the field width permits, or a decimal representation otherwise.
29120	pid	The decimal value of the process ID.
29121	ppid	The decimal value of the parent process ID.
29122	pgid	The decimal value of the process group ID.
29123	pcpu	The ratio of CPU time used recently to CPU time available in the same period,
29124		expressed as a percentage. The meaning of “recently” in this context is unspecified. The
29125		CPU time available is determined in an unspecified manner.
29126	vsz	The size of the process in (virtual) memory in kilobytes as a decimal integer.
29127	nice	The decimal value of the nice value of the process; see <i>nice</i> (on page 2872).
29128	etime	In the POSIX locale, the elapsed time since the process was started, in the form:
29129		[[<i>dd</i> -] <i>hh</i> :] <i>mm</i> : <i>ss</i>
29130		where <i>dd</i> shall represent the number of days, <i>hh</i> the number of hours, <i>mm</i> the number
29131		of minutes, and <i>ss</i> the number of seconds. The <i>dd</i> field shall be a decimal integer. The
29132		<i>hh</i> , <i>mm</i> , and <i>ss</i> fields shall be two-digit decimal integers padded on the left with zeros.
29133	time	In the POSIX locale, the cumulative CPU time of the process in the form:
29134		[[<i>dd</i> -] <i>hh</i> : <i>mm</i> : <i>ss</i>
29135		The <i>dd</i> , <i>hh</i> , <i>mm</i> , and <i>ss</i> fields shall be as described in the etime specifier.
29136	tty	The name of the controlling terminal of the process (if any) in the same format used by
29137		the <i>who</i> utility.
29138	comm	The name of the command being executed (<i>argv</i> [0] value) as a string.
29139	args	The command with all its arguments as a string. The implementation may truncate this
29140		value to the field width; it is implementation-defined whether any further truncation
29141		occurs. It is unspecified whether the string represented is a version of the argument list
29142		as it was passed to the command when it started, or is a version of the arguments as
29143		they may have been modified by the application. Applications cannot depend on being
29144		able to modify their argument list and having that modification be reflected in the
29145		output of <i>ps</i> .
29146		Any field need not be meaningful in all implementations. In such a case a hyphen (‘-’) should
29147		be output in place of the field value.
29148		Only comm and args shall be allowed to contain <blank> characters; all others shall not. Any
29149		implementation-defined variables shall be specified in the system documentation along with the
29150		default header and indicating if the field may contain <blank> characters.
29151		The following table specifies the default header to be used in the POSIX locale corresponding to
29152		each format specifier.

29153

Table 4-17 Variable Names and Default Headers in *ps*

29154

Format Specifier	Default Header	Format Specifier	Default Header
args	COMMAND	ppid	PPID
comm	COMMAND	rgroup	RGROUP
etime	ELAPSED	ruser	RUSER
group	GROUP	time	TIME
nice	NI	tty	TT
pcpu	%CPU	user	USER
pgid	PGID	vsz	VSZ
pid	PID		

29155

29156

29157

29158

29159

29160

29161

29162

29163 STDERR

29164 Used only for diagnostic messages.

29165 OUTPUT FILES

29166 None.

29167 EXTENDED DESCRIPTION

29168 None.

29169 EXIT STATUS

29170 The following exit values shall be returned:

29171 0 Successful completion.

29172 >0 An error occurred.

29173 CONSEQUENCES OF ERRORS

29174 Default.

29175 APPLICATION USAGE29176 Things can change while *ps* is running; the snapshot it gives is only true for an instant, and might
29177 not be accurate by the time it is displayed.29178 The **args** format specifier is allowed to produce a truncated version of the command arguments.
29179 In some implementations, this information is no longer available when the *ps* utility is executed.29180 If the field width is too narrow to display a textual ID, the system may use a numeric version.
29181 Normally, the system would be expected to choose large enough field widths, but if a large
29182 number of fields were selected to write, it might squeeze fields to their minimum sizes to fit on
29183 one line. One way to ensure adequate width for the textual IDs is to override the default header
29184 for a field to make it larger than most or all user or group names.29185 There is no special quoting mechanism for header text. The header text is the rest of the
29186 argument. If multiple header changes are needed, multiple **-o** options can be used, such as:29187 `ps -o "user=User Name" -o pid=Process\ ID`29188 On some systems, especially multi-level secure systems, *ps* may be severely restricted and
29189 produce information only about child processes owned by the user.**29190 EXAMPLES**

29191 The command:

29192 `ps -o user,pid,ppid=MOM -o args`

29193 writes at least the following in the POSIX locale:

29194 USER PID MOM COMMAND

29195 helene 34 12 ps -o uid,pid,ppid=MOM -o args

29196 The contents of the **COMMAND** field need not be the same in all implementations, due to
29197 possible truncation.

29198 **RATIONALE**

29199 There is very little commonality between BSD and System V implementations of *ps*. Many
29200 options conflict or have subtly different usages. The standard developers attempted to select a
29201 set of options that were useful on a wide range of systems and selected options that either can be
29202 implemented on both BSD and System V-based systems without breaking the current
29203 implementations or where the options are sufficiently similar that any changes would not be
29204 unduly problematic for users or implementors.

29205 It is recognized that on some systems, especially multi-level secure systems, *ps* may be nearly
29206 useless. The default output has therefore been chosen such that it does not break historical
29207 implementations and also is likely to provide at least some useful information on most systems.

29208 The major change is the addition of the format specification capability. The motivation for this
29209 invention is to provide a mechanism for users to access a wider range of system information, if
29210 the system permits it, in a portable manner. The fields chosen to appear in this volume of
29211 IEEE Std. 1003.1-200x were arrived at after considering what concepts were likely to be both
29212 reasonably useful to the “average” user and had a reasonable chance of being implemented on a
29213 wide range of systems. Again it is recognized that not all systems are able to provide all the
29214 information and, conversely, some may wish to provide more. It is hoped that the approach
29215 adopted will be sufficiently flexible and extensible to accommodate most systems.
29216 Implementations may be expected to introduce new format specifiers.

29217 The default output should consist of a short listing containing the process ID, terminal name,
29218 cumulative execution time, and command name of each process.

29219 The preference of the standard developers would have been to make the format specification an
29220 operand of the *ps* command. Unfortunately, BSD usage precluded this.

29221 At one time a format was included to display the environment array of the process. This was
29222 deleted because there is no portable way to display it.

29223 The **-A** option is equivalent to the BSD **-g** and the SVID **-e**. Because the two systems differed, a
29224 mnemonic compromise was selected.

29225 The **-a** option is described with some optional behavior because the SVID omits session leaders,
29226 but BSD does not.

29227 In an early proposal, format specifiers appeared for priority and start time. The former was not
29228 defined adequately in this volume of IEEE Std. 1003.1-200x and was removed in deference to the
29229 defined nice value; the latter because elapsed time was considered to be more useful.

29230 In a new BSD version of *ps*, a **-O** option can be used to write all of the default information,
29231 followed by additional format specifiers. This was not adopted because the default output is
29232 implementation-defined. Nevertheless, this is a useful option that should be reserved for that
29233 purpose. In the **-o** option for the POSIX Shell and Utilities *ps*, the format is the concatenation of
29234 each **-o**. Therefore, the user can have an alias or function that defines the beginning of their
29235 desired format and add more fields to the end of the output in certain cases where that would be
29236 useful.

29237 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*
29238 require that they all use the same format.

29239 The **pcpu** field indicates that the CPU time available is determined in an unspecified manner.
29240 This is because it is difficult to express an algorithm that is useful across all possible machine
29241 architectures. Historical counterparts to this value have attempted to show percentage of use in

29242 the recent past, such as the preceding minute. Frequently, these values for all processes did not
29243 add up to 100%. Implementations are encouraged to provide data in this field to users that will
29244 help them identify processes currently affecting the performance of the system.

29245 **FUTURE DIRECTIONS**

29246 None.

29247 **SEE ALSO**

29248 *kill, nice, renice*

29249 **CHANGE HISTORY**

29250 First released in Issue 2.

29251 **Issue 4**

29252 Aligned with the ISO/IEC 9945-2:1993 standard.

29253 **Issue 6**

29254 This utility is now marked as part of the User Portability Utilities option.

29255 The normative text is reworded to avoid use of the term “must” for application requirements.

29256 **NAME**

29257 pwd — return working directory name

29258 **SYNOPSIS**

29259 pwd [-L | -P]

29260 **DESCRIPTION**29261 The *pwd* utility shall write to standard output an absolute path name of the current working
29262 directory, which does not contain the file names dot or dot-dot.29263 **OPTIONS**29264 The *pwd* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
29265 12.2, Utility Syntax Guidelines.

29266 The following options shall be supported by the implementation:

29267 **-L** If the *PWD* environment variable contains an absolute path name of the current
29268 directory that does not contain the file names dot or dot-dot, *pwd* shall write this
29269 path name to standard output. Otherwise, the **-L** option shall behave as the **-P**
29270 option.29271 **-P** The absolute path name written shall not contain file names that, in the context of
29272 the path name, refer to files of type symbolic link.29273 If both **-L** and **-P** are specified, the last one shall apply. If neither **-L** nor **-P** is specified, the *pwd*
29274 utility shall behave as if **-L** had been specified.29275 **OPERANDS**

29276 None.

29277 **STDIN**

29278 Not used.

29279 **INPUT FILES**

29280 None.

29281 **ENVIRONMENT VARIABLES**29282 The following environment variables shall affect the execution of *pwd*:29283 **LANG** Provide a default value for the internationalization variables that are unset or null.
29284 If *LANG* is unset or null, the corresponding value from the implementation-
29285 defined default locale shall be used. If any of the internationalization variables
29286 contains an invalid setting, the utility shall behave as if none of the variables had
29287 been defined.29288 **LC_ALL** If set to a non-empty string value, override the values of all the other
29289 internationalization variables.29290 **LC_MESSAGES**29291 Determine the locale that should be used to affect the format and contents of
29292 diagnostic messages written to standard error.29293 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.29294 **PWD** If the **-P** option is in effect, this variable shall be set to an absolute path name of
29295 the current working directory that does not contain any components that specify
29296 symbolic links, does not contain any components that are dot, and does not
29297 contain any components that are dot-dot. If an application sets or unsets the value
29298 of *PWD*, the behavior of *pwd* is unspecified.

29299 **ASYNCHRONOUS EVENTS**

29300 Default.

29301 **STDOUT**29302 The *pwd* utility output is an absolute path name of the current working directory:

29303 "%s\n", <directory pathname>

29304 **STDERR**

29305 Used only for diagnostic messages.

29306 **OUTPUT FILES**

29307 None.

29308 **EXTENDED DESCRIPTION**

29309 None.

29310 **EXIT STATUS**

29311 The following exit values shall be returned:

29312 0 Successful completion.

29313 >0 An error occurred.

29314 **CONSEQUENCES OF ERRORS**

29315 If an error is detected, output shall not be written to standard output, a diagnostic message shall

29316 be written to standard error, and the exit status is not zero.

29317 **APPLICATION USAGE**

29318 None.

29319 **EXAMPLES**

29320 None.

29321 **RATIONALE**29322 Some implementations have historically provided *pwd* as a shell special built-in command.

29323 In most utilities, if an error occurs, partial output may be written to standard output. This does
29324 not happen in historical implementations of *pwd*. Because *pwd* is frequently used in historical
29325 shell scripts without checking the exit status, it is important that the historical behavior is
29326 required here; therefore, the CONSEQUENCES OF ERRORS section specifically disallows any
29327 partial output being written to standard output.

29328 **FUTURE DIRECTIONS**

29329 None.

29330 **SEE ALSO**29331 *cd*, the System Interfaces volume of IEEE Std. 1003.1-200x, *getcwd()*29332 **CHANGE HISTORY**

29333 First released in Issue 2.

29334 **Issue 4**

29335 Aligned with the ISO/IEC 9945-2: 1993 standard.

29336 **Issue 6**

29337 The *-P* and *-L* options are added to describe actions relating to symbolic links as specified in the
29338 IEEE P1003.2b draft standard.

29339 NAME

29340 qalter — alter batch job

29341 SYNOPSIS

```
29342 BE qalter [-a date_time][-A account_string][-c interval][-e path_name]
29343 [-h hold_list][-j join_list][-k keep_list][-l resource_list]
29344 [-m mail_options][-M mail_list][-N name][-o path_name]
29345 [-p priority][-r y|n][-S path_name_list][-u user_list]
29346 job_identifier ...
29347
```

29348 DESCRIPTION

29349 The attributes of a batch job are altered by a request to the batch server that manages the batch
 29350 job. The *qalter* utility is a user-accessible batch client that requests the alteration of the attributes
 29351 of one or more batch jobs.

29352 The *qalter* utility shall alter the attributes of those batch jobs, and only those batch jobs, for which
 29353 a batch *job_identifier* is presented to the utility.

29354 The *qalter* utility shall alter the attributes of batch jobs in the order in which the batch
 29355 *job_identifiers* are presented to the utility.

29356 If the *qalter* utility fails to process a batch *job_identifier* successfully, the utility shall proceed to
 29357 process the remaining batch *job_identifiers*, if any.

29358 For each batch *job_identifier* for which the *qalter* utility succeeds, each attribute of the identified
 29359 batch job shall be altered as indicated by all the options presented to the utility.

29360 For each identified batch job for which the *qalter* utility fails, the utility shall not alter any
 29361 attribute of the batch job.

29362 For each batch job that the *qalter* utility processes, the utility shall not modify any attribute other
 29363 than those required by the options and option-arguments presented to the utility.

29364 The *qalter* utility shall alter batch jobs by sending a *Modify Job Request* to the batch server that
 29365 manages each batch job. At the time the *qalter* utility exits, it shall have modified the batch job
 29366 corresponding to each successfully processed batch *job_identifier*. An attempt to alter the
 29367 attributes of a batch job in the RUNNING state is implementation-defined.

29368 OPTIONS

29369 The *qalter* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 29370 12.2, Utility Syntax Guidelines.

29371 The following options shall be supported by the implementation:

29372 **-a date_time** Redefine the time at which the batch job becomes eligible for execution.

29373 The *qalter* utility shall accept an option-argument that conforms to the syntax of
 29374 the *date_time* operand of the *touch* utility.

29375 The *qalter* utility shall set the *Execution_Time* attribute of the batch job to the
 29376 number of seconds since the Epoch that is equivalent to the local time expressed
 29377 by the value of the *date_time* option-argument. Specifying a *date_time* option-
 29378 argument that represents a time (number of seconds since the Epoch) earlier than
 29379 the time at which the utility exits shall have the same effect on batch job execution
 29380 as if the **-a** option had not been presented to the utility.

29381 **-A account_string**

29382 Redefine the account to which the resource consumption of the batch job should be
 29383 charged.

- 29384 The syntax of the *account_string* option-argument is unspecified.
- 29385 The *qalter* utility shall set the *Account_Name* attribute of the batch job to the value
29386 of the *account_string* option-argument.
- 29387 **-c interval** Redefine whether the batch job should be checkpointed, and if so, how often.
- 29388 The *qalter* utility shall accept a value for the interval option-argument that is one of
29389 the following:
- 29390 *n* No checkpointing is to be performed on the batch batch job
29391 (NO_CHECKPOINT).
- 29392 *s* Checkpointing is to be performed only when the batch server is shut
29393 down (CHECKPOINT_AT_SHUTDOWN).
- 29394 *c* Automatic periodic checkpointing is to be performed at the
29395 *Minimum_Cpu_Interval* attribute of the batch queue, in units of CPU
29396 minutes (CHECKPOINT_AT_MIN_CPU_INTERVAL).
- 29397 *c=minutes* Automatic periodic checkpointing is to be performed every *minutes*
29398 of CPU time, or every *Minimum_Cpu_Interval* minutes, whichever is
29399 greater. The *minutes* argument shall conform to the syntax for
29400 unsigned integers and shall be greater than zero.
- 29401 An implementation may define other checkpoint intervals. The conformance
29402 document for an implementation shall describe any alternative checkpoint
29403 intervals, how they are specified, their internal behavior, and how they affect the
29404 behavior of the utility.
- 29405 The *qalter* utility shall set the *Checkpoint* attribute of the batch job to the value of the
29406 *interval* option-argument.
- 29407 **-e path_name** Redefine the path to be used for the standard error stream of the batch job.
- 29408 The *qalter* utility shall accept a *path_name* option-argument that conforms to the
29409 syntax of the *path_name* element defined in the POSIX.1-1990 standard, which can
29410 be preceded by a host name element of the form *hostname:*.
- 29411 If the *path_name* option-argument constitutes an absolute path name, the *qalter*
29412 utility shall set the *Error_Path* attribute of the batch job to the value of the
29413 *path_name* option-argument, including the host name element, if present.
- 29414 If the *path_name* option-argument constitutes a relative path name and no host
29415 name element is specified, the *qalter* utility shall set the *Error_Path* attribute of the
29416 batch job to the value of the absolute path name derived by expanding the
29417 *path_name* option-argument relative to the current directory of the process that
29418 executes the *qalter* utility.
- 29419 If the *path_name* option-argument constitutes a relative path name and a host name
29420 element is specified, the *qalter* utility shall set the *Error_Path* attribute of the batch
29421 job to the value of the option-argument without expansion.
- 29422 If the *path_name* option-argument does not include a host name element, the *qalter*
29423 utility shall prefix the path name in the *Error_Path* attribute with *hostname:*, where
29424 *hostname* is the name of the host upon which the *qalter* utility is being executed.
- 29425 **-h hold_list** Redefine the types of holds, if any, on the batch job. The *qalter* **-h** option shall
29426 accept a value for the *hold_list* option-argument that is a string of alphanumeric
29427 characters in the portable character set.

29428 The *qalter* utility shall accept a value for the *hold_list* option-argument that is a
 29429 string of one or more of the characters 'u', 's', or 'o', or the single character
 29430 'n'. For each unique character in the *hold_list* option-argument, the *qalter* utility
 29431 shall add a value to the *Hold_Types* attribute of the batch job as follows, each
 29432 representing a different hold type:

29433 *u* USER

29434 *s* SYSTEM

29435 *o* OPERATOR

29436 If any of these characters are duplicated in the *hold_list* option-argument, the
 29437 duplicates shall be ignored. An existing *Hold_Types* attribute can be cleared by the
 29438 hold type:

29439 *n* NO_HOLD

29440 The *qalter* utility shall consider it an error if any hold type other than **n** is combined
 29441 with hold type **n**. Strictly conforming applications shall not repeat any of the
 29442 characters 'u', 's', 'o', or 'n' within the *hold_list* option-argument. The *qalter*
 29443 utility shall permit the repetition of characters, but shall not assign additional
 29444 meaning to the repeated characters. An implementation may define other hold
 29445 types. The conformance document for an implementation shall describe any
 29446 additional hold types, how they are specified, their internal behavior, and how
 29447 they affect the behavior of the utility.

29448 **-j** *join_list* Redefine which streams of the batch job are to be merged. The *qalter* **-j** option shall
 29449 accept a value for the *join_list* option-argument that is a string of alphanumeric
 29450 characters in the portable character set.

29451 The *qalter* utility shall accept a *join_list* option-argument that consists of one or
 29452 more of the characters 'e' and 'o', or the single character 'n'.

29453 All of the other batch job output streams specified shall be merged into the output
 29454 stream represented by the character listed first in the *join_list* option-argument.

29455 For each unique character in the *join_list* option-argument, the *qalter* utility shall
 29456 add a value to the *Join_Path* attribute of the batch job as follows, each representing
 29457 a different batch job stream to join:

29458 *e* The standard error of the batch batch job (JOIN_STD_ERROR).

29459 *o* The standard output of the batch batch job (JOIN_STD_OUTPUT).

29460 An existing *Join_Path* attribute can be cleared by the join type:

29461 **n** NO_JOIN

29462 If **n** is specified, then no files are joined. The *qalter* utility shall consider it an error if
 29463 any join type other than **n** is combined with join type **n**.

29464 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or
 29465 'n' within the *join_list* option-argument. The *qalter* utility shall permit the
 29466 repetition of characters, but shall not assign additional meaning to the repeated
 29467 characters.

29468 An implementation may define other join types. The conformance document for an
 29469 implementation shall describe any additional batch job streams, how they are
 29470 specified, their internal behavior, and how they affect the behavior of the utility.

29471 **-k *keep_list*** Redefine which output of the batch job to retain on the execution host.

29472 The *qalter* **-k** option shall accept a value for the *keep_list* option-argument that is a
29473 string of alphanumeric characters in the portable character set.

29474 The *qalter* utility shall accept a *keep_list* option-argument that consists of one or
29475 more of the characters 'e' and 'o' or the single character 'n'.

29476 For each unique character in the *keep_list* option-argument, the *qalter* utility shall
29477 add a value to the *Keep_Files* attribute of the batch job as follows, each representing
29478 a different batch job stream to keep:

29479 **e** The standard error of the batch batch job (KEEP_STD_ERROR).

29480 **o** The standard output of the batch batch job (KEEP_STD_OUTPUT).

29481 If both 'e' and 'o' are specified, then both files are retained. An existing
29482 *Keep_Files* attribute can be cleared by the keep type:

29483 **n** NO_KEEP

29484 If **n** is specified, then no files are retained. The *qalter* utility shall consider it an error
29485 if any keep type other than **n** is combined with keep type **n**.

29486 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or
29487 'n' within the *keep_list* option-argument. The *qalter* utility shall permit the
29488 repetition of characters, but shall not assign additional meaning to the repeated
29489 characters. An implementation may define other keep types. The conformance
29490 document for an implementation shall describe any additional keep types, how
29491 they are specified, their internal behavior, and how they affect the behavior of the
29492 utility.

29493 **-l *resource_list***

29494 Redefine the resources that are allowed or required by the batch job.

29495 The *qalter* utility shall accept a *resource_list* option-argument that conforms to the
29496 following syntax:

29497 resource=value[, , resource=value , , . . .]

29498 The *qalter* utility shall set one entry in the value of the *Resource_List* attribute of the
29499 batch job for each resource listed in the *resource_list* option-argument.

29500 Because the list of supported resource names might vary by batch server, the *qalter*
29501 utility shall rely on the batch server to validate the resource names and associated
29502 values. See Section 3.3.3 (on page 2337) for a means of removing *keyword=value*
29503 (and *value@keyword*) pairs and other general rules for list-oriented batch job
29504 attributes.

29505 **-m *mail_options***

29506 Redefine the points in the execution of the batch job at which the batch server is to
29507 send mail about a change in the state of the batch job.

29508 The *qalter* **-m** option shall accept a value for the *mail_options* option-argument that
29509 is a string of alphanumeric characters in the portable character set.

29510 The *qalter* utility shall accept a value for the *mail_options* option-argument that is a
29511 string of one or more of the characters 'e', 'b', and 'a', or the single character
29512 'n'. For each unique character in the *mail_options* option-argument, the *qalter*
29513 utility shall add a value to the *Mail_Users* attribute of the batch job as follows, each
29514 representing a different time during the life of a batch job at which to send mail:

- 29515 **e** MAIL_AT_EXIT
- 29516 **b** MAIL_AT_BEGINNING
- 29517 **a** MAIL_AT_ABORT
- 29518 If any of these characters are duplicated in the *mail_options* option-argument, the
29519 duplicates shall be ignored.
- 29520 An existing *Mail_Points* attribute can be cleared by the mail type:
- 29521 **n** NO_MAIL
- 29522 If **n** is specified, then mail is not sent. The *qalter* utility shall consider it an error if
29523 any mail type other than **n** is combined with mail type **n**. Strictly conforming
29524 applications shall not repeat any of the characters 'e', 'b', 'a', or 'n' within
29525 the *mail_options* option-argument. The *qalter* utility shall permit the repetition of
29526 characters but shall not assign additional meaning to the repeated characters.
- 29527 An implementation may define other mail types. The conformance document for
29528 an implementation shall describe any additional mail types, how they are
29529 specified, their internal behavior, and how they affect the behavior of the utility.
- 29530 **-M mail_list** Redefine the list of users to which the batch server that executes the batch job is to
29531 send mail, if the batch server sends mail about the batch job.
- 29532 The syntax of the *mail_list* option-argument is unspecified. If the implementation
29533 of the *qalter* utility uses a name service to locate users, the utility shall accept the
29534 syntax used by the name service.
- 29535 If the implementation of the *qalter* utility does not use a name service to locate
29536 users, the implementation shall accept the following syntax for user names:
- 29537 mail_address[, mail_address , ...]
- 29538 The interpretation of *mail_address* is implementation-defined.
- 29539 The *qalter* utility shall set the *Mail_Users* attribute of the batch job to the value of
29540 the *mail_list* option-argument.
- 29541 **-N name** Redefine the name of the batch job.
- 29542 The *qalter* **-N** option shall accept a value for the *name* option argument that is a
29543 string of up to 15 alphanumeric characters in the portable character set where the
29544 first character is alphabetic.
- 29545 The syntax of the *name* option-argument is unspecified.
- 29546 The *qalter* utility shall set the *Job_Name* attribute of the batch job to the value of the
29547 *name* option-argument.
- 29548 **-o path_name** Redefine the path for the standard output of the batch job.
- 29549 The *qalter* utility shall accept a *path_name* option-argument that conforms to the
29550 syntax of the *path_name* element defined in the POSIX.1-1990 standard, which can
29551 be preceded by a host name element of the form *hostname*:
- 29552 If the *path_name* option-argument constitutes an absolute path name, the *qalter*
29553 utility shall set the *Output_Path* attribute of the batch job to the value of the
29554 *path_name* option-argument.
- 29555 If the *path_name* option-argument constitutes a relative path name and no host
29556 name element is specified, the *qalter* utility shall set the *Output_Path* attribute of the

29557 batch job to the absolute path name derived by expanding the *path_name* option-
 29558 argument relative to the current directory of the process that executes the *qalter*
 29559 utility.

29560 If the *path_name* option-argument constitutes a relative path name and a host name
 29561 element is specified, the *qalter* utility shall set the *Output_Path* attribute of the batch
 29562 job to option-argument without any expansion of the path name.

29563 If the *path_name* option-argument does not include a host name element, the *qalter*
 29564 utility shall prefix the path name in the *Output_Path* attribute with *hostname:*,
 29565 where *hostname* is the name of the host upon which the *qalter* utility is being
 29566 executed.

29567 **-p *priority*** Redefine the priority of the batch job.

29568 The *qalter* utility shall accept a value for the priority option-argument that
 29569 conforms to the syntax for signed decimal integers, and which is not less than
 29570 -1 024 and not greater than 1 023.

29571 The *qalter* utility shall set the *Priority* attribute of the batch job to the value of the
 29572 *priority* option-argument.

29573 **-r *y* | *n*** Redefine whether the batch job is rerunable.

29574 If the value of the option-argument is *y*, the *qalter* utility shall set the *Rerunable*
 29575 attribute of the batch job to TRUE.

29576 If the value of the option-argument is *n*, the *qalter* utility shall set the *Rerunable*
 29577 attribute of the batch job to FALSE.

29578 The *qalter* utility shall consider it an error if any character other than 'y' or 'n' is
 29579 specified in the option-argument.

29580 **-S *path_name_list***

29581 Redefine the shell that interprets the script at the destination system.

29582 The *qalter* utility shall accept a *path_name_list* option-argument that conforms to
 29583 the following syntax:

29584 `pathname[@host][,pathname[@host],...]`

29585 The *qalter* utility shall accept only one path name that is missing a corresponding
 29586 host name. The *qalter* utility shall allow only one path name per named host.

29587 The *qalter* utility shall add a value to the *Shell_Path_List* attribute of the batch job
 29588 for each entry in the *path_name_list* option-argument. See Section 3.3.3 (on page
 29589 2337) for a means of removing *keyword=value* (and *value@keyword*) pairs and other
 29590 general rules for list-oriented batch job attributes.

29591 **-u *user_list*** Redefine the user name under which the batch job is to run at the destination
 29592 system.

29593 The *qalter* utility shall accept a *user_list* option-argument that conforms to the
 29594 following syntax:

29595 `username[@host][,username[@host],...]`

29596 The *qalter* utility shall accept only one user name that is missing a corresponding
 29597 host name. The *qalter* utility shall accept only one user name per named host.

29598 The *qalter* utility shall add a value to the *User_List* attribute of the batch job for each
 29599 entry in the *user_list* option-argument. See Section 3.3.3 (on page 2337) for a means

29600 of removing *keyword=value* (and *value@keyword*) pairs and other general rules for
29601 list-oriented batch job attributes.

29602 OPERANDS

29603 The *qalter* utility shall accept one or more operands that conform to the syntax for a batch
29604 *job_identifier* (see Section 3.3.1 (on page 2336)).

29605 STDIN

29606 Not used.

29607 INPUT FILES

29608 None.

29609 ENVIRONMENT VARIABLES

29610 The following environment variables shall affect the execution of *qalter*:

29611 *LANG* Provide a default value for the internationalization variables that are unset or null.
29612 If *LANG* is unset or null, the corresponding value from the implementation-
29613 defined default locale shall be used. If any of the internationalization variables
29614 contains an invalid setting, the utility shall behave as if none of the variables had
29615 been defined.

29616 *LC_ALL* If set to a non-empty string value, override the values of all the other
29617 internationalization variables.

29618 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
29619 characters (for example, single-byte as opposed to multi-byte characters in
29620 arguments).

29621 LC_MESSAGES

29622 Determine the locale that should be used to affect the format and contents of
29623 diagnostic messages written to standard error.

29624 *LC_TIME* Determine the format and contents of date and time strings written by *qalter*.

29625 *LOGNAME* Determine the login name of the user.

29626 *TZ* Determine the timezone in which the time and date are written. If the *TZ* variable
29627 is not set, an unspecified system default timezone is used.

29628 ASYNCHRONOUS EVENTS

29629 Default.

29630 STDOUT

29631 None.

29632 STDERR

29633 Used only for diagnostic messages.

29634 OUTPUT FILES

29635 None.

29636 EXTENDED DESCRIPTION

29637 None.

29638 EXIT STATUS

29639 The following exit values shall be returned:

29640 0 Successful completion.

29641 >0 An error occurred.

29642 CONSEQUENCES OF ERRORS

29643 In addition to the default behavior, the *qalter* utility shall not be required to write a diagnostic
29644 message to standard error when the error reply received from a batch server indicates that the
29645 batch *job_identifier* does not exist on the server. Whether or not the *qalter* utility attempts to
29646 locate the batch job on other batch servers is implementation-defined.

29647 APPLICATION USAGE

29648 None.

29649 EXAMPLES

29650 None.

29651 RATIONALE

29652 The *qalter* utility allows users to change the attributes of a batch job.

29653 As a means of altering a queued job, the *qalter* utility is superior to deleting and requeuing the
29654 batch job insofar as an altered job retains its place in the queue with some traditional selection
29655 algorithms. In addition, the *qalter* utility is both shorter and simpler than a sequence of *qdel* and
29656 *qsub* utilities.

29657 The result of an attempt on the part of a user to alter a batch job in a RUNNING state is
29658 implementation-defined because a batch job in the RUNNING state will already have opened its
29659 output files and otherwise performed any actions indicated by the options in effect at the time
29660 the batch job began execution.

29661 The options processed by the *qalter* utility are identical to those of the *qsub* utility, with a few
29662 exceptions: **-V**, **-v**, and **-q**. The **-V** and **-v** are inappropriate for the *qalter* utility, since they
29663 capture potentially transient environment information from the submitting process. The **-q**
29664 option would specify a new queue, which would largely negate the previously stated advantage
29665 of using *qalter*; furthermore, the *qmove* utility provides a superior means of moving jobs.

29666 Each of the following paragraphs provides the rationale for a *qalter* option.

29667 Additional rationale concerning these options can be found in the rationale for the *qsub* utility.

29668 The **-a** option allows users to alter the date and time at which a batch job becomes eligible to
29669 run.

29670 The **-A** option allows users to change the account that will be charged for the resources
29671 consumed by the batch job. Support for the **-A** option is mandatory for conforming
29672 implementations of *qalter*, even though support of accounting is optional for servers. Whether or
29673 not to support accounting is left to the implementor of the server, but mandatory support of the
29674 **-A** option assures users of a consistent interface and allows them to control accounting on
29675 servers that support accounting.

29676 The **-c** option allows users to alter the checkpointing interval of a batch job. A checkpointing
29677 system, which is not defined by IEEE Std. 1003.1-200x, allows recovery of a batch job at the most
29678 recent checkpoint in the event of a crash. Checkpointing is typically used for jobs that consume
29679 expensive computing time or must meet a critical schedule. Users should be allowed to make
29680 the tradeoff between the overhead of checkpointing and the risk to the timely completion of the
29681 batch job; therefore, this volume of IEEE Std. 1003.1-200x provides the checkpointing interval
29682 option. Support for checkpointing is optional for servers.

29683 The **-e** option allows users to alter the name and location of the standard error stream written by
29684 a batch job. However, the path of the standard error stream is meaningless if the value of the
29685 *Join_Path* attribute of the batch job is TRUE.

29686 The **-h** option allows users to set the hold type in the *Hold_Types* attribute of a batch job. The
29687 *qhold* and *qrls* utilities add or remove hold types to the *Hold_Types* attribute, respectively. The **-h**

- 29688 option has been modified to allow for implementation-defined hold types.
- 29689 The `-j` option allows users to alter the decision to join (merge) the standard error stream of the
29690 batch job with the standard output stream of the batch job.
- 29691 The `-l` option allows users to change the resource limits imposed on a batch job.
- 29692 The `-m` option allows users to modify the list of points in the life of a batch job at which the
29693 designated users will receive mail notification.
- 29694 The `-M` option allows users to alter the list of users who will receive notification about events in
29695 the life of a batch job.
- 29696 The `-N` option allows users to change the name of a batch job.
- 29697 The `-o` option allows users to alter the name and path to which the standard output stream of
29698 the batch job will be written.
- 29699 The `-P` option allows users to modify the priority of a batch job. Support for priority is optional
29700 for batch servers.
- 29701 The `-r` option allows users to alter the rerunability status of a batch job.
- 29702 The `-S` option allows users to change the name and location of the shell image that will be
29703 invoked to interpret the script of the batch job. This option has been modified to allow a list of
29704 shell name and locations associated with different host.
- 29705 The `-u` option allows users to change the user identifier under which the batch job will execute.
- 29706 The `job_identifier` operand syntax is provided so that the user can differentiate between the
29707 originating and destination (or executing) batch server. These may or may not be the same. The
29708 `.server_name` portion identifies the originating batch server, while the `@server` portion identifies
29709 the destination batch server.
- 29710 Historically, the `qalter` utility has been a component of the Network Queuing System (NQS), the
29711 existing practice from which this utility has been derived.
- 29712 **FUTURE DIRECTIONS**
- 29713 None.
- 29714 **SEE ALSO**
- 29715 `qdel`, `qhold`, `qmove`, `qrls`, `qsub`, `touch`, Chapter 3 (on page 2313)
- 29716 **CHANGE HISTORY**
- 29717 Derived from IEEE Std. 1003.2d-1994.

29718 **NAME**

29719 qdel — delete batch jobs

29720 **SYNOPSIS**29721 BE qdel *job_identifier* ...

29722

29723 **DESCRIPTION**29724 A batch job is deleted by sending a request to the batch server that manages the batch job. A
29725 batch job that has been deleted is no longer subject to management by batch services.29726 The *qdel* utility is a user-accessible client of batch services that requests the deletion of one or
29727 more batch jobs.29728 The *qdel* utility shall request a batch server to delete those batch jobs for which a batch
29729 *job_identifier* is presented to the utility.29730 The *qdel* utility shall delete batch jobs in the order in which their batch *job_identifiers* are
29731 presented to the utility.29732 If the *qdel* utility fails to process any batch *job_identifier* successfully, the utility shall proceed to
29733 process the remaining batch *job_identifiers*, if any.29734 The *qdel* utility shall delete each batch job by sending a *Delete Job Request* to the batch server that
29735 manages the batch job.29736 The *qdel* utility shall not exit until the batch job corresponding to each successfully processed
29737 batch *job_identifier* has been deleted.29738 **OPTIONS**

29739 None.

29740 **OPERANDS**29741 The *qdel* utility shall accept one or more operands that conform to the syntax for a batch
29742 *job_identifier* (see Section 3.3.1 (on page 2336)).29743 **STDIN**

29744 Not used.

29745 **INPUT FILES**

29746 None.

29747 **ENVIRONMENT VARIABLES**29748 The following environment variables shall affect the execution of *qdel*:29749 *LANG* Provide a default value for the internationalization variables that are unset or null.
29750 If *LANG* is unset or null, the corresponding value from the implementation-
29751 defined default locale shall be used. If any of the internationalization variables
29752 contains an invalid setting, the utility shall behave as if none of the variables had
29753 been defined.29754 *LC_ALL* If set to a non-empty string value, override the values of all the other
29755 internationalization variables.29756 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
29757 characters (for example, single-byte as opposed to multi-byte characters in
29758 arguments).29759 *LC_MESSAGES*29760 Determine the locale that should be used to affect the format and contents of
29761 diagnostic messages written to standard error.

- 29762 *LC_TIME* Determine the format and contents of date and time strings written by *qdel*.
- 29763 *LOGNAME* Determine the login name of the user.
- 29764 *TZ* Determine the timezone in which the time and date are written. If the *TZ* variable
29765 is not set, an unspecified system default timezone is used.
- 29766 **ASYNCHRONOUS EVENTS**
- 29767 Default.
- 29768 **STDOUT**
- 29769 An implementation of the *qdel* utility may write informative messages to standard output.
- 29770 **STDERR**
- 29771 Used only for diagnostic messages.
- 29772 **OUTPUT FILES**
- 29773 None.
- 29774 **EXTENDED DESCRIPTION**
- 29775 None.
- 29776 **EXIT STATUS**
- 29777 The following exit values shall be returned:
- 29778 0 Successful completion.
- 29779 >0 An error occurred.
- 29780 **CONSEQUENCES OF ERRORS**
- 29781 In addition to the default behavior, the *qdel* utility shall not be required to write a diagnostic
29782 message to standard error when the error reply received from a batch server indicates that the
29783 batch *job_identifier* does not exist on the server. Whether or not the *qdel* utility waits to output the
29784 diagnostic message while attempting to locate the job on other servers is implementation-
29785 defined.
- 29786 **APPLICATION USAGE**
- 29787 None.
- 29788 **EXAMPLES**
- 29789 None.
- 29790 **RATIONALE**
- 29791 The *qdel* utility allows users and administrators to delete jobs.
- 29792 The *qdel* utility provides functionality that is not otherwise available. For example, the *kill* utility
29793 of the operating system does not suffice. First, to use the *kill* utility, the user might have to log in
29794 on a remote node, because the *kill* utility does not operate across the network. Second, unlike
29795 *qdel*, *kill* cannot remove jobs from queues. Lastly, the arguments of the *qdel* utility are job
29796 identifiers rather than process identifiers, and so this utility can be passed the output of the
29797 *qselect* utility, thus providing users with a means of deleting a list of jobs.
- 29798 Because a set of jobs can be selected using the *qselect* utility, the *qdel* utility has not been
29799 complicated with options that provide for selection of jobs. Instead, the batch jobs to be deleted
29800 are identified individually by their job identifiers.
- 29801 Historically, the *qdel* utility has been a component of NQS, the existing practice on which it is
29802 based. However, the *qdel* utility defined in this volume of IEEE Std. 1003.1-200x does not provide
29803 an option for specifying a signal number to send to the batch job prior to the killing of the
29804 process; that capability has been subsumed by the *qsig* utility.

29805 A discussion was held about the delays of networking and the possibility that the batch server
29806 may never respond, due to a down router, down batch server, or other network mishap. The
29807 DESCRIPTION records this under the words “fails to process any job identifier”. In the broad
29808 sense, the network problem is also an error, which causes the failure to process the batch job
29809 identifier.

29810 **FUTURE DIRECTIONS**

29811 None.

29812 **SEE ALSO**

29813 *kill, qselect, qsig*, Chapter 3 (on page 2313)

29814 **CHANGE HISTORY**

29815 Derived from IEEE Std. 1003.2d-1994.

29816 **NAME**

29817 qhold — hold batch jobs

29818 **SYNOPSIS**29819 BE qhold [-h *hold_list*] *job_identifier* ...

29820

29821 **DESCRIPTION**

29822 A hold is placed on a batch job by a request to the batch server that manages the batch job. A
 29823 batch job that has one or more holds is not eligible for execution. The *qhold* utility is a user-
 29824 accessible client of batch services that requests one or more types of hold to be placed on one or
 29825 more batch jobs.

29826 The *qhold* utility shall place holds on those batch jobs for which a batch *job_identifier* is presented
 29827 to the utility.

29828 The *qhold* utility shall place holds on batch jobs in the order in which their batch *job_identifiers*
 29829 are presented to the utility. If the *qhold* utility fails to process any batch *job_identifier* successfully,
 29830 the utility shall proceed to process the remaining batch *job_identifiers*, if any.

29831 The *qhold* utility shall place holds on each batch job by sending a *Hold Job Request* to the batch
 29832 server that manages the batch job.

29833 The *qhold* utility shall not exit until holds have been placed on the batch job corresponding to
 29834 each successfully processed batch *job_identifier*.

29835 **OPTIONS**

29836 The *qhold* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 29837 12.2, Utility Syntax Guidelines.

29838 The following option shall be supported by the implementation:

29839 **-h *hold_list*** Define the types of holds to be placed on the batch job.

29840 The *qhold* **-h** option shall accept a value for the *hold_list* option-argument that is a
 29841 string of alphanumeric characters in the portable character set (see the Base
 29842 Definitions volume of IEEE Std. 1003.1-200x, Section 6.1, Portable Character Set).

29843 The *qhold* utility shall accept a value for the *hold_list* option-argument that is a
 29844 string of one or more of the characters 'u', 's', or 'o', or the single character
 29845 'n'.

29846 For each unique character in the *hold_list* option-argument, the *qhold* utility shall
 29847 add a value to the *Hold_Types* attribute of the batch job as follows, each
 29848 representing a different hold type:

29849 **u** USER

29850 **s** SYSTEM

29851 **o** OPERATOR

29852 If any of these characters are duplicated in the *hold_list* option-argument, the
 29853 duplicates shall be ignored.

29854 An existing *Hold_Types* attribute can be cleared by the following hold type:

29855 **n** NO_HOLD

29856 The *qhold* utility shall consider it an error if any hold type other than **n** is combined
 29857 with hold type **n**.

29858 Strictly conforming applications shall not repeat any of the characters 'u', 's',
29859 'o', or 'n' within the *hold_list* option-argument. The *qhold* utility shall permit the
29860 repetition of characters, but shall not assign additional meaning to the repeated
29861 characters.

29862 An implementation may define other hold types. The conformance document for
29863 an implementation shall describe any additional hold types, how they are
29864 specified, their internal behavior, and how they affect the behavior of the utility.

29865 If the *-h* option is not presented to the *qhold* utility, the implementation shall set
29866 the *Hold_Types* attribute to *USER*.

29867 OPERANDS

29868 The *qhold* utility shall accept one or more operands that conform to the syntax for a batch
29869 *job_identifier* (see Section 3.3.1 (on page 2336)).

29870 STDIN

29871 Not used.

29872 INPUT FILES

29873 None.

29874 ENVIRONMENT VARIABLES

29875 The following environment variables shall affect the execution of *qhold*:

29876 *LANG* Provide a default value for the internationalization variables that are unset or null.
29877 If *LANG* is unset or null, the corresponding value from the implementation-
29878 defined default locale shall be used. If any of the internationalization variables
29879 contains an invalid setting, the utility shall behave as if none of the variables had
29880 been defined.

29881 *LC_ALL* If set to a non-empty string value, override the values of all the other
29882 internationalization variables.

29883 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
29884 characters (for example, single-byte as opposed to multi-byte characters in
29885 arguments).

29886 *LC_MESSAGES*

29887 Determine the locale that should be used to affect the format and contents of
29888 diagnostic messages written to standard error.

29889 *LC_TIME* Determine the format and contents of date and time strings written by *qhold*.

29890 *LOGNAME* Determine the login name of the user.

29891 *TZ* Determine the timezone in which the time and date are written. If the *TZ* variable
29892 is not set, an unspecified system default timezone is used.

29893 ASYNCHRONOUS EVENTS

29894 Default.

29895 STDOUT

29896 None.

29897 STDERR

29898 Used only for diagnostic messages.

29899 **OUTPUT FILES**

29900 None.

29901 **EXTENDED DESCRIPTION**

29902 None.

29903 **EXIT STATUS**

29904 The following exit values shall be returned:

29905 0 Successful completion.

29906 >0 An error occurred.

29907 **CONSEQUENCES OF ERRORS**

29908 In addition to the default behavior, the *qhold* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qhold* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

29913 **APPLICATION USAGE**

29914 None.

29915 **EXAMPLES**

29916 None.

29917 **RATIONALE**

29918 The *qhold* utility allows users to place a hold on one or more jobs. A hold makes a batch job ineligible for execution.

29920 The *qhold* utility has options that allow the user to specify the type of hold. Should the user wish to place a hold on a set of jobs that meet a selection criteria, such a list of jobs can be acquired using the *qselect* utility.

29923 The *-h* option allows the user to specify the type of hold that is to be placed on the job. This option allows for USER, SYSTEM, OPERATOR, and implementation-defined hold types. The USER and OPERATOR holds are distinct. The batch server that manages the batch job will verify that the user is authorized to set the specified hold for the batch job.

29927 Mail is not required on hold because the administrator has the tools and libraries to build this option if he or she wishes.

29929 Historically, the *qhold* utility has been a part of some existing batch systems, although it has not traditionally been a part of the NQS.

29931 **FUTURE DIRECTIONS**

29932 None.

29933 **SEE ALSO**29934 *qselect*, Chapter 3 (on page 2313)29935 **CHANGE HISTORY**

29936 Derived from IEEE Std. 1003.2d-1994.

29937 **NAME**

29938 qmove — move batch jobs

29939 **SYNOPSIS**29940 BE qmove *destination job_identifier* ...

29941

29942 **DESCRIPTION**

29943 To move a batch job is to remove the batch job from the batch queue in which it resides and
 29944 instantiate the batch job in another batch queue. A batch job is moved by a request to the batch
 29945 server that manages the batch job. The *qmove* utility is a user-accessible batch client that requests
 29946 the movement of one or more batch jobs.

29947 The *qmove* utility shall move those batch jobs, and only those batch jobs, for which a batch
 29948 *job_identifier* is presented to the utility.

29949 The *qmove* utility shall move batch jobs in the order in which the corresponding batch
 29950 *job_identifiers* are presented to the utility.

29951 If the *qmove* utility fails to process a batch *job_identifier* successfully, the utility shall proceed to
 29952 process the remaining batch *job_identifiers*, if any.

29953 The *qmove* utility shall move batch jobs by sending a *Move Job Request* to the batch server that
 29954 manages each batch job. The *qmove* utility shall not exit before the batch jobs corresponding to all
 29955 successfully processed batch *job_identifiers* have been moved.

29956 **OPTIONS**

29957 None.

29958 **OPERANDS**

29959 The *qmove* utility shall accept one operand that conforms to the syntax for a *destination* (see
 29960 Section 3.3.2 (on page 2337)).

29961 The *qmove* utility shall accept one or more operands that conform to the syntax for a batch
 29962 *job_identifier* (see Section 3.3.1 (on page 2336)).

29963 **STDIN**

29964 Not used.

29965 **INPUT FILES**

29966 None.

29967 **ENVIRONMENT VARIABLES**29968 The following environment variables shall affect the execution of *qmove*:

29969 *LANG* Provide a default value for the internationalization variables that are unset or null.
 29970 If *LANG* is unset or null, the corresponding value from the implementation-
 29971 defined default locale shall be used. If any of the internationalization variables
 29972 contains an invalid setting, the utility shall behave as if none of the variables had
 29973 been defined.

29974 *LC_ALL* If set to a non-empty string value, override the values of all the other
 29975 internationalization variables.

29976 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 29977 characters (for example, single-byte as opposed to multi-byte characters in
 29978 arguments).

29979 *LC_MESSAGES*

29980 Determine the locale that should be used to affect the format and contents of

- 29981 diagnostic messages written to standard error.
- 29982 *LC_TIME* Determine the format and contents of date and time strings written by *qmove*.
- 29983 *LOGNAME* Determine the login name of the user.
- 29984 *TZ* Determine the timezone in which the time and date are written. If the *TZ* variable
29985 is not set, an unspecified system default timezone is used.
- 29986 **ASYNCHRONOUS EVENTS**
- 29987 Default.
- 29988 **STDOUT**
- 29989 None.
- 29990 **STDERR**
- 29991 Used only for diagnostic messages.
- 29992 **OUTPUT FILES**
- 29993 None.
- 29994 **EXTENDED DESCRIPTION**
- 29995 None.
- 29996 **EXIT STATUS**
- 29997 The following exit values shall be returned:
- 29998 0 Successful completion.
- 29999 >0 An error occurred.
- 30000 **CONSEQUENCES OF ERRORS**
- 30001 In addition to the default behavior, the *qmove* utility shall not be required to write a diagnostic
30002 message to standard error when the error reply received from a batch server indicates that the
30003 batch *job_identifier* does not exist on the server. Whether or not the *qmove* utility waits to output
30004 the diagnostic message while attempting to locate the job on other servers is implementation-
30005 defined.
- 30006 **APPLICATION USAGE**
- 30007 None.
- 30008 **EXAMPLES**
- 30009 None.
- 30010 **RATIONALE**
- 30011 The *qmove* utility allows users to move jobs between queues.
- 30012 The alternative to using the *qmove* utility—deleting the batch job and requeuing it—entails
30013 considerably more typing.
- 30014 Since the means of selecting jobs based on attributes has been encapsulated in the *qselect* utility,
30015 the only option of the *qmove* utility concerns authorization. The *-u* option provides the user with
30016 the convenience of changing the user identifier under which the batch job will execute.
- 30017 Minimalism and consistency has taken precedence over convenience; the *-u* option has been
30018 deleted because the equivalent capability exists with the *-u* option of the *qalter* utility.
- 30019 **FUTURE DIRECTIONS**
- 30020 None.

30021 **SEE ALSO**

30022 *qalter, qselect*, Chapter 3 (on page 2313)

30023 **CHANGE HISTORY**

30024 Derived from IEEE Std. 1003.2d-1994.

30025 **NAME**

30026 qmsg — send message to batch jobs

30027 **SYNOPSIS**30028 BE qmsg [-E][-O] *message_string* *job_identifier* ...

30029

30030 **DESCRIPTION**

30031 To send a message to a batch job is to request that a server write a message string into one or
 30032 more output files of the batch job. A message is sent to a batch job by a request to the batch
 30033 server that manages the batch job. The *qmsg* utility is a user-accessible batch client that requests
 30034 the sending of messages to one or more batch jobs.

30035 The *qmsg* utility shall write messages into the files of batch jobs by sending a *Job Message Request*
 30036 to the batch server that manages the batch job. The *qmsg* utility shall not directly write the
 30037 message into the files of the batch job.

30038 The *qmsg* utility shall send a *Job Message Request* for those batch jobs, and only those batch jobs,
 30039 for which a batch *job_identifier* is presented to the utility.

30040 The *qmsg* utility shall send *Job Message Requests* for batch jobs in the order in which their batch
 30041 *job_identifiers* are presented to the utility.

30042 If the *qmsg* utility fails to process any batch *job_identifier* successfully, the utility shall proceed to
 30043 process the remaining batch *job_identifiers*, if any.

30044 The *qmsg* utility shall not exit before a *Job Message Request* has been sent to the server that
 30045 manages the batch job that corresponds to each successfully processed batch *job_identifier*.

30046 **OPTIONS**

30047 The *qmsg* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 30048 12.2, Utility Syntax Guidelines.

30049 The following options shall be supported by the implementation:

30050 **-E** Specify that the message is written to the standard error of each batch job.

30051 The *qmsg* utility shall write the message into the standard error of the batch job.

30052 **-O** Specify that the message is written to the standard output of each batch job.

30053 The *qmsg* utility shall write the message into the standard output of the batch job.

30054 If neither the **-O** nor the **-E** option is presented to the *qmsg* utility, the utility shall write the
 30055 message into an implementation-defined file. The conformance document for the
 30056 implementation shall describe the name and location of the implementation-defined file. If both
 30057 the **-O** and the **-E** options are presented to the *qmsg* utility, then the utility shall write the
 30058 messages to both standard output and standard error.

30059 **OPERANDS**

30060 The *qmsg* utility shall accept a minimum of two operands, *message_string* and one or more batch
 30061 *job_identifiers*.

30062 The *message_string* operand shall be the string to be written to one or more output files of the
 30063 batch job followed by a <newline>. If the string contains <blank>s, then the application shall
 30064 ensure that the string is quoted. The *message_string* shall be encoded in the portable character set
 30065 (see the Base Definitions volume of IEEE Std. 1003.1-200x, Section 6.1, Portable Character Set).

30066 All remaining operands are batch *job_identifiers* that conform to the syntax for a batch
 30067 *job_identifier* (see Section 3.3.1 (on page 2336)).

30068 **STDIN**

30069 Not used.

30070 **INPUT FILES**

30071 None.

30072 **ENVIRONMENT VARIABLES**30073 The following environment variables shall affect the execution of *qmsg*:

30074 *LANG* Provide a default value for the internationalization variables that are unset or null.
30075 If *LANG* is unset or null, the corresponding value from the implementation-
30076 defined default locale shall be used. If any of the internationalization variables
30077 contains an invalid setting, the utility shall behave as if none of the variables had
30078 been defined.

30079 *LC_ALL* If set to a non-empty string value, override the values of all the other
30080 internationalization variables.

30081 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
30082 characters (for example, single-byte as opposed to multi-byte characters in
30083 arguments).

30084 *LC_MESSAGES*
30085 Determine the locale that should be used to affect the format and contents of
30086 diagnostic messages written to standard error.

30087 *LC_TIME* Determine the format and contents of date and time strings written by *qmsg*.

30088 *LOGNAME* Determine the login name of the user.

30089 *TZ* Determine the timezone in which the time and date are written. If the *TZ* variable
30090 is not set, an unspecified system default timezone is used.

30091 **ASYNCHRONOUS EVENTS**

30092 Default.

30093 **STDOUT**

30094 None.

30095 **STDERR**

30096 Used only for diagnostic messages.

30097 **OUTPUT FILES**

30098 None.

30099 **EXTENDED DESCRIPTION**

30100 None.

30101 **EXIT STATUS**

30102 The following exit values shall be returned:

30103 0 Successful completion.

30104 >0 An error occurred.

30105 **CONSEQUENCES OF ERRORS**

30106 In addition to the default behavior, the *qmsg* utility shall not be required to write a diagnostic
30107 message to standard error when the error reply received from a batch server indicates that the
30108 batch *job_identifier* does not exist on the server. Whether or not the *qmsg* utility waits to output
30109 the diagnostic message while attempting to locate the job on other servers is implementation-
30110 defined.

30111 APPLICATION USAGE

30112 None.

30113 EXAMPLES

30114 None.

30115 RATIONALE

30116 The *qmsg* utility allows users to write messages into the output files of running jobs. Users,
30117 including operators and administrators, have a number of occasions when they want to place
30118 messages in the output files of a batch job. For example, if a disk that is being used by a batch job
30119 is showing errors, the operator might note this in the standard error stream of the batch job.

30120 The options of the *qmsg* utility provide users with the means of placing the message in the
30121 output stream of their choice. The default output stream for the message—if the user does not
30122 designate an output stream—is implementation-defined, since many implementations will
30123 provide, as an extension to this volume of IEEE Std. 1003.1-200x, a log file that shows the history
30124 of utility execution.

30125 If users wish to send a message to a set of jobs that meet a selection criteria, the *qselect* utility can
30126 be used to acquire the appropriate list of job identifiers.

30127 The **-E** option allows users to place the message in the standard error stream of the batch job.

30128 The **-O** option allows users to place the message in the standard output stream of the batch job.

30129 Historically, the *qmsg* utility is an existing practice in the offerings of one or more implementors
30130 of an NQS-derived batch system. The utility has been found to be useful enough that it deserves
30131 to be included in this volume of IEEE Std. 1003.1-200x.

30132 FUTURE DIRECTIONS

30133 None.

30134 SEE ALSO

30135 *qselect*, Chapter 3 (on page 2313)

30136 CHANGE HISTORY

30137 Derived from IEEE Std. 1003.2d-1994.

30138 **NAME**

30139 qrerun — rerun batch jobs

30140 **SYNOPSIS**30141 BE `qrerun job_identifier ...`

30142

30143 **DESCRIPTION**

30144 To rerun a batch job is to terminate the session leader of the batch job, delete any associated
 30145 checkpoint files, and return the batch job to the batch queued state. A batch job is rerun by a
 30146 request to the batch server that manages the batch job. The *qrerun* utility is a user-accessible
 30147 batch client that requests the rerunning of one or more batch jobs.

30148 The *qrerun* utility shall rerun those batch jobs for which a batch *job_identifier* is presented to the
 30149 utility.

30150 The *qrerun* utility shall rerun batch jobs in the order in which their batch *job_identifiers* are
 30151 presented to the utility.

30152 If the *qrerun* utility fails to process any batch *job_identifier* successfully, the utility shall proceed
 30153 to process the remaining batch *job_identifiers*, if any.

30154 The *qrerun* utility shall rerun batch jobs by sending a *Rerun Job Request* to the batch server that
 30155 manages each batch job.

30156 For each successfully processed batch *job_identifier*, the *qrerun* utility shall have rerun the
 30157 corresponding batch batch job at the time the utility exits.

30158 **OPTIONS**

30159 None.

30160 **OPERANDS**

30161 The *qrerun* utility shall accept one or more operands that conform to the syntax for a batch
 30162 *job_identifier* (see Section 3.3.1 (on page 2336)).

30163 **STDIN**

30164 Not used.

30165 **INPUT FILES**

30166 None.

30167 **ENVIRONMENT VARIABLES**30168 The following environment variables shall affect the execution of *qrerun*:

30169 *LANG* Provide a default value for the internationalization variables that are unset or null.
 30170 If *LANG* is unset or null, the corresponding value from the implementation-
 30171 defined default locale shall be used. If any of the internationalization variables
 30172 contains an invalid setting, the utility shall behave as if none of the variables had
 30173 been defined.

30174 *LC_ALL* If set to a non-empty string value, override the values of all the other
 30175 internationalization variables.

30176 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 30177 characters (for example, single-byte as opposed to multi-byte characters in
 30178 arguments).

30179 *LC_MESSAGES*

30180 Determine the locale that should be used to affect the format and contents of
 30181 diagnostic messages written to standard error.

- 30182 *LC_TIME* Determine the format and contents of date and time strings written by *qrerun*.
- 30183 *LOGNAME* Determine the login name of the user.
- 30184 *TZ* Determine the timezone in which the time and date are written. If the *TZ* variable
30185 is not set, an unspecified system default timezone is used.
- 30186 **ASYNCHRONOUS EVENTS**
- 30187 Default.
- 30188 **STDOUT**
- 30189 None.
- 30190 **STDERR**
- 30191 Used only for diagnostic messages.
- 30192 **OUTPUT FILES**
- 30193 None.
- 30194 **EXTENDED DESCRIPTION**
- 30195 None.
- 30196 **EXIT STATUS**
- 30197 The following exit values shall be returned:
- 30198 0 Successful completion.
- 30199 >0 An error occurred.
- 30200 **CONSEQUENCES OF ERRORS**
- 30201 In addition to the default behavior, the *qrerun* utility shall not be required to write a diagnostic
30202 message to standard error when the error reply received from a batch server indicates that the
30203 batch *job_identifier* does not exist on the server. Whether or not the *qrerun* utility waits to output
30204 the diagnostic message while attempting to locate the job on other servers is implementation-
30205 defined.
- 30206 **APPLICATION USAGE**
- 30207 None.
- 30208 **EXAMPLES**
- 30209 None.
- 30210 **RATIONALE**
- 30211 The *qrerun* utility allows users to cause jobs in the running state to exit and rerun.
- 30212 The *qrerun* utility is a new utility, *vis-a-vis* existing practice, that has been defined in this volume
30213 of IEEE Std. 1003.1-200x to correct user-perceived deficiencies in the existing practice.
- 30214 **FUTURE DIRECTIONS**
- 30215 None.
- 30216 **SEE ALSO**
- 30217 Chapter 3 (on page 2313)
- 30218 **CHANGE HISTORY**
- 30219 Derived from IEEE Std. 1003.2d-1994.

30220 NAME

30221 qrls — release batch jobs

30222 SYNOPSIS

30223 BE qrls [-h *hold_list*] *job_identifier* ...

30224

30225 DESCRIPTION

30226 A batch job might have one or more holds, which prevent the batch job from executing. A batch
 30227 job from which all the holds have been removed becomes eligible for execution and is said to
 30228 have been released. A batch job hold is removed by sending a request to the batch server that
 30229 manages the batch job. The *qrls* utility is a user-accessible client of batch services that requests
 30230 holds be removed from one or more batch jobs.

30231 The *qrls* utility shall remove one or more holds from those batch jobs for which a batch
 30232 *job_identifier* is presented to the utility.

30233 The *qrls* utility shall remove holds from batch jobs in the order in which their batch *job_identifiers*
 30234 are presented to the utility.

30235 If the *qrls* utility fails to process a batch *job_identifier* successfully, the utility shall proceed to
 30236 process the remaining batch *job_identifiers*, if any.

30237 The *qrls* utility shall remove holds on each batch job by sending a *Release Job Request* to the batch
 30238 server that manages the batch job.

30239 The *qrls* utility shall not exit until the holds have been removed from the batch job
 30240 corresponding to each successfully processed batch *job_identifier*.

30241 OPTIONS

30242 The *qrls* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 30243 12.2, Utility Syntax Guidelines.

30244 The following option shall be supported by the implementation:

30245 **-h *hold_list*** Define the types of holds to be removed from the batch job.

30246 The *qrls* **-h** option shall accept a value for the *hold_list* option-argument that is a
 30247 string of alphanumeric characters in the portable character set (see the Base
 30248 Definitions volume of IEEE Std. 1003.1-200x, Section 6.1, Portable Character Set).

30249 The *qrls* utility shall accept a value for the *hold_list* option-argument that is a string
 30250 of one or more of the characters 'u', 's', or 'o', or the single character 'n'.

30251 For each unique character in the *hold_list* option-argument, the *qrls* utility shall add
 30252 a value to the *Hold_Types* attribute of the batch job as follows, each representing a
 30253 different hold type:

30254 **u** USER

30255 **s** SYSTEM

30256 **o** OPERATOR

30257 If any of these characters are duplicated in the *hold_list* option-argument, the
 30258 duplicates shall be ignored.

30259 An existing *Hold_Types* attribute can be cleared by the following hold type:

30260 **n** NO_HOLD

- 30261 The *qrls* utility shall consider it an error if any hold type other than **n** is combined
30262 with hold type **n**.
- 30263 Strictly conforming applications shall not repeat any of the characters 'u', 's',
30264 'o', or 'n' within the *hold_list* option-argument. The *qrls* utility shall permit the
30265 repetition of characters, but shall not assign additional meaning to the repeated
30266 characters.
- 30267 An implementation may define other hold types. The conformance document for
30268 an implementation shall describe any additional hold types, how they are
30269 specified, their internal behavior, and how they affect the behavior of the utility.
- 30270 If the **-h** option is not presented to the *qrls* utility, the implementation shall remove
30271 the **USER** hold in the *Hold_Types* attribute.
- 30272 **OPERANDS**
- 30273 The *qrls* utility shall accept one or more operands that conform to the syntax for a batch
30274 *job_identifier* (see Section 3.3.1 (on page 2336)).
- 30275 **STDIN**
- 30276 Not used.
- 30277 **INPUT FILES**
- 30278 None.
- 30279 **ENVIRONMENT VARIABLES**
- 30280 The following environment variables shall affect the execution of *qrls*:
- 30281 *LANG* Provide a default value for the internationalization variables that are unset or null.
30282 If *LANG* is unset or null, the corresponding value from the implementation-
30283 defined default locale shall be used. If any of the internationalization variables
30284 contains an invalid setting, the utility shall behave as if none of the variables had
30285 been defined.
- 30286 *LC_ALL* If set to a non-empty string value, override the values of all the other
30287 internationalization variables.
- 30288 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
30289 characters (for example, single-byte as opposed to multi-byte characters in
30290 arguments).
- 30291 *LC_MESSAGES*
- 30292 Determine the locale that should be used to affect the format and contents of
30293 diagnostic messages written to standard error.
- 30294 *LC_TIME* Determine the format and contents of date and time strings written by *qrls*.
- 30295 *LOGNAME* Determine the login name of the user.
- 30296 *TZ* Determine the timezone in which the time and date are written. If the *TZ* variable
30297 is not set, an unspecified system default timezone is used.
- 30298 **ASYNCHRONOUS EVENTS**
- 30299 Default.
- 30300 **STDOUT**
- 30301 None.

30302 **STDERR**

30303 Used only for diagnostic messages.

30304 **OUTPUT FILES**

30305 None.

30306 **EXTENDED DESCRIPTION**

30307 None.

30308 **EXIT STATUS**

30309 The following exit values shall be returned:

30310 0 Successful completion.

30311 >0 An error occurred.

30312 **CONSEQUENCES OF ERRORS**

30313 In addition to the default behavior, the *qrls* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qrls* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

30318 **APPLICATION USAGE**

30319 None.

30320 **EXAMPLES**

30321 None.

30322 **RATIONALE**

30323 The *qrls* utility allows users, operators, and administrators to remove holds from jobs.

30324 The *qrls* utility does not support any job selection options or wildcard arguments. Users may acquire a list of jobs selected by attributes using the *qselect* utility. For example, a user could select all of their held jobs.

30327 The *-h* option allows the user to specify the type of hold that is to be removed. This option allows for USER, SYSTEM, OPERATOR, and implementation-defined hold types. The batch server that manages the batch job will verify whether the user is authorized to remove the specified hold for the batch job. If more than one type of hold has been placed on the batch job, a user may wish to remove only some of them.

30332 Mail is not required on release because the administrator has the tools and libraries to build this option if required.

30334 The *qrls* utility is a new utility *vis-a-vis* existing practice; it has been defined in this volume of IEEE Std. 1003.1-200x as the natural complement to the *qhold* utility.

30336 **FUTURE DIRECTIONS**

30337 None.

30338 **SEE ALSO**

30339 *qhold*, *qselect*, Chapter 3 (on page 2313)

30340 **CHANGE HISTORY**

30341 Derived from IEEE Std. 1003.2d-1994.

30342 NAME

30343 qselect — select batch jobs

30344 SYNOPSIS

```
30345 BE qselect [-a [op]date_time][-A account_string][-c [op]interval]
30346 [-h hold_list][-l resource_list][-N name][-p [op]priority]
30347 [-q destination][-r y|n][-s states][-u user_list]
30348
```

30349 DESCRIPTION

30350 To select a set of batch jobs is to return the batch *job_identifiers* for each batch job that meets a list
 30351 of selection criteria. A set of batch jobs is selected by a request to a batch server. The *qselect*
 30352 utility is a user-accessible batch client that requests the selection of batch jobs.

30353 Upon successful completion, the *qselect* utility shall have returned a list of zero or more batch
 30354 *job_identifiers* that meet the criteria specified by the options and option-arguments presented to
 30355 the utility.

30356 The *qselect* utility shall select batch jobs by sending a *Select Jobs Request* to a batch server. The
 30357 *qselect* utility shall not exit until the server replies to each request generated.

30358 For each option presented to the *qselect* utility, the utility shall restrict the set of selected batch
 30359 jobs as described in the OPTIONS section.

30360 The *qselect* utility shall not restrict selection of batch jobs except by authorization and as required
 30361 by the options presented to the utility.

30362 When an option is specified with a mandatory or optional *op* component to the option-
 30363 argument, then *op* shall specify a relation between the value of a certain batch job attribute and
 30364 the *value* component of the option-argument. If an *op* is allowable on an option, then the
 30365 description of the option letter indicates the *op* as either mandatory or optional. Acceptable
 30366 strings for the *op* component, and the relation the string indicates, are shown in the following
 30367 list:

30368 .eq. The value represented by the attribute of the batch job is equal to the value represented
 30369 by the option-argument.

30370 .ge. The value represented by the attribute of the batch job is greater than or equal to the
 30371 value represented by the option-argument.

30372 .gt. The value represented by the attribute of the batch job is greater than the value
 30373 represented by the option-argument.

30374 .lt. The value represented by the attribute of the batch job is less than the value
 30375 represented by the option-argument.

30376 .le. The value represented by the attribute of the batch job is less than or equal to the value
 30377 represented by the option-argument.

30378 .ne. The value represented by the attribute of the batch job is not equal to the value
 30379 represented by the option-argument.

30380 OPTIONS

30381 The *qselect* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 30382 12.2, Utility Syntax Guidelines.

30383 The following options shall be supported by the implementation:

30384 **-a [op]date_time**

30385 Restrict selection to a specific time, or a range of times.

- 30386 The *qselect* utility shall select only batch jobs for which the value of the
30387 *Execution_Time* attribute is related to the Epoch equivalent of the local time
30388 expressed by the value of the *date_time* component of the option-argument in the
30389 manner indicated by the value of the *op* component of the option-argument.
- 30390 The *qselect* utility shall accept a *date_time* component of the option-argument that
30391 conforms to the syntax of the *date_time* operand of the *touch* utility.
- 30392 If the *op* component of the option-argument is not presented to the *qselect* utility,
30393 the utility shall select batch jobs for which the *Execution_Time* attribute is equal to
30394 the *date_time* component of the option-argument.
- 30395 When comparing times, the *qselect* utility shall use the following definitions for the
30396 *op* component of the option-argument:
- 30397 .eq. The time represented by value of the *Execution_Time* attribute of the batch
30398 job is equal the time represented by the *date_time* component of the
30399 option-argument.
 - 30400 .ge. The time represented by value of the *Execution_Time* attribute of the batch
30401 job is after or equal to the time represented by the *date_time* component of
30402 the option-argument.
 - 30403 .gt. The time represented by value of the *Execution_Time* attribute of the batch
30404 job is after the time represented by the *date_time* component of the
30405 option-argument.
 - 30406 .lt. The time represented by value of the *Execution_Time* attribute of the batch
30407 job is before the time represented by the *date_time* component of the
30408 option-argument.
 - 30409 .le. The time represented by value of the *Execution_Time* attribute of the batch
30410 job is before or equal to the time represented by the *date_time* component
30411 of the option-argument.
 - 30412 .ne. The time represented by value of the *Execution_Time* attribute of the batch
30413 job is not equal to the time represented by the *date_time* component of the
30414 option-argument.
- 30415 The *qselect* utility shall accept the defined character strings for the *op* component of
30416 the option-argument.
- 30417 **-A** *account_string*
30418 Restrict selection to the batch jobs charging a specified account.
- 30419 The *qselect* utility shall select only batch jobs for which the value of the
30420 *Account_Name* attribute of the batch job matches the value of the *account_string*
30421 option-argument.
- 30422 The syntax of the *account_string* option-argument is unspecified.
- 30423 **-c** [*op*]*interval*
30424 Restrict selection to batch jobs within a range of checkpoint intervals.
- 30425 The *qselect* utility shall select only batch jobs for which the value of the *Checkpoint*
30426 attribute relates to the value of the *interval* component of the option-argument in
30427 the manner indicated by the value of the *op* component of the option-argument.
- 30428 If the *op* component of the option-argument is omitted, the *qselect* utility shall
30429 select batch jobs for which the value of the *Checkpoint* attribute is equal to the value

30430 of the *interval* component of the option-argument.

30431 When comparing checkpoint intervals, the *qselect* utility shall use the following
30432 definitions for the *op* component of the option-argument:

30433 .eq. The value of the *Checkpoint* attribute of the batch job equals the value of
30434 the *interval* component of the option-argument.

30435 .ge. The value of the *Checkpoint* attribute of the batch job is greater than or
30436 equal to the value of the *interval* component option-argument.

30437 .gt. The value of the *Checkpoint* attribute of the batch job is greater than the
30438 value of the *interval* component option-argument.

30439 .lt. The value of the *Checkpoint* attribute of the batch job is less than the value
30440 of the *interval* component option-argument.

30441 .le. The value of the *Checkpoint* attribute of the batch job is less than or equal
30442 to the value of the *interval* component option-argument.

30443 .ne. The value of the *Checkpoint* attribute of the batch job does not equal the
30444 value of the *interval* component option-argument.

30445 The *qselect* utility shall accept the defined character strings for the *op* component of
30446 the option-argument.

30447 The ordering relationship for the values of the interval option-argument is defined
30448 to be:

30449 'n' .gt. 's' .gt. 'c=minutes' .ge. 'c'

30450 When comparing *Checkpoint* attributes with an interval having the value of the
30451 single character 'u', only equality or inequality are valid comparisons.

30452 **-h hold_list** Restrict selection to batch jobs that have a specific type of hold.

30453 The *qselect* utility shall select only batch jobs for which the value of the *Hold_Types*
30454 attribute matches the value of the *hold_list* option-argument.

30455 The *qselect* **-h** option shall accept a value for the *hold_list* option-argument that is a
30456 string of alphanumeric characters in the portable character set (see the Base
30457 Definitions volume of IEEE Std. 1003.1-200x, Section 6.1, Portable Character Set).

30458 The *qselect* utility shall accept a value for the *hold_list* option-argument that is a
30459 string of one or more of the characters 'u', 's', or 'o', or the single character
30460 'n'.

30461 Each unique character in the *hold_list* option-argument of the *qselect* utility is
30462 defined as follows, each representing a different hold type:

30463 **u** USER

30464 **s** SYSTEM

30465 **o** OPERATOR

30466 If any of these characters are duplicated in the *hold_list* option-argument, the
30467 duplicates shall be ignored.

30468 The *qselect* utility shall consider it an error if any hold type other than **n** is
30469 combined with hold type **n**.

30470 Strictly conforming applications shall not repeat any of the characters 'u', 's',
 30471 'o', or 'n' within the *hold_list* option-argument. The *qselect* utility shall permit
 30472 the repetition of characters, but shall not assign additional meaning to the repeated
 30473 characters.

30474 An implementation may define other hold types. The conformance document for
 30475 an implementation shall describe any additional hold types, how they are
 30476 specified, their internal behavior, and how they affect the behavior of the utility.

30477 **-I *resource_list***
 30478 Restrict selection to batch jobs with specified resource limits and attributes.

30479 The *qselect* utility shall accept a *resource_list* option-argument with the following
 30480 syntax:

30481 *resource_name op value [, , resource_name op value , , ...]*

30482 When comparing resource values, the *qselect* utility shall use the following
 30483 definitions for the *op* component of the option-argument:

30484 *.eq.* The value of the resource of the same name in the *Resource_List* attribute
 30485 of the batch job equals the value of the value component of the option-
 30486 argument.

30487 *.ge.* The value of the resource of the same name in the *Resource_List* attribute
 30488 of the batch job is greater than or equal to the value of the *value*
 30489 component of the option-argument.

30490 *.gt.* The value of the resource of the same name in the *Resource_List* attribute
 30491 of the batch job is greater than the value of the value component of the
 30492 option-argument.

30493 *.lt.* The value of the resource of the same name in the *Resource_List* attribute
 30494 of the batch job is less than the value of the value component of the
 30495 option-argument.

30496 *.ne.* The value of the resource of the same name in the *Resource_List* attribute
 30497 of the batch job does not equal the value of the value component of the
 30498 option-argument.

30499 *.le.* The value of the resource of the same name in the *Resource_List* attribute
 30500 of the batch job is less than or equal to the value of the *value* component
 30501 of the option-argument.

30502 When comparing the limit of a *Resource_List* attribute with the *value* component of
 30503 the option-argument, if the limit, the value, or both are non-numeric, only equality
 30504 or inequality are valid comparisons.

30505 The *qselect* utility shall select only batch jobs for which the values of the
 30506 *resource_names* listed in the *resource_list* option-argument match the corresponding
 30507 limits of the *Resource_List* attribute of the batch job.

30508 Limits of *resource_names* present in the *Resource_List* attribute of the batch job that
 30509 have no corresponding values in the *resource_list* option-argument shall not be
 30510 considered when selecting batch jobs.

30511 **-N *name*** Restrict selection to batch jobs with a specified name.

30512 The *qselect* utility shall select only batch jobs for which the value of the *Job_Name*
 30513 attribute matches the value of the *name* option-argument. The string specified in

- 30514 the *name* option-argument shall be passed, uninterpreted, to the server. This allows
30515 an implementation to match “wildcard” patterns against batch job names.
- 30516 An implementation shall describe in the conformance document the format it
30517 supports for matching against the *Job_Name* attribute.
- 30518 **-p [op]priority**
- 30519 Restrict selection to batch jobs of the specified priority or range of priorities.
- 30520 The *qselect* utility shall select only batch jobs for which the value of the *Priority*
30521 attribute of the batch job relates to the value of the *priority* component of the
30522 option-argument in the manner indicated by the value of the *op* component of the
30523 option-argument.
- 30524 If the *op* component of the option-argument is omitted, the *qselect* utility shall
30525 select batch jobs for which the value of the *Priority* attribute of the batch job is
30526 equal to the value of the *priority* component of the option-argument.
- 30527 When comparing priority values, the *qselect* utility shall use the following
30528 definitions for the *op* component of the option-argument:
- 30529 .eq. The value of the *Priority* attribute of the batch job equals the value of the
30530 *priority* component of the option-argument.
- 30531 .ge. The value of the *Priority* attribute of the batch job is greater than or equal
30532 to the value of the *priority* component option-argument.
- 30533 .gt. The value of the *Priority* attribute of the batch job is greater than the value
30534 of the *priority* component option-argument.
- 30535 .lt. The value of the *Priority* attribute of the batch job is less than the value of
30536 the *priority* component option-argument.
- 30537 .lte. The value of the *Priority* attribute of the batch job is less than or equal to
30538 the value of the *priority* component option-argument.
- 30539 .ne. The value of the *Priority* attribute of the batch job does not equal the value
30540 of the *priority* component option-argument.
- 30541 **-q destination**
- 30542 Restrict selection to the specified batch queue or server, or both.
- 30543 The *qselect* utility shall select only batch jobs that are located at the destination
30544 indicated by the value of the *destination* option-argument.
- 30545 The destination defines a batch queue, a server, or a batch queue at a server.
- 30546 The *qselect* utility shall accept an option-argument for the **-q** option that conforms
30547 to the syntax for a destination. If the **-q** option is not presented to the *qselect* utility,
30548 the utility shall select batch jobs from all batch queues at the default batch server.
- 30549 If the option-argument describes only a batch queue, the *qselect* utility shall select
30550 only batch jobs from the batch queue of the specified name at the default batch
30551 server. The means by which *qselect* determines the default server is
30552 implementation-defined.
- 30553 If the option-argument describes only a batch server, the *qselect* utility shall select
30554 batch jobs from all the batch queues at that batch server.
- 30555 If the option-argument describes both a batch queue and a batch server, the *qselect*
30556 utility shall select only batch jobs from the specified batch queue at the specified

30557		server.
30558	-r y n	Restrict selection to batch jobs with the specified rerunability status.
30559		The <i>qselect</i> utility shall select only batch jobs for which the value of the <i>Rerunable</i>
30560		attribute of the batch job matches the value of the option-argument.
30561		The <i>qselect</i> utility shall accept a value for the option-argument that consists of
30562		either the single character 'y' or the single character 'n'. The character 'y'
30563		represents the value TRUE, and the character 'n' represents the value FALSE.
30564	-s states	Restrict selection to batch jobs in the specified states.
30565		The <i>qselect</i> utility shall accept an option-argument that consists of any combination
30566		of the characters 'e', 'q', 'r', 'w', 'h', and 't'.
30567		Conforming applications shall not repeat any character in the option-argument.
30568		The <i>qselect</i> utility shall permit the repetition of characters in the option-argument,
30569		but shall not assign additional meaning to repeated characters.
30570		The <i>qselect</i> utility shall interpret the characters in the <i>states</i> option-argument as
30571		follows:
30572	e	Represents the EXITING state.
30573	q	Represents the QUEUED state.
30574	r	Represents the RUNNING state.
30575	t	Represents the TRANSITING state.
30576	h	Represents the HELD state.
30577	w	Represents the WAITING state.
30578		For each character in the <i>states</i> option-argument, the <i>qselect</i> utility shall select batch
30579		jobs in the corresponding state.
30580	-u user_list	Restrict selection to batch jobs owned by the specified user names.
30581		The <i>qselect</i> utility shall select only the batch jobs of those users specified in the
30582		<i>user_list</i> option-argument.
30583		The <i>qselect</i> utility shall accept a <i>user_list</i> option-argument that conforms to the
30584		following syntax:
30585		<i>username</i> [@ <i>host</i>][, , <i>username</i> [@ <i>host</i>], , . . .]
30586		The <i>qselect</i> utility shall accept only one user name that is missing a corresponding
30587		host name. The <i>qselect</i> utility shall accept only one user name per named host.
30588	OPERANDS	
30589		None.
30590	STDIN	
30591		Not used.
30592	INPUT FILES	
30593		None.

30594 **ENVIRONMENT VARIABLES**

30595 The following environment variables shall affect the execution of *qselect*:

30596 *LANG* Provide a default value for the internationalization variables that are unset or null.
 30597 If *LANG* is unset or null, the corresponding value from the implementation-
 30598 defined default locale shall be used. If any of the internationalization variables
 30599 contains an invalid setting, the utility shall behave as if none of the variables had
 30600 been defined.

30601 *LC_ALL* If set to a non-empty string value, override the values of all the other
 30602 internationalization variables.

30603 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 30604 characters (for example, single-byte as opposed to multi-byte characters in
 30605 arguments).

30606 *LC_MESSAGES*
 30607 Determine the locale that should be used to affect the format and contents of
 30608 diagnostic messages written to standard error.

30609 *LC_TIME* Determine the format and contents of date and time strings written by *qselect*.

30610 *LOGNAME* Determine the login name of the user.

30611 *TZ* Determine the timezone in which the time and date are written. If the *TZ* variable
 30612 is not set, an unspecified system default timezone is used.

30613 **ASYNCHRONOUS EVENTS**

30614 Default.

30615 **STDOUT**

30616 The *qselect* utility shall write zero or more batch *job_identifiers* to standard output.

30617 The *qselect* utility shall separate the batch *job_identifiers* written to standard output by white
 30618 space.

30619 The *qselect* utility shall write batch *job_identifiers* in the following format:

30620 *sequence_number.server_name@server*

30621 **STDERR**

30622 Used only for diagnostic messages.

30623 **OUTPUT FILES**

30624 None.

30625 **EXTENDED DESCRIPTION**

30626 None.

30627 **EXIT STATUS**

30628 The following exit values shall be returned:

30629 0 Successful completion.

30630 >0 An error occurred.

30631 **CONSEQUENCES OF ERRORS**

30632 Default.

30633 **APPLICATION USAGE**

30634 None.

30635 **EXAMPLES**

30636 The following example shows how a user might use the *qselect* utility in conjunction with the
 30637 *qdel* utility to delete all of his or her jobs in the queued state without affecting any jobs that are
 30638 already running:

30639 `qdel $(qselect -s q)`

30640 or:

30641 `qselect -s q || xargs qdel`30642 **RATIONALE**

30643 The *qselect* utility allows users to acquire a list of job identifiers that match user-specified
 30644 selection criteria. The list of identifiers returned by the *qselect* utility conforms to the syntax of
 30645 the batch job identifier list processed by a utility such as *qmove*, *qdel*, and *qrls*. The *qselect* utility is
 30646 thus a powerful tool for causing another batch system utility to act upon a set of jobs that match
 30647 a list of selection criteria.

30648 The options of the *qselect* utility let the user apply a number of useful filters for selecting jobs.
 30649 Each option further restricts the selection of jobs. Many of the selection options allow the
 30650 specification of a relational operator. The FORTRAN-like syntax of the operator—that is,
 30651 ".lt.", was chosen rather than the C-like "<=" meta-characters.

30652 The *-a* option allows users to restrict the selected jobs to those that have been submitted (or
 30653 altered) to wait until a particular time. The time period is determined by the argument of this
 30654 option, which includes both a time and an operator—it is thus possible to select jobs waiting
 30655 until a specific time, jobs waiting until after a certain time, or those waiting for a time before the
 30656 specified time.

30657 The *-A* option allows users to restrict the selected jobs to those that have been submitted (or
 30658 altered) to charge a particular account.

30659 The *-c* option allows users to restrict the selected jobs to those whose checkpointing interval
 30660 falls within the specified range.

30661 The *-l* option allows users to select those jobs whose resource limits fall within the range
 30662 indicated by the value of the option. For example, a user could select those jobs for which the
 30663 CPU time limit is greater than two hours.

30664 The *-N* option allows users to select jobs by job name. For instance, all the parts of a task that
 30665 have been divided in parallel jobs might be given the same name, and thus manipulated as a
 30666 group by means of this option.

30667 The *-q* option allows users to select jobs in a specified queue.

30668 The *-r* option allows users to select only those jobs with a specified rerun criteria. For instance, a
 30669 user might select only those jobs that can be rerun for use with the *qrerun* utility.

30670 The *-s* option allows users to select only those jobs that are in a certain state.

30671 The *-u* option allows users to select jobs that have been submitted to execute under a particular
 30672 account.

30673 The selection criteria provided by the options of the *qselect* utility allow users to select jobs based
 30674 on all the appropriate attributes that can be assigned to jobs by the *qsub* utility. When
 30675 implementors extend the *qsub* utility, or another utilities, using the *-W* option, they may likewise
 30676 elect to extend the *qselect* utility to allow additional selection criteria.

30677 Historically, the *qselect* utility has not been a part of existing practice; it is an improvement that
30678 has been introduced in this volume of IEEE Std. 1003.1-200x.

30679 **FUTURE DIRECTIONS**

30680 None.

30681 **SEE ALSO**

30682 *qdel*, *qrerun*, *qrls*, *qselect*, *qsub*, *touch*, Chapter 3 (on page 2313)

30683 **CHANGE HISTORY**

30684 Derived from IEEE Std. 1003.2d-1994.

30685 NAME

30686 qsig — signal batch jobs

30687 SYNOPSIS

30688 BE qsig [-s *signal*] *job_identifier* ...

30689

30690 DESCRIPTION

30691 To signal a batch job is to send a signal to the session leader of the batch job. A batch job is
 30692 signaled by sending a request to the batch server that manages the batch job. The *qsig* utility is a
 30693 user-accessible batch client that requests the signaling of a batch job.

30694 The *qsig* utility shall signal those batch jobs for which a batch *job_identifier* is presented to the
 30695 utility. The *qsig* utility shall not signal any batch jobs whose batch *job_identifiers* are not
 30696 presented to the utility.

30697 The *qsig* utility shall signal batch jobs in the order in which the corresponding batch
 30698 *job_identifiers* are presented to the utility. If the *qsig* utility fails to process a batch *job_identifier*
 30699 successfully, the utility shall proceed to process the remaining batch *job_identifiers*, if any.

30700 The *qsig* utility shall signal batch jobs by sending a *Signal Job Request* to the batch server that
 30701 manages the batch job.

30702 For each successfully processed batch *job_identifier*, the *qsig* utility shall have received a
 30703 completion reply to each *Signal Job Request* sent to a batch server at the time the utility exits.

30704 OPTIONS

30705 The *qsig* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 30706 12.2, Utility Syntax Guidelines.

30707 The following option shall be supported by the implementation:

30708 **-s *signal*** Define the signal to be sent to the batch job.

30709 The *qsig* utility shall accept a *signal* option-argument that is either a symbolic
 30710 signal name or an unsigned integer signal number (see the POSIX.1-1990 standard,
 30711 Section 3.3.1.1). The *qsig* utility shall accept signal names for which the SIG prefix
 30712 has been omitted.

30713 If the *signal* option-argument is a signal name, the *qsig* utility shall send that name.

30714 If the *signal* option-argument is a number, the *qsig* utility shall send the signal
 30715 value represented by the number.

30716 If the **-s** option is not presented to the *qsig* utility, the utility shall send the signal
 30717 SIGTERM to each signaled batch job.

30718 OPERANDS

30719 The *qsig* utility shall accept one or more operands that conform to the syntax for a batch
 30720 *job_identifier* (see Section 3.3.1 (on page 2336)).

30721 STDIN

30722 Not used.

30723 INPUT FILES

30724 None.

30725 ENVIRONMENT VARIABLES

30726 The following environment variables shall affect the execution of *qsig*:

30727 *LANG* Provide a default value for the internationalization variables that are unset or null.
30728 If *LANG* is unset or null, the corresponding value from the implementation-
30729 defined default locale shall be used. If any of the internationalization variables
30730 contains an invalid setting, the utility shall behave as if none of the variables had
30731 been defined.

30732 *LC_ALL* If set to a non-empty string value, override the values of all the other
30733 internationalization variables.

30734 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
30735 characters (for example, single-byte as opposed to multi-byte characters in
30736 arguments).

30737 *LC_MESSAGES*
30738 Determine the locale that should be used to affect the format and contents of
30739 diagnostic messages written to standard error.

30740 *LC_TIME* Determine the format and contents of date and time strings written by *qsig*.

30741 *LOGNAME* Determine the login name of the user.

30742 *TZ* Determine the timezone in which the time and date are written. If the *TZ* variable
30743 is not set, an unspecified system default timezone is used.

30744 ASYNCHRONOUS EVENTS

30745 Default.

30746 STDOUT

30747 An implementation of the *qsig* utility may write informative messages to standard output.

30748 STDERR

30749 Used only for diagnostic messages.

30750 OUTPUT FILES

30751 None.

30752 EXTENDED DESCRIPTION

30753 None.

30754 EXIT STATUS

30755 The following exit values shall be returned:

30756 0 Successful completion.

30757 >0 An error occurred.

30758 CONSEQUENCES OF ERRORS

30759 In addition to the default behavior, the *qsig* utility shall not be required to write a diagnostic
30760 message to standard error when the error reply received from a batch server indicates that the
30761 batch *job_identifier* does not exist on the server. Whether or not the *qsig* utility waits to output the
30762 diagnostic message while attempting to locate the batch job on other servers is implementation-
30763 defined.

30764 **APPLICATION USAGE**

30765 None.

30766 **EXAMPLES**

30767 None.

30768 **RATIONALE**30769 The *qsig* utility allows users to signal batch jobs.

30770 A user may be unable to signal a batch job with the *kill* utility of the operating system for a
30771 number of reasons. First, the process ID of the batch job may be unknown to the user. Second,
30772 the processes of the batch job may be on a remote node. However, by virtue of communication
30773 between batch nodes, the *qsig* utility can arrange for the signaling of a process.

30774 Because a batch job that is not running cannot be signaled, and because the signal may not
30775 terminate the batch job, the *qsig* utility is not a substitute for the *qdel* utility.

30776 The options of the *qsig* utility allow the user to specify the signal that is to be sent to the batch
30777 job.

30778 The *-s* option allows users to specify a signal by name or by number, and thus override the
30779 default signal. The POSIX.1-1990 standard defines signals by both name and number.

30780 The *qsig* utility is a new utility, *vis-a-vis* existing practice; it has been defined in this volume of
30781 IEEE Std. 1003.1-200x in response to user-perceived shortcomings in existing practice.

30782 **FUTURE DIRECTIONS**

30783 None.

30784 **SEE ALSO**30785 *kill*, *qdel*, Chapter 3 (on page 2313)30786 **CHANGE HISTORY**

30787 Derived from IEEE Std. 1003.2d-1994.

30788 **NAME**

30789 qstat — show status of batch jobs

30790 **SYNOPSIS**30791 BE qstat [-f] *job_identifier* ...30792 qstat -Q [-f] *destination* ...30793 qstat -B [-f] *server_name* ...

30794

30795 **DESCRIPTION**

30796 The status of a batch job, batch queue, or batch server is obtained by a request to the server. The
 30797 *qstat* utility is a user-accessible batch client that requests the status of one or more batch jobs,
 30798 batch queues, or servers, and writes the status information to standard output.

30799 For each successfully processed batch *job_identifier*, the *qstat* utility shall display information
 30800 about the corresponding batch job.

30801 For each successfully processed destination, the *qstat* utility shall display information about the
 30802 corresponding batch queue.

30803 For each successfully processed server name, the *qstat* utility shall display information about the
 30804 corresponding server.

30805 The *qstat* utility shall acquire batch job status information by sending a *Job Status Request* to a
 30806 batch server. The *qstat* utility shall acquire batch queue status information by sending a *Queue*
 30807 *Status Request* to a batch server. The *qstat* utility shall acquire server status information by
 30808 sending a *Server Status Request* to a batch server.

30809 **OPTIONS**

30810 The *qstat* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 30811 12.2, Utility Syntax Guidelines.

30812 The following options shall be supported by the implementation:

30813 **-f** Specify that a full display is produced.

30814 The minimum contents of a full display are specified in the STDOUT section.

30815 Additional contents and format of a full display are implementation-defined.

30816 **-Q** Specify that the operand is a destination.

30817 The *qstat* utility shall display information about each batch queue at each
 30818 destination identified as an operand.

30819 **-B** Specify that the operand is a server name.

30820 The *qstat* utility shall display information about each server identified as an
 30821 operand.

30822 **OPERANDS**

30823 If the **-Q** option is presented to the *qstat* utility, the utility shall accept one or more operands that
 30824 conform to the syntax for a destination (see Section 3.3.2 (on page 2337)).

30825 If the **-B** option is presented to the *qstat* utility, the utility shall accept one or more *server_name*
 30826 operands.

30827 If neither the **-B** nor the **-Q** option is presented to the *qstat* utility, the utility shall accept one or
 30828 more operands that conform to the syntax for a batch *job_identifier* (see Section 3.3.1 (on page
 30829 2336)).

30830 **STDIN**

30831 Not used.

30832 **INPUT FILES**

30833 None.

30834 **ENVIRONMENT VARIABLES**30835 The following environment variables shall affect the execution of *qstat*:

30836 *COLUMNS* Override the system-selected horizontal screen size. See the Base Definitions
30837 volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables for valid
30838 values and results when it is unset or null.

30839 *HOME* Determine the path name of the user's home directory.

30840 *LANG* Provide a default value for the internationalization variables that are unset or null.
30841 If *LANG* is unset or null, the corresponding value from the implementation-
30842 defined default locale shall be used. If any of the internationalization variables
30843 contains an invalid setting, the utility shall behave as if none of the variables had
30844 been defined.

30845 *LC_ALL* If set to a non-empty string value, override the values of all the other
30846 internationalization variables.

30847 *LC_COLLATE*

30848 Determine the locale for the behavior of ranges, equivalence classes and multi-
30849 character collating elements within regular expressions.

30850 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
30851 characters (for example, single-byte as opposed to multi-byte characters in
30852 arguments).

30853 *LC_MESSAGES*

30854 Determine the locale that should be used to affect the format and contents of
30855 diagnostic messages written to standard error.

30856 *LC_NUMERIC*

30857 Determine the locale for selecting the radix character used when writing floating-
30858 point formatted output.

30859 *LC_TIME* Determine the format and contents of date and time strings written by *qstat*.

30860 *LINES* Override the system-selected vertical screen size, used as the number of lines in a
30861 screenful and the vertical screen size in visual mode. See the Base Definitions
30862 volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables for valid
30863 values and results when it is unset or null.

30864 *LOGNAME* Determine the login name of the user.

30865 *TERM* Determine the terminal type. If this variable is unset or null, and if the *-T* option is
30866 not specified, an unspecified default terminal type shall be used.

30867 *TZ* Determine the timezone in which the time and date are written. If the *TZ* variable
30868 is not set, an unspecified system default timezone is used.

30869 **ASYNCHRONOUS EVENTS**

30870 Default.

30871 **STDOUT**

30872 If an operand presented to the *qstat* utility is a batch *job_identifier* and the *-f* option is not
 30873 specified, the *qstat* utility shall display the following items on a single line, in the stated order,
 30874 with white space between each item, for each successfully processed operand:

- 30875 • The batch *job_identifier*
- 30876 • The batch job name
- 30877 • The *Job_Owner* attribute
- 30878 • The CPU time used by the batch job
- 30879 • The batch job state
- 30880 • The batch job location

30881 If an operand presented to the *qstat* utility is a batch *job_identifier* and the *-f* option is specified,
 30882 the *qstat* utility shall display the following items for each success fully processed operand:

- 30883 • The batch *job_identifier*
- 30884 • The batch job name
- 30885 • The *Job_Owner* attribute
- 30886 • The execution user ID
- 30887 • The CPU time used by the batch job
- 30888 • The batch job state
- 30889 • The batch job location
- 30890 • Additional implementation-defined information, if any, about the batch job or batch queue

30891 If an operand presented to the *qstat* utility is a destination, the *-Q* option is specified, and the *-f*
 30892 option is not specified, the *qstat* utility shall display the following items on a single line, in the
 30893 stated order, with white space between each item, for each successfully processed operand:

- 30894 • The batch queue name
- 30895 • The maximum number of batch jobs that are allowed to run in the batch queue concurrently
- 30896 • The total number of batch jobs in the batch queue
- 30897 • The status of the batch queue
- 30898 • For each state, the number of batch jobs in that state in the batch queue and the name of the
 30899 state
- 30900 • The type of batch queue (execution or routing)

30901 If the operands presented to the *qstat* utility are destinations, the *-Q* option is specified, and the
 30902 *-f* option is specified, the *qstat* utility shall display the following items for each successfully
 30903 processed operand:

- 30904 • The batch queue name
- 30905 • The maximum number of batch jobs that are allowed to run in the batch queue concurrently
- 30906 • The total number of batch jobs in the batch queue
- 30907 • The status of the batch queue

- 30908 • For each state, the number of batch jobs in that state in the batch queue and the name of the
30909 state
- 30910 • The type of batch queue (execution or routing)
- 30911 • Additional implementation-defined information, if any, about the batch queue
- 30912 If the operands presented to the *qstat* utility are batch server names, the **-B** option is specified,
30913 and the **-f** option is not specified, the *qstat* utility shall display the following items on a single
30914 line, in the stated order, with white space between each item, for each successfully processed
30915 operand:
- 30916 • The batch server name
- 30917 • The maximum number of batch jobs that are allowed to run in the batch queue concurrently
- 30918 • The total number of batch jobs managed by the batch server
- 30919 • The status of the batch server
- 30920 • For each state, the number of batch jobs in that state and the name of the state
- 30921 If the operands presented to the *qstat* utility are server names, the **-B** option is specified, and the
30922 **-f** option is specified, the *qstat* utility shall display the following items for each successfully
30923 processed operand:
- 30924 • The server name
- 30925 • The maximum number of batch jobs that are allowed to run in the batch queue concurrently
- 30926 • The total number of batch jobs managed by the server
- 30927 • The status of the server
- 30928 • For each state, the number of batch jobs in that state and the name of the state
- 30929 • Additional implementation-defined information, if any, about the server
- 30930 **STDERR**
- 30931 Used only for diagnostic messages.
- 30932 **OUTPUT FILES**
- 30933 None.
- 30934 **EXTENDED DESCRIPTION**
- 30935 None.
- 30936 **EXIT STATUS**
- 30937 The following exit values shall be returned:
- 30938 0 Successful completion.
- 30939 >0 An error occurred.
- 30940 **CONSEQUENCES OF ERRORS**
- 30941 In addition to the default behavior, the *qstat* utility shall not be required to write a diagnostic
30942 message to standard error when the error reply received from a batch server indicates that the
30943 batch *job_identifier* does not exist on the server. Whether or not the *qstat* utility waits to output
30944 the diagnostic message while attempting to locate the batch job on other servers is
30945 implementation-defined.

30946 APPLICATION USAGE

30947 None.

30948 EXAMPLES

30949 None.

30950 RATIONALE

30951 The *qstat* utility allows users to display the status of jobs and listing the batch jobs in queues.

30952 The operands of the *qstat* utility may be either job identifiers, queues (specified as destination
30953 identifiers), or batch server names. The **-Q** and **-B** options, or absence thereof, indicate the
30954 nature of the operands.

30955 The other options of the *qstat* utility allow the user to control the amount of information
30956 displayed and the format in which it is displayed. Should a user wish to display the status of a
30957 set of jobs that match a selection criteria, the *qselect* utility may be used to acquire such a list.

30958 The **-f** option allows users to request a “full” display in an implementation-defined format. |

30959 Historically, the *qstat* utility has been a part of the NQS and its derivatives, the existing practice |
30960 on which it is based.

30961 FUTURE DIRECTIONS

30962 None.

30963 SEE ALSO

30964 *qselect*, Chapter 3 (on page 2313)

30965 CHANGE HISTORY

30966 Derived from IEEE Std. 1003.2d-1994.

30967 NAME

30968 qsub — submit a script

30969 SYNOPSIS

```

30970 BE qsub [-a date_time][-A account_string][-c interval]
30971      [-C directive_prefix][-e path_name][-h][-j join_list][-k keep_list]
30972      [-m mail_options][-M mail_list][-N name]
30973      [-o path_name][-p priority][-q destination][-r y|n]
30974      [-S path_name_list][-u user_list][-v variable_list][-V]
30975      [-z][script]
30976

```

30977 DESCRIPTION

30978 To submit a script is to create a batch job that executes the script. A script is submitted by a
 30979 request to a batch server. The *qsub* utility is a user-accessible batch client that submits a script.

30980 Upon successful completion, the *qsub* utility shall have created a batch job that will execute the
 30981 submitted script.

30982 The *qsub* utility shall submit a script by sending a *Queue Job Request* to a batch server.

30983 The *qsub* utility shall place the value of the following environment variables in the *Variable_List*
 30984 attribute of the batch job: *HOME*, *LANG*, *LOGNAME*, *PATH*, *MAIL*, *SHELL*, and *TZ*. The name
 30985 of the environment variable shall be the current name prefixed with the string *PBS_O_*.

30986 **Note:** If the current value of the *HOME* variable in the environment space of the *qsub* utility
 30987 is */aa/bb/cc*, then *qsub* shall place *PBS_O_HOME=/aa/bb/cc* in the *Variable_List*
 30988 attribute of the batch job.

30989 In addition to the variables described above, the *qsub* utility shall add the following variables
 30990 with the indicated values to the variable list:

30991 *PBS_O_WORKDIR* The absolute path of the current working directory of the *qsub* utility process.

30992 *PBS_O_HOST* The name of the host on which the *qsub* utility is running.

30993 OPTIONS

30994 The *qsub* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 30995 12.2, Utility Syntax Guidelines.

30996 The following options shall be supported by the implementation:

30997 **-a date_time** Define the time at which a batch job becomes eligible for execution.

30998 The *qsub* utility shall accept an option-argument that conforms to the syntax of the
 30999 *date_time* operand of the *touch* utility.

31000

Table 4-18 Environment Variable Values (Utilities)

31001

31002

31003

31004

31005

31006

31007

31008

31009

31010

Variable Name	Value at qsub Time
<i>PBS_O_HOME</i>	<i>HOME</i>
<i>PBS_O_HOST</i>	Client host name
<i>PBS_O_LANG</i>	<i>LANG</i>
<i>PBS_O_LOGNAME</i>	<i>LOGNAME</i>
<i>PBS_O_PATH</i>	<i>PATH</i>
<i>PBS_O_MAIL</i>	<i>MAIL</i>
<i>PBS_O_SHELL</i>	<i>SHELL</i>
<i>PBS_O_TZ</i>	<i>TZ</i>
<i>PBS_O_WORKDIR</i>	Current working directory

31011

31012

31013

Note: The server that initiates execution of the batch job will add other variables to the batch job's environment; see Section 3.2.2.1 (on page 2319).

31014

31015

31016

31017

The *qsub* utility shall set the *Execution_Time* attribute of the batch job to the number of seconds since the Epoch that is equivalent to the local time expressed by the value of the *date_time* option-argument. The Epoch is defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 3.151, Epoch.

31018

31019

31020

If the *-a* option is not presented to the *qsub* utility, the utility shall set the *Execution_Time* attribute of the batch job to a time (number of seconds since the Epoch) that is earlier than the time at which the utility exits.

31021

-A *account_string*

31022

31023

Define the account to which the resource consumption of the batch job should be charged.

31024

The syntax of the *account_string* option-argument is unspecified.

31025

31026

The *qsub* utility shall set the *Account_Name* attribute of the batch job to the value of the *account_string* option-argument.

31027

31028

If the *-A* option is not presented to the *qsub* utility, the utility shall omit the *Account_Name* attribute from the attributes of the batch job.

31029

-c *interval*

Define whether the batch job should be checkpointed, and if so, how often.

31030

31031

The *qsub* utility shall accept a value for the interval option-argument that is one of the following:

31032

31033

n No checkpointing shall be performed on the batch batch job (NO_CHECKPOINT).

31034

31035

s Checkpointing shall be performed only when the batch server is shut down (CHECKPOINT_AT_SHUTDOWN).

31036

31037

31038

c Automatic periodic checkpointing shall be performed at the *Minimum_Cpu_Interval* attribute of the batch queue, in units of CPU minutes (CHECKPOINT_AT_MIN_CPU_INTERVAL).

31039

31040

31041

31042

c=minutes Automatic periodic checkpointing shall be performed every minutes of CPU time, or every *Minimum_Cpu_Interval* minutes, whichever is greater. The *minutes* argument shall conform to the syntax for unsigned integers and shall be greater than zero.

- 31043 The *qsub* utility shall set the *Checkpoint* attribute of the batch job to the value of the
31044 *interval* option-argument.
- 31045 If the *-c* option is not presented to the *qsub* utility, the utility shall set the
31046 *Checkpoint* attribute of the batch job to the single character 'u'
31047 (CHECKPOINT_UNSPECIFIED).
- 31048 **-C *directive_prefix***
31049 Define the prefix that declares a directive to the *qsub* utility within the script.
- 31050 The *directive_prefix* is not a batch job attribute; it affects the behavior of the *qsub*
31051 utility.
- 31052 If the *-C* option is presented to the *qsub* utility, and the value of the *directive_prefix*
31053 option-argument is the null string, the utility shall not scan the script file for
31054 directives. If the *-C* option is not presented to the *qsub* utility, then the value of the
31055 *PBS_DPREFIX* environment variable is used. If the environment variable is not
31056 defined, then #PBS encoded in the portable character set is the default.
- 31057 **-e *path_name*** Define the path to be used for the standard error stream of the batch job.
- 31058 The *qsub* utility shall accept a *path_name* option-argument which can be preceded
31059 by a host name element of the form *hostname:*.
- 31060 If the *path_name* option-argument constitutes an absolute path name, the *qsub*
31061 utility shall set the *Error_Path* attribute of the batch job to the value of the
31062 *path_name* option-argument.
- 31063 If the *path_name* option-argument constitutes a relative path name and no host
31064 name element is specified, the *qsub* utility shall set the *Error_Path* attribute of the
31065 batch job to the value of the absolute path name derived by expanding the
31066 *path_name* option-argument relative to the current directory of the process
31067 executing *qsub*.
- 31068 If the *path_name* option-argument constitutes a relative path name and a host name
31069 element is specified, the *qsub* utility shall set the *Error_Path* attribute of the batch
31070 job to the value of the *path_name* option-argument without expansion. The host
31071 name element shall be included.
- 31072 If the *path_name* option-argument does not include a host name element, the *qsub*
31073 utility shall prefix the path name with *hostname:*, where *hostname* is the name of the
31074 host upon which the *qsub* utility is being executed.
- 31075 If the *-e* option is not presented to the *qsub* utility, the utility shall set the
31076 *Error_Path* attribute of the batch job to the host name and path of the current
31077 directory of the submitting process and the default file name.
- 31078 The default file name for standard error has the following format:
31079 *job_name.e**sequence_number*
- 31080 **-h** Specify that a USER hold is applied to the batch job.
- 31081 The *qsub* utility shall set the value of the *Hold_Types* attribute of the batch job to the
31082 value USER.
- 31083 If the *-h* option is not presented to the *qsub* utility, the utility shall set the
31084 *Hold_Types* attribute of the batch job to the value NO_HOLD.
- 31085 **-j *join_list*** Define which streams of the batch job are to be merged. The *qsub* *-j* option shall
31086 accept a value for the *join_list* option-argument that is a string of alphanumeric

31087 characters in the portable character set (see the Base Definitions volume of
 31088 IEEE Std. 1003.1-200x, Section 6.1, Portable Character Set).

31089 The *qsub* utility shall accept a *join_list* option-argument that consists of one or
 31090 more of the characters 'e' and 'o' or the single character 'n'.

31091 All of the other batch job output streams specified will be merged into the output
 31092 stream represented by the character listed first in the *join_list* option-argument.

31093 For each unique character in the *join_list* option-argument, the *qsub* utility shall
 31094 add a value to the *Join_Path* attribute of the batch job as follows, each representing
 31095 a different batch job stream to join:

31096 *e* The standard error of the batch batch job (JOIN_STD_ERROR).

31097 *o* The standard output of the batch batch job (JOIN_STD_OUTPUT).

31098 An existing *Join_Path* attribute can be cleared by the following join type:

31099 **n** NO_JOIN

31100 If **n** is specified, then no files are joined. The *qsub* utility shall consider it an error if
 31101 any join type other than **n** is combined with join type **n**.

31102 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or
 31103 'n' within the *join_list* option-argument. The *qsub* utility shall permit the
 31104 repetition of characters, but shall not assign additional meaning to the repeated
 31105 characters.

31106 An implementation may define other join types. The conformance document for an
 31107 implementation shall describe any additional batch job streams, how they are
 31108 specified, their internal behavior, and how they affect the behavior of the utility.

31109 If the **-j** option is not presented to the *qsub* utility, the utility shall set the value of
 31110 the *Join_Path* attribute of the batch job to NO_JOIN.

31111 **-k keep_list** Define which output of the batch job to retain on the execution host.

31112 The *qsub* **-k** option shall accept a value for the *keep_list* option-argument that is a
 31113 string of alphanumeric characters in the portable character set (see the Base
 31114 Definitions volume of IEEE Std. 1003.1-200x, Section 6.1, Portable Character Set).

31115 The *qsub* utility shall accept a *keep_list* option-argument that consists of one or
 31116 more of the characters 'e' and 'o' or the single character 'n'.

31117 For each unique character in the *keep_list* option-argument, the *qsub* utility shall
 31118 add a value to the *Keep_Files* attribute of the batch job as follows, each representing
 31119 a different batch job stream to keep:

31120 *e* The standard error of the batch batch job (KEEP_STD_ERROR).

31121 *o* The standard output of the batch batch job (KEEP_STD_OUTPUT).

31122 If both *e* and *o* are specified, then both files are retained. An existing *Keep_Files*
 31123 attribute can be cleared by the following keep type:

31124 **n** NO_KEEP

31125 If **n** is specified, then no files are retained. The *qsub* utility shall consider it an error
 31126 if any keep type other than **n** is combined with keep type **n**.

31127 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or
 31128 'n' within the *keep_list* option-argument. The *qsub* utility shall permit the

- 31129 repetition of characters, but shall not assign additional meaning to the repeated
31130 characters.
- 31131 An implementation may define other keep types. The conformance document for
31132 an implementation shall describe any additional keep types, how they are
31133 specified, their internal behavior, and how they affect the behavior of the utility. If
31134 the `-k` option is not presented to the `qsub` utility, the utility shall set the `Keep_Files`
31135 attribute of the batch job to the value `NO_KEEP`.
- 31136 **-m** *mail_options*
- 31137 Define the points in the execution of the batch job at which the batch server that
31138 manages the batch job shall send mail about a change in the state of the batch job.
- 31139 The `qsub -m` option shall accept a value for the *mail_options* option-argument that
31140 is a string of alphanumeric characters in the portable character set (see the Base
31141 Definitions volume of IEEE Std. 1003.1-200x, Section 6.1, Portable Character Set).
- 31142 The `qsub` utility shall accept a value for the *mail_options* option-argument that is a
31143 string of one or more of the characters 'e', 'b', and 'a', or the single character
31144 'n'.
- 31145 For each unique character in the *mail_options* option-argument, the `qsub` utility shall
31146 add a value to the `Mail_Users` attribute of the batch job as follows, each
31147 representing a different time during the life of a batch job at which to send mail:
- 31148 e MAIL_AT_EXIT
- 31149 b MAIL_AT_BEGINNING
- 31150 a MAIL_AT_ABORT
- 31151 If any of these characters are duplicated in the *mail_options* option-argument, the
31152 duplicates shall be ignored.
- 31153 An existing `Mail_Points` attribute can be cleared by the following mail type:
- 31154 n NO_MAIL
- 31155 If `n` is specified, then mail is not sent. The `qsub` utility shall consider it an error if
31156 any mail type other than `n` is combined with mail type `n`.
- 31157 Strictly conforming applications shall not repeat any of the characters 'e', 'b',
31158 'a', or 'n' within the *mail_options* option-argument.
- 31159 The `qsub` utility shall permit the repetition of characters, but shall not assign
31160 additional meaning to the repeated characters. An implementation may define
31161 other mail types. The conformance document for an implementation shall describe
31162 any additional mail types, how they are specified, their internal behavior, and how
31163 they affect the behavior of the utility.
- 31164 If the `-m` option is not presented to the `qsub` utility, the utility shall set the
31165 `Mail_Points` attribute to the value `MAIL_AT_ABORT`.
- 31166 **-M** *mail_list* Define the list of users to which a batch server that executes the batch job shall
31167 send mail, if the server sends mail about the batch job.
- 31168 The syntax of the *mail_list* option-argument is unspecified.
- 31169 If the implementation of the `qsub` utility uses a name service to locate users, the
31170 utility should accept the syntax used by the name service.

- 31171 If the implementation of the *qsub* utility does not use a name service to locate
31172 users, the implementation should accept the following syntax for user names:
- 31173 *mail_address*[, , *mail_address* , , ...]
- 31174 The interpretation of *mail_address* is implementation-defined.
- 31175 The *qsub* utility shall set the *Mail_Users* attribute of the batch job to the value of the
31176 *mail_list* option-argument.
- 31177 If the **-M** option is not presented to the *qsub* utility, the utility shall place only the
31178 user name and host name for the current process in the *Mail_Users* attribute of the
31179 batch job.
- 31180 **-N name** Define the name of the batch job.
- 31181 The *qsub* **-N** option shall accept a value for the *name* option-argument that is a
31182 string of up to 15 alphanumeric characters in the portable character set (see the
31183 Base Definitions volume of IEEE Std. 1003.1-200x, Section 6.1, Portable Character
31184 Set) where the first character is alphabetic.
- 31185 The *qsub* utility shall set the value of the *Job_Name* attribute of the batch job to the
31186 value of the *name* option-argument.
- 31187 If the **-N** option is not presented to the *qsub* utility, the utility shall set the
31188 *Job_Name* attribute of the batch job to the name of the *script* argument from which
31189 the directory specification if any, has been removed.
- 31190 If the **-N** option is not presented to the *qsub* utility, and the script is read from
31191 standard input, the utility shall set the *Job_Name* attribute of the batch job to the
31192 value STDIN.
- 31193 **-o path_name** Define the path for the standard output of the batch job.
- 31194 The *qsub* utility shall accept a *path_name* option-argument that conforms to the
31195 syntax of the *path_name* element defined in the POSIX.1-1990 standard, which can
31196 be preceded by a host name element of the form *hostname*:
- 31197 If the *path_name* option-argument constitutes an absolute path name, the *qsub*
31198 utility shall set the *Output_Path* attribute of the batch job to the value of the
31199 *path_name* option-argument without expansion.
- 31200 If the *path_name* option-argument constitutes a relative path name and no host
31201 name element is specified, the *qsub* utility shall set the *Output_Path* attribute of the
31202 batch job to the path name derived by expanding the value of the *path_name*
31203 option-argument relative to the current directory of the process executing the *qsub*.
- 31204 If the *path_name* option-argument constitutes a relative path name and a host name
31205 element is specified, the *qsub* utility shall set the *Output_Path* attribute of the batch
31206 job to the value of the *path_name* option-argument without expansion.
- 31207 If the *path_name* option-argument does not specify a host name element, the *qsub*
31208 utility shall prefix the path name with *hostname*:, where *hostname* is the name of the
31209 host upon which the *qsub* utility is executing.
- 31210 If the **-o** option is not presented to the *qsub* utility, the utility shall set the
31211 *Output_Path* attribute of the batch job to the host name and path of the current
31212 directory of the submitting process and the default file name.
- 31213 The default file name for standard output has the following format:

- 31214 *job_name.osequence_number*
- 31215 **-p priority** Define the priority the batch job should have relative to other batch jobs owned by
31216 the batch server.
- 31217 The *qsub* utility shall set the *Priority* attribute of the batch job to the value of the
31218 *priority* option-argument.
- 31219 If the **-p** option is not presented to the *qsub* utility, the value of the *Priority*
31220 attribute is implementation-defined.
- 31221 The *qsub* utility shall accept a value for the *priority* option-argument that conforms
31222 to the syntax for signed decimal integers, and which is not less than -1 024 and not
31223 greater than 1 023.
- 31224 **-q destination**
- 31225 Define the destination of the batch job.
- 31226 The destination is not a batch job attribute; it determines the batch server, and
31227 possibly the batch queue, to which the *qsub* utility batch queues the batch job.
- 31228 The *qsub* utility shall submit the script to the batch server named by the *destination*
31229 option-argument or the server that owns the batch queue named in the *destination*
31230 option-argument.
- 31231 The *qsub* utility shall accept an option-argument for the **-q** option that conforms to
31232 the syntax for a destination (see Section 3.3.2 (on page 2337)).
- 31233 If the **-q** option is not presented to the *qsub* utility, the *qsub* utility shall submit the
31234 batch job to the default destination. The mechanism for determining the default
31235 destination is implementation-defined.
- 31236 **-r y | n** Define whether the batch job is rerunable.
- 31237 If the value of the option-argument is *y*, the *qsub* utility shall set the *Rerunable*
31238 attribute of the batch job to TRUE.
- 31239 If the value of the option-argument is *n*, the *qsub* utility shall set the *Rerunable*
31240 attribute of the batch job to FALSE.
- 31241 If the **-r** option is not presented to the *qsub* utility, the utility shall set the *Rerunable*
31242 attribute of the batch job to TRUE.
- 31243 **-S path_name_list**
- 31244 Define the path name to the shell under which the batch job is to execute.
- 31245 The *qsub* utility shall accept a *path_name_list* option-argument that conforms to the
31246 following syntax:
- 31247 *pathname[@host][, , pathname[@host] , , ...]*
- 31248 The *qsub* utility shall allow only one path name for a given host name. The *qsub*
31249 utility shall allow only one path name that is missing a corresponding host name.
- 31250 The *qsub* utility shall add a value to the *Shell_Path_List* attribute of the batch job for
31251 each entry in the *path_name_list* option-argument.
- 31252 If the **-S** option is not presented to the *qsub* utility, the utility shall set the
31253 *Shell_Path_List* attribute of the batch job to the null string.
- 31254 The conformance document for an implementation shall describe the mechanism
31255 used to set the default shell and determine the current value of the default shell.

- 31256 An implementation shall provide a means for the installation to set the default
31257 shell to the login shell of the user under which the batch job is to execute. See
31258 Section 3.3.3 (on page 2337) for a means of removing *keyword=value* (and
31259 *value@keyword*) pairs and other general rules for list-oriented batch job attributes.
- 31260 **-u *user_list*** Define the user name under which the batch job is to execute.
- 31261 The *qsub* utility shall accept a *user_list* option-argument that conforms to the
31262 following syntax:
- 31263 *username[@host][, ,username[@host], , ...]*
- 31264 The *qsub* utility shall accept only one user name that is missing a corresponding
31265 host name. The *qsub* utility shall accept only one user name per named host.
- 31266 The *qsub* utility shall add a value to the *User_List* attribute of the batch job for each
31267 entry in the *user_list* option-argument.
- 31268 If the **-u** option is not presented to the *qsub* utility, the utility shall set the *User_List*
31269 attribute of the batch job to the user name from which the utility is executing. See
31270 Section 3.3.3 (on page 2337) for a means of removing *keyword=value* (and
31271 *value@keyword*) pairs and other general rules for list-oriented batch job attributes.
- 31272 **-v *variable_list***
- 31273 Add to the list of variables that are exported to the session leader of the batch job.
- 31274 A *variable_list* is a set of strings of either the form *<variable>* or *<variable=value>*,
31275 delimited by commas.
- 31276 If the **-v** option is presented to the *qsub* utility, the utility shall also add, to the
31277 environment *Variable_List* attribute of the batch job, every variable named in the
31278 environment *variable_list* option-argument and, optionally, values of specified
31279 variables.
- 31280 If a value is not provided on the command line, the *qsub* utility shall set the value
31281 of each variable in the environment *Variable_List* attribute of the batch job to the
31282 value of the corresponding environment variable for the process in which the
31283 utility is executing; see Table 4-18 (on page 3013).
- 31284 A conforming application shall not repeat a variable in the environment
31285 *variable_list* option-argument.
- 31286 The *qsub* utility shall not repeat a variable in the environment *Variable_List*
31287 attribute of the batch job. See Section 3.3.3 (on page 2337) for a means of removing
31288 *keyword=value* (and *value@keyword*) pairs and other general rules for list-oriented
31289 batch job attributes.
- 31290 **-V** Specify that all of the environment variables of the process are exported to the
31291 context of the batch job.
- 31292 The *qsub* utility shall place every environment variable in the process in which the
31293 utility is executing in the list and shall set the value of each variable in the attribute
31294 to the value of that variable in the process.
- 31295 **-z** Specify that the utility does not write the batch *job_identifier* of the created batch
31296 job to standard output.
- 31297 If the **-z** option is presented to the *qsub* utility, the utility shall not write the batch
31298 *job_identifier* of the created batch job to standard output.

31299 If the `-z` option is not presented to the *qsub* utility, the utility shall write the
31300 identifier of the created batch job to standard output.

31301 OPERANDS

31302 The *qsub* utility shall accept a *script* operand that indicates the path to the script of the batch job.

31303 If the *script* operand is not presented to the *qsub* utility, or if the operand is the single-character
31304 string `'-'`, the utility shall read the script from standard input.

31305 If the script represents a partial path, the *qsub* utility shall expand the path relative to the current
31306 directory of the process executing the utility.

31307 STDIN

31308 The *qsub* utility reads the script of the batch job from standard input if the script operand is
31309 omitted or is the single character `'-'`.

31310 INPUT FILES

31311 In addition to binding the file indicated by the *script* operand to the batch job, the *qsub* utility
31312 reads the script file and acts on directives in the script.

31313 ENVIRONMENT VARIABLES

31314 The following environment variables shall affect the execution of *qsub*:

31315 *LANG* Provide a default value for the internationalization variables that are unset or null.
31316 If *LANG* is unset or null, the corresponding value from the implementation-
31317 defined default locale shall be used. If any of the internationalization variables
31318 contains an invalid setting, the utility shall behave as if none of the variables had
31319 been defined.

31320 *LC_ALL* If set to a non-empty string value, override the values of all the other
31321 internationalization variables.

31322 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
31323 characters (for example, single-byte as opposed to multi-byte characters in
31324 arguments).

31325 *LC_MESSAGES*
31326 Determine the locale that should be used to affect the format and contents of
31327 diagnostic messages written to standard error.

31328 *LC_TIME* Determine the format and contents of date and time strings written by *qsub*.

31329 *LOGNAME* Determine the login name of the user.

31330 *PBS_DPREFIX*

31331 Determine the default prefix for directives within the script.

31332 *SHELL* Determine the path name of the preferred command language interpreter of the
31333 user.

31334 *TZ* Determine the timezone in which the time and date are written. If the *TZ* variable
31335 is not set, an unspecified system default timezone is used.

31336 ASYNCHRONOUS EVENTS

31337 Once created, a batch job exists until it exits, aborts, or is deleted.

31338 After a batch job is created by the *qsub* utility, batch servers might route, execute, modify, or
31339 delete the batch job.

31340 **STDOUT**

31341 The *qsub* utility writes the batch *job_identifier* assigned to the batch job to standard output, unless
31342 the **-z** option is specified.

31343 **STDERR**

31344 Used only for diagnostic messages.

31345 **OUTPUT FILES**

31346 None.

31347 **EXTENDED DESCRIPTION**31348 **Script Preservation**

31349 The *qsub* utility shall make the script available to the server executing the batch job in such a way
31350 that the server executes the script as it exists at the time of submission.

31351 The *qsub* utility can send a copy of the script to the server with the *Queue Job Request* or store a
31352 temporary copy of the script in a location specified to the server.

31353 **Option Specification**

31354 A script can contain directives to the *qsub* utility.

31355 The *qsub* utility shall scan the lines of the script for directives, skipping blank lines, until the first
31356 line that begins with a string other than the directive string; if directives occur on subsequent
31357 lines, the utility shall ignore those directives.

31358 Lines are separated by a <newline>. If the first line of the script begins with "#!" or a colon
31359 (' : '), then it is skipped. The *qsub* utility shall process a line in the script as a directive if and
31360 only if the string of characters from the first non-white-space character on the line until the first
31361 <space> or <tab> character on the line match the directive prefix. If a line in the script contains a
31362 directive and the final characters of the line are backslash ('\ ') and <newline>, then the next
31363 line shall be interpreted as a continuation of that directive.

31364 The *qsub* utility shall process the options and option-arguments contained on the directive prefix
31365 line using the same syntax as if the options were input on the *qsub* utility.

31366 The *qsub* utility shall continue to process a directive prefix line until after a <newline> is
31367 encountered. An implementation may ignore lines which, according to the syntax of the shell
31368 that will interpret the script, are comments. An implementation shall describe in the
31369 conformance document the format of any shell comments that it will recognize.

31370 If an option is present in both a directive and the arguments to the *qsub* utility, the utility shall
31371 ignore the option and the corresponding option-argument, if any, in the directive.

31372 If an option that is present in the directive is not present in the arguments to the *qsub* utility, the
31373 utility shall process the option and the option-argument, if any.

31374 In order of preference, the *qsub* utility shall select the directive prefix from one of the following
31375 sources:

- 31376 • If the **-C** option is presented to the utility, the value of the *directive_prefix* option-argument
- 31377 • If the environment variable *PBS_DPREFIX* is defined, the value of that variable
- 31378 • The four-character string "#PBS" encoded in the portable character set

31379 If the **-C** option is present in the script file it shall be ignored.

31380 **EXIT STATUS**

31381 The following exit values shall be returned:

31382 0 Successful completion.

31383 >0 An error occurred.

31384 **CONSEQUENCES OF ERRORS**

31385 Default.

31386 **APPLICATION USAGE**

31387 None.

31388 **EXAMPLES**

31389 None.

31390 **RATIONALE**

31391 The *qsub* utility allows users to create a batch job that will process the script specified as the operand of the utility.

31393 The options of the *qsub* utility allow users to control many aspects of the queuing and execution of a batch job.

31395 The **-a** option allows users to designate the time after which the batch job will become eligible to run. By specifying an execution time, users can take advantage of resources at off-peak hours, synchronize jobs with chronologically predictable events, and perhaps take advantage of off-peak pricing of computing time. For these reasons and others, a timing option is existing practice on the part of almost every batch system, including NQS.

31400 The **-A** option allows users to specify the account that will be charged for the batch job. Support for account is not mandatory for conforming batch servers.

31402 The **-C** option allows users to prescribe the prefix for directives within the script file. The default prefix "#PBS" may be inappropriate if the script will be interpreted with an alternate shell, as specified by the **-S** option.

31405 The **-c** option allows users to establish the checkpointing interval for their jobs. A checkpointing system, which is not defined by this volume of IEEE Std. 1003.1-200x, allows recovery of a batch job at the most recent checkpoint in the event of a crash. Checkpointing is typically used for jobs that consume expensive computing time or must meet a critical schedule. Users should be allowed to make the tradeoff between the overhead of checkpointing and the risk to the timely completion of the batch job; therefore, this volume of IEEE Std. 1003.1-200x provides the checkpointing interval option. Support for checkpointing is optional for batch servers.

31412 The **-e** option allows users to redirect the standard error streams of their jobs to a non-default path. For example, if the submitted script generally produces a great deal of useless error output, a user might redirect the standard error output to the null device. Or, if the file system holding the default location (the home directory of the user) has too little free space, the user might redirect the standard error stream to a file in another file system.

31417 The **-h** option allows users to create a batch job that is held until explicitly released. The ability to create a held job is useful when some external event must complete before the batch job can execute. For example, the user might submit a held job and release it when the system load has dropped.

31421 The **-j** option allows users to merge the standard error of a batch job into its standard output stream, which has the advantage of showing the sequential relationship between output and error messages.

- 31424 The **-m** option allows users to designate those points in the execution of a batch job at which
31425 mail will be sent to the submitting user, or to the account(s) indicated by the **-M** option. By
31426 requesting mail notification at points of interest in the life of a job, the submitting user, or other
31427 designated users, can track the progress of a batch job.
- 31428 The **-N** option allows users to associate a name with the batch job. The job name in no way
31429 affects the processing of the batch job, but rather serves as a mnemonic handle for users. For
31430 example, the batch job name can help the user distinguish between multiple jobs listed by the
31431 *qstat* utility.
- 31432 The **-o** option allows users to redirect the standard output stream. A user might, for example,
31433 wish to redirect to the null device the standard output stream of a job that produces copious yet
31434 superfluous output.
- 31435 The **-P** option allows users to designate the relative priority of a batch job for selection from a
31436 queue.
- 31437 The **-q** option allows users to specify an initial queue for the batch job. If the user specifies a
31438 routing queue, the batch server routes the batch job to another queue for execution or
31439 further routing. If the user specifies a non-routing queue, the batch server of the queue
31440 eventually executes the batch job.
- 31441 The **-r** option allows users to control whether the submitted job will be rerun if the controlling
31442 batch node fails during execution of the batch job. The **-r** option likewise allows users to
31443 indicate whether or not the batch job is eligible to be rerun by the *qrerun* utility. Some jobs cannot
31444 be correctly rerun because of changes they make in the state of databases or other aspects of
31445 their environment. This volume of IEEE Std. 1003.1-200x specifies that the default, if the **-r**
31446 option is not presented to the utility, will be that the batch job cannot be rerun, since the result of
31447 rerunning a non-rerunable job might be catastrophic.
- 31448 The **-S** option allows users to specify the program (usually a shell) that will be invoked to
31449 process the script of the batch job. This option has been modified to allow a list of shell names
31450 and locations associated with different hosts.
- 31451 The **-u** option is useful when the submitting user is authorized to use more than one account on
31452 a given host, in which case the **-u** option allows the user to select from among those accounts.
31453 The option-argument is a list of user-host pairs, so that the submitting user can provide different
31454 user identifiers for different nodes in the event the batch job is routed. The **-u** option provides a
31455 lot of flexibility to accommodate sites with complex account structures. Users that have the
31456 same user identifier on all the hosts they are authorized to use will not need to use the **-u** option.
- 31457 The **-V** option allows users to export all their current environment variables, as of the time the
31458 batch job is submitted, to the context of the processes of the batch job.
- 31459 The **-v** option allows users to export specific environment variables from their current process
31460 to the processes of the batch job.
- 31461 The **-z** option allows users to suppress the writing of the batch job identifier to standard output.
31462 The **-z** option is an existing NQS practice that has been standardized.
- 31463 Historically, the *qsub* utility has served the batch job-submission function in the NQS system, the
31464 existing practice on which it is based. Some changes and additions have been made to the *qsub*
31465 utility in this volume of IEEE Std. 1003.1-200x, *vis-a-vis* NQS, as a result of the growing pool of
31466 experience with distributed batch systems.
- 31467 The set of features of the *qsub* utility as defined in this volume of IEEE Std. 1003.1-200x appears
31468 to incorporate all the common existing practice on potentially POSIX-conformant platforms.
31469 Where implementors wish to extend the functionality of their *qsub* utility, they may (as defined

31470 by IEEE Std. 1003.1-200x) use the **-W** option to provide implementation-defined extensions.

31471 **FUTURE DIRECTIONS**

31472 None.

31473 **SEE ALSO**

31474 *qrun, qstat, touch*, Chapter 3 (on page 2313)

31475 **CHANGE HISTORY**

31476 Derived from IEEE Std. 1003.2d-1994.

31477 **Issue 6**

31478 The **-I** option has been removed as there is no portable description of the resources that are
31479 allowed or required by the batch job.

31480 **NAME**

31481 read — read a line from standard input

31482 **SYNOPSIS**

31483 read [-r] var...

31484 **DESCRIPTION**31485 The *read* utility shall read a single line from standard input.

31486 By default, unless the *-r* option is specified, backslash ('**') shall act as an escape character, as
 31487 described in Section 2.2.1 (on page 2236). If standard input is a terminal device and the invoking
 31488 shell is interactive, *read* shall prompt for a continuation line when:

- 31489 • The shell reads an input line ending with a backslash, unless the *-r* option is specified.
- 31490 • A here-document is not terminated after a <newline> character is entered.

31491 The line shall be split into fields as in the shell (see Section 2.6.5 (on page 2249)); the first field
 31492 shall be assigned to the first variable *var*, the second field to the second variable *var*, and so on. If
 31493 there are fewer *var* operands specified than there are fields, the leftover fields and their
 31494 intervening separators shall be assigned to the last *var*. If there are fewer fields than *vars*, the
 31495 remaining *vars* shall be set to empty strings.

31496 The setting of variables specified by the *var* operands shall affect the current shell execution
 31497 environment; see Section 2.13 (on page 2273). If it is called in a subshell or separate utility
 31498 execution environment, such as one of the following:

```
31499 (read foo)
31500 nohup read ...
31501 find . -exec read ... \;
```

31502 it shall not affect the shell variables in the caller's environment.

31503 **OPTIONS**31504 The *read* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
31505 12.2, Utility Syntax Guidelines.

31506 The following option is supported:

31507 *-r* Do not treat a backslash character in any special way. Consider each backslash to
 31508 be part of the input line.

31509 **OPERANDS**

31510 The following operand shall be supported:

31511 *var* The name of an existing or nonexisting shell variable.31512 **STDIN**

31513 The standard input shall be a text file.

31514 **INPUT FILES**

31515 None.

31516 **ENVIRONMENT VARIABLES**31517 The following environment variables shall affect the execution of *read*:

31518 *IFS* Determine the internal field separators used to delimit fields; see Section 2.5.3 (on
 31519 page 2242).

31520 *LANG* Provide a default value for the internationalization variables that are unset or null.
 31521 If *LANG* is unset or null, the corresponding value from the implementation-
 31522 defined default locale shall be used. If any of the internationalization variables

31523 contains an invalid setting, the utility shall behave as if none of the variables had
 31524 been defined.

31525 **LC_ALL** If set to a non-empty string value, override the values of all the other
 31526 internationalization variables.

31527 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 31528 characters (for example, single-byte as opposed to multi-byte characters in
 31529 arguments).

31530 **LC_MESSAGES**
 31531 Determine the locale that should be used to affect the format and contents of
 31532 diagnostic messages written to standard error.

31533 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

31534 **PS2** Provide the prompt string that an interactive shell shall write to standard error
 31535 when a line ending with a backslash is read and the **-r** option was not specified, or
 31536 if a here-document is not terminated after a <newline> character is entered.

31537 **ASYNCHRONOUS EVENTS**
 31538 Default.

31539 **STDOUT**
 31540 Not used.

31541 **STDERR**
 31542 Used for diagnostic messages and prompts for continued input.

31543 **OUTPUT FILES**
 31544 None.

31545 **EXTENDED DESCRIPTION**
 31546 None.

31547 **EXIT STATUS**
 31548 The following exit values shall be returned:
 31549 0 Successful completion.
 31550 >0 End-of-file was detected or an error occurred.

31551 **CONSEQUENCES OF ERRORS**
 31552 Default.

31553 **APPLICATION USAGE**
 31554 The *read* utility has historically been a shell built-in.
 31555 The **-r** option is included to enable *read* to subsume the purpose of the *line* utility, which is not
 31556 included in IEEE Std. 1003.1-200x.
 31557 The results are undefined if an end-of-file is detected following a backslash at the end of a line
 31558 when **-r** is not specified.

31559 **EXAMPLES**
 31560 The following command:
 31561 while read -r xx yy
 31562 do
 31563 printf "%s %s\n" "\$yy" "\$xx"
 31564 done < *input_file*

31565 prints a file with the first field of each line moved to the end of the line.

31566 **RATIONALE**

31567 The *read* utility historically has been a shell built-in. It was separated off into its own utility to
31568 take advantage of the richer description of functionality introduced by this volume of
31569 IEEE Std. 1003.1-200x.

31570 Since *read* affects the current shell execution environment, it is generally provided as a shell
31571 regular built-in. If it is called in a subshell or separate utility execution environment, such as one
31572 of the following:

```
31573 (read foo)
31574 nohup read ...
31575 find . -exec read ... \;
```

31576 it does not affect the shell variables in the environment of the caller.

31577 **FUTURE DIRECTIONS**

31578 None.

31579 **SEE ALSO**

31580 None.

31581 **CHANGE HISTORY**

31582 First released in Issue 2.

31583 **Issue 4**

31584 Relocated from the *sh* description for alignment with the ISO/IEC 9945-2: 1993 standard.

31585 NAME

31586 renice — set nice values of running processes

31587 SYNOPSIS

31588 UP `renice -n increment [-g | -p | -u] ID ...`

31589

31590 DESCRIPTION

31591 The *renice* utility shall request that the nice values (see the Base Definitions volume of
 31592 IEEE Std. 1003.1-200x, Section 3.241, Nice Value) of one or more running processes be changed.
 31593 By default, the applicable processes are specified by their process IDs. When a process group is
 31594 specified (see `-g`), the request applies to all processes in the process group.

31595 The nice value shall be bounded in an implementation-defined manner. If the requested
 31596 *increment* would raise or lower the nice value of the executed utility beyond implementation-
 31597 defined limits, then the limit whose value was exceeded shall be used.

31598 When a user is *reniced*, the request applies to all processes whose saved set-user-ID matches the
 31599 user ID corresponding to the user.

31600 Regardless of which options are supplied or any other factor, *renice* shall not alter the nice values
 31601 of any process unless the user requesting such a change has appropriate privileges to do so for
 31602 the specified process. If the user lacks appropriate privileges to perform the requested action, the
 31603 utility shall return an error status.

31604 The saved set-user-ID of the user's process shall be checked instead of its effective user ID when
 31605 *renice* attempts to determine the user ID of the process in order to determine whether the user
 31606 has appropriate privileges.

31607 OPTIONS

31608 The *renice* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 31609 12.2, Utility Syntax Guidelines.

31610 The following options shall be supported:

31611 `-g` Interpret all operands as unsigned decimal integer process group IDs.

31612 `-n increment` Specify how the nice value of the specified process or processes is to be adjusted.
 31613 The *increment* option-argument is a positive or negative decimal integer that shall
 31614 be used to modify the nice value of the specified process or processes.

31615 Positive *increment* values shall cause a lower nice value. Negative *increment* values
 31616 may require appropriate privileges and shall cause a higher nice value.

31617 `-p` Interpret all operands as unsigned decimal integer process IDs. The `-p` option is
 31618 the default if no options are specified.

31619 `-u` Interpret all operands as users. If a user exists with a user name equal to the
 31620 operand, then the user ID of that user is used in further processing. Otherwise, if
 31621 the operand represents an unsigned decimal integer, it shall be used as the numeric
 31622 user ID of the user.

31623 OPERANDS

31624 The following operands shall be supported:

31625 *ID* A process ID, process group ID, or user name/user ID, depending on the option
 31626 selected.

31627 **STDIN**

31628 Not used.

31629 **INPUT FILES**

31630 None.

31631 **ENVIRONMENT VARIABLES**31632 The following environment variables shall affect the execution of *renice*:

31633 *LANG* Provide a default value for the internationalization variables that are unset or null.
31634 If *LANG* is unset or null, the corresponding value from the implementation-
31635 defined default locale shall be used. If any of the internationalization variables
31636 contains an invalid setting, the utility shall behave as if none of the variables had
31637 been defined.

31638 *LC_ALL* If set to a non-empty string value, override the values of all the other
31639 internationalization variables.

31640 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
31641 characters (for example, single-byte as opposed to multi-byte characters in
31642 arguments).

31643 *LC_MESSAGES*
31644 Determine the locale that should be used to affect the format and contents of
31645 diagnostic messages written to standard error.

31646 XSI *NLS_PATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

31647 **ASYNCHRONOUS EVENTS**

31648 Default.

31649 **STDOUT**

31650 Not used.

31651 **STDERR**

31652 Used only for diagnostic messages.

31653 **OUTPUT FILES**

31654 None.

31655 **EXTENDED DESCRIPTION**

31656 None.

31657 **EXIT STATUS**

31658 The following exit values shall be returned:

31659 0 Successful completion.

31660 >0 An error occurred.

31661 **CONSEQUENCES OF ERRORS**

31662 Default.

31663 **APPLICATION USAGE**

31664 None.

31665 **EXAMPLES**

31666 1. Adjust the nice value so that process IDs 987 and 32 would have a lower nice value:

31667 `renice -n 5 -p 987 32`31668 2. Adjust the nice value so that group IDs 324 and 76 would have a higher nice value, if the
31669 user has the appropriate privileges to do so:31670 `renice -n -4 -g 324 76`31671 3. Adjust the nice value so that numeric user ID 8 and user **sas** would have a lower nice
31672 value:31673 `renice -n 4 -u 8 sas`31674 Useful nice value increments on historical systems include 19 or 20 (the affected processes run
31675 only when nothing else in the system attempts to run) and any negative number (to make
31676 processes run faster).31677 **RATIONALE**31678 The *gid*, *pid*, and *user* specifications do not fit either the definition of operand or option-
31679 argument. However, for clarity, they have been included in the OPTIONS section, rather than
31680 the OPERANDS section.31681 The definition of nice value is not intended to suggest that all processes in a system have
31682 priorities that are comparable. Scheduling policy extensions such as the realtime priorities in
31683 POSIX.4 make the notion of a single underlying priority for all scheduling policies problematic.
31684 Some systems may implement the *nice*-related features to affect all processes on the system,
31685 others to affect just the general time-sharing activities implied by this volume of
31686 IEEE Std. 1003.1-200x, and others may have no effect at all. Because of the use of
31687 “implementation-defined” in *nice* and *renice*, a wide range of implementation strategies are
31688 possible.31689 Originally, this utility was written in the historical manner, using the term “nice value”. This
31690 was always a point of concern with users because it was never intuitively obvious what this
31691 meant. With a newer version of *renice*, which used the term “system scheduling priority”, it was
31692 hoped that novice users could better understand what this utility was meant to do. Also, it
31693 would be easier to document what the utility was meant to do. Unfortunately, the addition of
31694 the POSIX realtime scheduling capabilities introduced the concepts of process and thread
31695 scheduling priorities that were totally unaffected by the *nice/renice* utilities or the
31696 *nice()/setpriority()* functions. Continuing to use the term “system scheduling priority” would
31697 have incorrectly suggested that these utilities and functions were indeed affecting these realtime
31698 priorities. It was decided to revert to the historical term “nice value” to reference this unrelated
31699 process attribute.31700 Although this utility has use by system administrators (and in fact appears in the system
31701 administration portion of the BSD documentation), the standard developers considered that it
31702 was very useful for individual end users to control their own processes.31703 **FUTURE DIRECTIONS**

31704 None.

31705 **SEE ALSO**31706 *nice*31707 **CHANGE HISTORY**

31708 First released in Issue 4.

31709 **Issue 5**31710 In the SYNOPSIS, an ellipsis is added to the `-u` option in all three obsolescent forms.31711 **Issue 6**

31712 This utility is now marked as part of the User Portability Utilities option.

31713 The APPLICATION USAGE section is added.

31714 The obsolescent forms of the SYNOPSIS are removed.

31715 Text previously conditional on `POSIX_SAVED_IDS` is mandatory in this issue. This is a FIPS
31716 requirement.

31717 NAME

31718 rm — remove directory entries

31719 SYNOPSIS

31720 rm [-fiRr] *file*...

31721 DESCRIPTION

31722 The *rm* utility shall remove the directory entry specified by each *file* argument.

31723 If either of the files dot or dot-dot are specified as the basename portion of an operand (that is,
 31724 the final path name component), *rm* shall write a diagnostic message to standard error and do
 31725 nothing more with such operands.

31726 For each *file* the following steps shall be taken:

- 31727 1. If the *file* does not exist:
 - 31728 a. If the **-f** option is not specified, write a diagnostic message to standard error.
 - 31729 b. Go on to any remaining *files*.
 - 31730 2. If *file* is of type directory, the following steps shall be taken:
 - 31731 a. If neither the **-R** option nor the **-r** option is specified, write a diagnostic message to
 31732 standard error, do nothing more with *file*, and go on to any remaining files.
 - 31733 b. If the **-f** option is not specified, and either the permissions of *file* do not permit
 31734 writing and the standard input is a terminal or the **-i** option is specified, write a
 31735 prompt to standard error and read a line from the standard input. If the response is
 31736 not affirmative, do nothing more with the current file and go on to any remaining
 31737 files.
 - 31738 c. For each entry contained in *file*, other than dot or dot-dot, the four steps listed here
 31739 (1-4) shall be taken with the entry as if it were a *file* operand. The *rm* utility shall not
 31740 traverse directories by following symbolic links into other parts of the hierarchy, but
 31741 shall remove the links themselves.
 - 31742 d. If the **-i** option is specified, write a prompt to standard error and read a line from the
 31743 standard input. If the response is not affirmative, do nothing more with the current
 31744 file, and go on to any remaining files.
 - 31745 3. If *file* is not of type directory, the **-f** option is not specified, and either the permissions of
 31746 *file* do not permit writing and the standard input is a terminal or the **-i** option is specified,
 31747 write a prompt to the standard error and read a line from the standard input. If the
 31748 response is not affirmative, do nothing more with the current file and go on to any
 31749 remaining files.
 - 31750 4. If the current file is a directory, *rm* shall perform actions equivalent to the *rmdir*()
 31751 function defined in the System Interfaces volume of IEEE Std. 1003.1-200x called with a path name
 31752 of the current file used as the *path* argument. If the current file is not a directory, *rm* shall
 31753 perform actions equivalent to the *unlink*() function defined in the System Interfaces
 31754 volume of IEEE Std. 1003.1-200x called with a path name of the current file used as the *path*
 31755 argument.
- 31756 If this fails for any reason, *rm* shall write a diagnostic message to standard error, do
 31757 nothing more with the current file, and go on to any remaining files.

31758 The *rm* utility shall be able to descend to arbitrary depths in a file hierarchy, and shall not fail
 31759 due to path length limitations (unless an operand specified by the user exceeds system
 31760 limitations).

31761 **OPTIONS**

31762 The *rm* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
31763 12.2, Utility Syntax Guidelines.

31764 The following options shall be supported:

31765 **-f** Do not prompt for confirmation. Do not write diagnostic messages or modify the
31766 exit status in the case of nonexistent operands. Any previous occurrences of the **-i**
31767 option shall be ignored.

31768 **-i** Prompt for confirmation as described previously. Any previous occurrences of the
31769 **-f** option shall be ignored.

31770 **-R** Remove file hierarchies. See the DESCRIPTION.

31771 **-r** Equivalent to **-R**.

31772 **OPERANDS**

31773 The following operand shall be supported:

31774 *file* A path name of a directory entry to be removed.

31775 **STDIN**

31776 Used to read an input line in response to each prompt specified in the STDOUT section.
31777 Otherwise, the standard input shall not be used.

31778 **INPUT FILES**

31779 None.

31780 **ENVIRONMENT VARIABLES**

31781 The following environment variables shall affect the execution of *rm*:

31782 *LANG* Provide a default value for the internationalization variables that are unset or null.
31783 If *LANG* is unset or null, the corresponding value from the implementation-
31784 defined default locale shall be used. If any of the internationalization variables
31785 contains an invalid setting, the utility shall behave as if none of the variables had
31786 been defined.

31787 *LC_ALL* If set to a non-empty string value, override the values of all the other
31788 internationalization variables.

31789 *LC_COLLATE*

31790 Determine the locale for the behavior of ranges, equivalence classes, and multi-
31791 character collating elements used in the extended regular expression defined for
31792 the **yesexpr** locale keyword in the *LC_MESSAGES* category.

31793 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
31794 characters (for example, single-byte as opposed to multi-byte characters in
31795 arguments) and the behavior of character classes within regular expressions used
31796 in the extended regular expression defined for the **yesexpr** locale keyword in the
31797 *LC_MESSAGES* category.

31798 *LC_MESSAGES*

31799 Determine the locale for the processing of affirmative responses that should be
31800 used to affect the format and contents of diagnostic messages written to standard
31801 error.

31802 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

31803 **ASYNCHRONOUS EVENTS**

31804 Default.

31805 **STDOUT**

31806 Not used.

31807 **STDERR**

31808 Prompts shall be written to standard error under the conditions specified in the DESCRIPTION
31809 and OPTIONS sections. The prompts shall contain the *file* path name, but their format is
31810 otherwise unspecified. The standard error also shall be used for diagnostic messages.

31811 **OUTPUT FILES**

31812 None.

31813 **EXTENDED DESCRIPTION**

31814 None.

31815 **EXIT STATUS**

31816 The following exit values shall be returned:

31817 0 All of the named directory entries for which *rm* performed actions equivalent to *rmdir()* or
31818 *unlink()* functions were removed.

31819 >0 An error occurred.

31820 **CONSEQUENCES OF ERRORS**

31821 Default.

31822 **APPLICATION USAGE**

31823 The *rm* utility is forbidden to remove the names dot and dot-dot in order to avoid the
31824 consequences of inadvertently doing something like:

31825 `rm -r .*`

31826 Some systems do not permit the removal of the last link to an executable binary file that is being
31827 executed; see the [EBUSY] error in the *unlink()* function defined in the System Interfaces volume
31828 of IEEE Std. 1003.1-200x. Thus, the *rm* utility can fail to remove such files.

31829 The *-i* option causes *rm* to prompt and read the standard input even if the standard input is not
31830 a terminal, but in the absence of *-i* the mode prompting is not done when the standard input is
31831 not a terminal.

31832 **EXAMPLES**

31833 1. The following command:

31834 `rm a.out core`31835 removes the directory entries: **a.out** and **core**.

31836 2. The following command:

31837 `rm -Rf junk`31838 removes the directory **junk** and all its contents, without prompting.31839 **RATIONALE**

31840 The *-i* option causes *rm* to prompt and read the standard input even if the standard input is not
31841 a terminal, but, in the absence of *-i*, the mode prompting is not done when the standard input is
31842 not a terminal.

31843 For absolute clarity, paragraphs (2b) and (3) in the DESCRIPTION of *rm* describing the behavior
31844 when prompting for confirmation, should be interpreted in the following manner:

```
31845     if ((NOT f_option) AND
31846         ((not_writable AND input_is_terminal) OR i_option))
```

31847 The exact format of the interactive prompts is unspecified. Only the general nature of the
31848 contents of prompts are specified because implementations may desire more descriptive
31849 prompts than those used on historical implementations. Therefore, an application not using the
31850 `-f` option, or using the `-i` option, relies on the system to provide the most suitable dialog directly
31851 with the user, based on the behavior specified.

31852 The `-r` option is historical practice on all known systems. The synonym `-R` option is provided
31853 for consistency with the other utilities in this volume of IEEE Std. 1003.1-200x that provide
31854 options requesting recursive descent through the file hierarchy.

31855 The behavior of the `-f` option in historical versions of *rm* is inconsistent. In general, along with
31856 “forcing” the unlink without prompting for permission, it always causes diagnostic messages to
31857 be suppressed and the exit status to be unmodified for nonexistent operands and files that
31858 cannot be unlinked. In some versions, however, the `-f` option suppresses usage messages and
31859 system errors as well. Suppressing such messages is not a service to either shell scripts or users.

31860 It is less clear that error messages regarding files that cannot be unlinked (removed) should be
31861 suppressed. Although this is historical practice, this volume of IEEE Std. 1003.1-200x does not
31862 permit the `-f` option to suppress such messages.

31863 When given the `-r` and `-i` options, historical versions of *rm* prompt the user twice for each
31864 directory, once before removing its contents and once before actually attempting to delete the
31865 directory entry that names it. This allows the user to “prune” the file hierarchy walk. Historical
31866 versions of *rm* were inconsistent in that some did not do the former prompt for directories
31867 named on the command line and others had obscure prompting behavior when the `-i` option
31868 was specified and the permissions of the file did not permit writing. The POSIX Shell and
31869 Utilities *rm* differs little from historic practice, but does require that prompts be consistent.
31870 Historical versions of *rm* were also inconsistent in that prompts were done to both standard
31871 output and standard error. This volume of IEEE Std. 1003.1-200x requires that prompts be done
31872 to standard error, for consistency with *cp* and *mv*, and to allow historical extensions to *rm* that
31873 provide an option to list deleted files on standard output.

31874 The *rm* utility is required to descend to arbitrary depths so that any file hierarchy may be
31875 deleted. This means, for example, that the *rm* utility cannot run out of file descriptors during its
31876 descent (that is, if the number of file descriptors is limited, *rm* cannot be implemented in the
31877 historical fashion where one file descriptor is used per directory level). Also, *rm* is not permitted
31878 to fail because of path length restrictions, unless an operand specified by the user is longer than
31879 {PATH_MAX}.

31880 The *rm* utility removes symbolic links themselves, not the files they refer to, as a consequence of
31881 the dependence on the `unlink()` functionality, per the DESCRIPTION. When removing
31882 hierarchies with `-r` or `-R`, the prohibition on following symbolic links has to be made explicit.

31883 FUTURE DIRECTIONS

31884 None.

31885 SEE ALSO

31886 *rmdir*, the System Interfaces volume of IEEE Std. 1003.1-200x, `remove()`, `unlink()`

31887 CHANGE HISTORY

31888 First released in Issue 2.

31889 Issue 4

31890 Aligned with the ISO/IEC 9945-2: 1993 standard.

31891 Issue 5

31892 FUTURE DIRECTIONS section added.

31893 Issue 6

31894 Text is added to clarify actions relating to symbolic links as specified in the IEEE P1003.2b draft
31895 standard.

31896 **NAME**31897 rmdel — remove a delta from an SCCS file (**DEVELOPMENT**)31898 **SYNOPSIS**31899 XSI `rmdel -r SID file...`

31900

31901 **DESCRIPTION**

31902 The *rmdel* utility shall remove the delta specified by the SID from each named SCCS file. The
 31903 delta to be removed shall be the most recent delta in its branch in the delta chain of each named
 31904 SCCS file. In addition, the application shall ensure that the SID specified is not that of a version
 31905 being edited for the purpose of making a delta; that is, if a *p-file* (see *get* (on page 2685)) exists for
 31906 the named SCCS file, the SID specified shall not appear in any entry of the *p-file*.

31907 Removal of a delta shall be restricted to:

- 31908 1. The user who made the delta
- 31909 2. The owner of the SCCS file
- 31910 3. The owner of the directory containing the SCCS file

31911 **OPTIONS**

31912 The *rmdel* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 31913 12.2, Utility Syntax Guidelines.

31914 The following option shall be supported:

31915 **-r** *SID* Specify the SCCS identification string (*SID*) of the delta to be deleted.31916 **OPERANDS**

31917 The following operand shall be supported:

31918 *file* A path name of an existing SCCS file or a directory. If *file* is a directory, the *rmdel*
 31919 utility shall behave as though each file in the directory were specified as a named
 31920 file, except that non-SCCS files (last component of the path name does not begin
 31921 with *s.*) and unreadable files shall be silently ignored.

31922 If exactly one *file* operand appears, and it is '-', the standard input shall be read;
 31923 each line of the standard input is taken to be the name of an SCCS file to be
 31924 processed. Non-SCCS files and unreadable files shall be silently ignored.

31925 **STDIN**

31926 The standard input shall be a text file used only when the *file* operand is specified as '-'. Each
 31927 line of the text file shall be interpreted as an SCCS path name.

31928 **INPUT FILES**

31929 The SCCS files are files of unspecified format.

31930 **ENVIRONMENT VARIABLES**31931 The following environment variables shall affect the execution of *rmdel*:

31932 *LANG* Provide a default value for the internationalization variables that are unset or null.
 31933 If *LANG* is unset or null, the corresponding value from the implementation-
 31934 defined default locale shall be used. If any of the internationalization variables
 31935 contains an invalid setting, the utility shall behave as if none of the variables had
 31936 been defined.

31937 *LC_ALL* If set to a non-empty string value, override the values of all the other
 31938 internationalization variables.

- 31939 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
31940 characters (for example, single-byte as opposed to multi-byte characters in
31941 arguments and input files).
- 31942 *LC_MESSAGES*
31943 Determine the locale that should be used to affect the format and contents of
31944 diagnostic messages written to standard error.
- 31945 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 31946 **ASYNCHRONOUS EVENTS**
31947 Default.
- 31948 **STDOUT**
31949 Not used.
- 31950 **STDERR**
31951 Used only for diagnostic messages.
- 31952 **OUTPUT FILES**
31953 The SCCS files are files of unspecified format. During processing of a *file*, a temporary *x-file*, as
31954 described in *admin* (on page 2340), may be created and deleted; a locking *z-file*, as described in
31955 *get* (on page 2685), may be created and deleted.
- 31956 **EXTENDED DESCRIPTION**
31957 None.
- 31958 **EXIT STATUS**
31959 The following exit values shall be returned:
31960 0 Successful completion.
31961 >0 An error occurred.
- 31962 **CONSEQUENCES OF ERRORS**
31963 Default.
- 31964 **APPLICATION USAGE**
31965 None.
- 31966 **EXAMPLES**
31967 None.
- 31968 **RATIONALE**
31969 None.
- 31970 **FUTURE DIRECTIONS**
31971 None.
- 31972 **SEE ALSO**
31973 *delta*, *get*, *prs*
- 31974 **CHANGE HISTORY**
31975 First released in Issue 2.
- 31976 **Issue 4**
31977 Format reorganized.
31978 Utility Syntax Guidelines support mandated.
31979 Internationalized environment variable support mandated.

31980 **Issue 6**

31981 The normative text is reworded to avoid use of the term “must” for application requirements. |

31982 The normative text is reworded to emphasise the term “shall” for implementation requirements. |

31983 **NAME**

31984 rmdir — remove directories

31985 **SYNOPSIS**31986 rmdir [-p] *dir...*31987 **DESCRIPTION**31988 The *rmdir* utility shall remove the directory entry specified by each *dir* operand, which, in order
31989 to succeed, the application shall ensure refers to an empty directory.31990 Directories shall be processed in the order specified. If a directory and a subdirectory of that
31991 directory are specified in a single invocation of the *rmdir* utility, the application shall specify the
31992 subdirectory before the parent directory so that the parent directory will be empty when the
31993 *rmdir* utility tries to remove it.31994 **OPTIONS**31995 The *rmdir* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
31996 12.2, Utility Syntax Guidelines.

31997 The following option shall be supported:

31998 **-p** Remove all directories in a path name. For each *dir* operand:

- 31999 1. The directory entry it names shall be removed.
-
- 32000 2. If the
- dir*
- operand includes more than one path name component, effects
-
- 32001 equivalent to the following command shall occur:

32002 rmdir -p \$(dirname *dir*)32003 **OPERANDS**

32004 The following operand shall be supported:

32005 *dir* A path name of an empty directory to be removed.32006 **STDIN**

32007 Not used.

32008 **INPUT FILES**

32009 None.

32010 **ENVIRONMENT VARIABLES**32011 The following environment variables shall affect the execution of *rmdir*:32012 **LANG** Provide a default value for the internationalization variables that are unset or null.
32013 If *LANG* is unset or null, the corresponding value from the implementation-
32014 defined default locale shall be used. If any of the internationalization variables
32015 contains an invalid setting, the utility shall behave as if none of the variables had
32016 been defined.32017 **LC_ALL** If set to a non-empty string value, override the values of all the other
32018 internationalization variables.32019 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
32020 characters (for example, single-byte as opposed to multi-byte characters in
32021 arguments).32022 **LC_MESSAGES**32023 Determine the locale that should be used to affect the format and contents of
32024 diagnostic messages written to standard error.

- 32025 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 32026 **ASYNCHRONOUS EVENTS**
- 32027 Default.
- 32028 **STDOUT**
- 32029 Not used.
- 32030 **STDERR**
- 32031 Used only for diagnostic messages.
- 32032 **OUTPUT FILES**
- 32033 None.
- 32034 **EXTENDED DESCRIPTION**
- 32035 None.
- 32036 **EXIT STATUS**
- 32037 The following exit values shall be returned:
- 32038 0 Each directory entry specified by a *dir* operand was removed successfully.
- 32039 >0 An error occurred.
- 32040 **CONSEQUENCES OF ERRORS**
- 32041 Default.
- 32042 **APPLICATION USAGE**
- 32043 The definition of an empty directory is one that contains, at most, directory entries for dot and dot-dot.
- 32044
- 32045 **EXAMPLES**
- 32046 If a directory **a** in the current directory is empty except it contains a directory **b** and **a/b** is empty
- 32047 except it contains a directory **c**:
- 32048 `rmdir -p a/b/c`
- 32049 removes all three directories.
- 32050 **RATIONALE**
- 32051 On historical System V systems, the **-p** option also caused a message to be written to the
- 32052 standard output. The message indicated whether the whole path was removed or whether part
- 32053 of the path remained for some reason. The **STDERR** section requires this diagnostic when the
- 32054 entire path specified by a *dir* operand is not removed, but does not allow the status message
- 32055 reporting success to be written as a diagnostic.
- 32056 The *rmdir* utility on System V also included an **-s** option that suppressed the informational
- 32057 message output by the **-p** option. This option has been omitted because the informational
- 32058 message is not specified by this volume of IEEE Std. 1003.1-200x.
- 32059 **FUTURE DIRECTIONS**
- 32060 None.
- 32061 **SEE ALSO**
- 32062 *rm*, the System Interfaces volume of IEEE Std. 1003.1-200x, *remove()*, *rmdir()*, *unlink()*
- 32063 **CHANGE HISTORY**
- 32064 First released in Issue 2.

32065 **Issue 4**

32066

Separated from the *rm* description and aligned with the ISO/IEC 9945-2: 1993 standard.

32067 **Issue 6**

32068

The normative text is reworded to avoid use of the term “must” for application requirements.

32069 **NAME**32070 **sact** — print current SCCS file-editing activity (**DEVELOPMENT**)32071 **SYNOPSIS**32072 XSI `sact file...`

32073

32074 **DESCRIPTION**

32075 The *sact* utility shall inform the user of any impending deltas to a named SCCS file by writing a
 32076 list to standard output. This situation occurs when *get -e* has been executed previously without
 32077 a subsequent execution of *delta*.

32078 **OPTIONS**

32079 None.

32080 **OPERANDS**

32081 The following operand shall be supported:

32082 *file* A path name of an existing SCCS file or a directory. If *file* is a directory, the *sact*
 32083 utility shall behave as though each file in the directory were specified as a named
 32084 file, except that non-SCCS files (last component of the path name does not begin
 32085 with *s*.) and unreadable files shall be silently ignored.

32086 If a single instance *file* is specified as *'-'*, the standard input shall be read; each
 32087 line of the standard input shall be taken to be the name of an SCCS file to be
 32088 processed. Non-SCCS files and unreadable files shall be silently ignored.

32089 **STDIN**

32090 The standard input shall be a text file used only when the *file* operand is specified as *'-'*. Each
 32091 line of the text file shall be interpreted as an SCCS path name.

32092 **INPUT FILES**

32093 Any SCCS files interrogated are files of an unspecified format.

32094 **ENVIRONMENT VARIABLES**32095 The following environment variables shall affect the execution of *sact*:

32096 *LANG* Provide a default value for the internationalization variables that are unset or null.
 32097 If *LANG* is unset or null, the corresponding value from the implementation-
 32098 defined default locale shall be used. If any of the internationalization variables
 32099 contains an invalid setting, the utility shall behave as if none of the variables had
 32100 been defined.

32101 *LC_ALL* If set to a non-empty string value, override the values of all the other
 32102 internationalization variables.

32103 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 32104 characters (for example, single-byte as opposed to multi-byte characters in
 32105 arguments and input files).

32106 *LC_MESSAGES*

32107 Determine the locale that should be used to affect the format and contents of
 32108 diagnostic messages written to standard error.

32109 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

32110 **ASYNCHRONOUS EVENTS**

32111 Default.

32112 **STDOUT**

32113 The output for each named file shall consist of a line in the following format:

32114 "%sΔ%sΔ%sΔ%sΔ%s\n", <SID>, <new SID>, <login>, <date>, <time>

32115 <SID> Specifies the SID of a delta that currently exists in the SCCS file to which changes
32116 are made to make the new delta.

32117 <new SID> Specifies the SID for the new delta to be created.

32118 <login> Contains the login name of the user who makes the delta (that is, who executed a
32119 *get* for editing).32120 <date> Contains the date that *get -e* was executed, in the format used by the *prs :D:* data
32121 keyword.32122 <time> Contains the time that *get -e* was executed, in the format used by the *prs :T:* data
32123 keyword.32124 If there is more than one named file or if a directory or standard input is named, each path name
32125 shall be written before each of the preceding lines:

32126 "\n%s:\n", <pathname>

32127 **STDERR**32128 Used only for optional informative messages concerning SCCS files with no impending deltas,
32129 and for diagnostic messages.32130 **OUTPUT FILES**

32131 None.

32132 **EXTENDED DESCRIPTION**

32133 None.

32134 **EXIT STATUS**

32135 The following exit values shall be returned:

32136 0 Successful completion.

32137 >0 An error occurred.

32138 **CONSEQUENCES OF ERRORS**

32139 Default.

32140 **APPLICATION USAGE**

32141 None.

32142 **EXAMPLES**

32143 None.

32144 **RATIONALE**

32145 None.

32146 **FUTURE DIRECTIONS**

32147 None.

32148 **SEE ALSO**32149 *delta, get, unget*32150 **CHANGE HISTORY**

32151 First released in Issue 2.

32152 **Issue 4**

32153 Format reorganized.

32154 Utility Syntax Guidelines support mandated.

32155 Internationalized environment variable support mandated.

32156 **Issue 4, Version 2**32157 The `STDERR` section encompasses informative messages concerning SCCS files with no
32158 impending deltas.32159 **Issue 6**

32160 The normative text is reworded to emphasise the term “shall” for implementation requirements.

32161 NAME

32162 `sccs` — front end for the SCCS subsystem (**DEVELOPMENT**)

32163 SYNOPSIS

32164 XSI `sccs [-r][-d path][-p path] command [options...][operands...]`

32165

32166 DESCRIPTION

32167 The `sccs` utility is a front end to the SCCS programs. It also includes the capability to run set-
32168 user-id to another user to provide additional protection.32169 The `sccs` utility shall invoke the specified *command* with the specified *options* and *operands*. By
32170 default, each of the *operands* shall be modified by prefixing it with the string **SCCS/s.**32171 The *command* can be the name of one of the SCCS utilities in this volume of IEEE Std. 1003.1-200x
32172 (*admin*, *delta*, *get*, *prs*, *rmdel*, *sact*, *unget*, *val*, or *what*) or one of the pseudo-utilities listed in the
32173 EXTENDED DESCRIPTION section.

32174 OPTIONS

32175 The `sccs` utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
32176 12.2, Utility Syntax Guidelines, except that *options* operands are actually options to be passed to
32177 the utility named by *command*. When the portion of the command:32178 `command [options ...] [operands ...]`32179 is considered, all of the pseudo-utilities used as *command* shall support the Utility Syntax
32180 Guidelines. Any of the other SCCS utilities that can be invoked in this manner support the
32181 Guidelines to the extent indicated by their individual OPTIONS sections.32182 The following options shall be supported preceding the *command* operand:32183 **-d path** A path name of a directory to be used as a root directory for the SCCS files. The
32184 default is the current directory. The **-d** option takes precedence over the
32185 **PROJECTDIR** variable. See **-p**.32186 **-p path** A path name of a directory in which the SCCS files are located. The default is the
32187 **SCCS** directory.32188 The **-p** option differs from the **-d** option in that the **-d** option-argument is
32189 prefixed to the entire path name and the **-p** option-argument is inserted before the
32190 final component of the path name. For example:32191 `sccs -d /x -p y get a/b`

32192 converts to:

32193 `get /x/a/y/s.b`

32194 This allows the creation of aliases such as:

32195 `alias syssccs="sccs -d /usr/src"`

32196 which is used as:

32197 `syssccs get cmd/who.c`32198 **-r** Invoke *command* with the real user ID of the process, not any effective user ID that
32199 the `sccs` utility is set to. Certain commands (*admin*, **check**, **clean**, **diffs**, **info**, *rmdel*,
32200 and **tell**) cannot be run set-user-ID by all users, since this would allow anyone to
32201 change the authorizations. These commands are always run as the real user.

32202 **OPERANDS**

32203 The following operands shall be supported:

32204 *command* An SCCS utility name or the name of one of the pseudo-utilities listed in the
32205 EXTENDED DESCRIPTION section.

32206 *options* An option or option-argument to be passed to *command*.

32207 *operands* An operand to be passed to *command*.

32208 **STDIN**

32209 See the utility description for the specified *command*.

32210 **INPUT FILES**

32211 See the utility description for the specified *command*.

32212 **ENVIRONMENT VARIABLES**

32213 The following environment variables shall affect the execution of *sccs*:

32214 *LANG* Provide a default value for the internationalization variables that are unset or null.
32215 If *LANG* is unset or null, the corresponding value from the implementation-
32216 defined default locale shall be used. If any of the internationalization variables
32217 contains an invalid setting, the utility shall behave as if none of the variables had
32218 been defined.

32219 *LC_ALL* If set to a non-empty string value, override the values of all the other
32220 internationalization variables.

32221 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
32222 characters (for example, single-byte as opposed to multi-byte characters in
32223 arguments and input files).

32224 *LC_MESSAGES*

32225 Determine the locale that should be used to affect the format and contents of
32226 diagnostic messages written to standard error.

32227 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

32228 *PROJECTDIR*

32229 Provide a default value for the *-d path* option. If the value of *PROJECTDIR* begins
32230 with a slash, it shall be considered an absolute path name; otherwise, the value of
32231 *PROJECTDIR* is treated as a user name and that user's initial working directory
32232 shall be examined for a subdirectory *src* or *source*. If such a directory is found, it
32233 shall be used. Otherwise, the value shall be used as a relative path name.

32234 Additional environment variable effects may be found in the utility description for the specified
32235 *command*.

32236 **ASYNCHRONOUS EVENTS**

32237 Default.

32238 **STDOUT**

32239 See the utility description for the specified *command*.

32240 **STDERR**

32241 See the utility description for the specified *command*.

32242 **OUTPUT FILES**

32243 See the utility description for the specified *command*.

32244 **EXTENDED DESCRIPTION**

32245 The following pseudo-utilities shall be supported as *command* operands. All options referred to
32246 in the following list are values given in the *options* operands following *command*.

32247 **check** Equivalent to **info**, except that nothing shall be printed if nothing is being edited, and a
32248 non-zero exit status shall be returned if anything is being edited. The intent is to have
32249 this included in an “install” entry in a makefile to ensure that everything is included
32250 into the SCCS file before a version is installed.

32251 **clean** Remove everything from the current directory that can be recreated from SCCS files,
32252 but do not remove any files being edited. If the **-b** option is given, branches shall be
32253 ignored in the determination of whether they are being edited; this is dangerous if
32254 branches are kept in the same directory.

32255 **create** Create an SCCS file, taking the initial contents from the file of the same name. Any
32256 options to *admin* are accepted. If the creation is successful, the original files shall be
32257 renamed by prefixing the basenames with a comma. These renamed files should be
32258 removed after it has been verified that the SCCS files have been created successfully.

32259 **delget** Perform a *delta* on the named files and then *get* new versions. The new versions shall
32260 have ID keywords expanded and shall not be editable. Any **-m**, **-p**, **-r**, **-s**, and **-y**
32261 options shall be passed to *delta*, and any **-b**, **-c**, **-e**, **-i**, **-k**, **-l**, **-s**, and **-x** options shall be
32262 passed to *get*.

32263 **deledit** Equivalent to **delget**, except that the *get* phase shall include the **-e** option. This option
32264 is useful for making a checkpoint of the current editing phase. The same options are
32265 passed to *delta* as described above, and all the options listed for *get* above except **-e** are
32266 passed to **edit**.

32267 **diffs** Write a difference listing between the current version of the files checked out for
32268 editing and the versions in SCCS format. Any **-r**, **-c**, **-i**, **-x**, and **-t** options shall be
32269 passed to *get*; any **-l**, **-s**, **-e**, **-f**, **-h**, and **-b** options shall be passed to *diff*. A **-C** option
32270 shall be passed to *diff* as **-c**.

32271 **edit** Equivalent to *get -e*.

32272 **fix** Remove the named delta, but leave a copy of the delta with the changes that were in it.
32273 It is useful for fixing small compiler bugs, and so on. The application shall ensure that it
32274 is followed by a **-r** *SID* option. Since **fix** doesn't leave audit trails, it should be used
32275 carefully.

32276 **info** Write a listing of all files being edited. If the **-b** option is given, branches (that is, *SIDs*
32277 with two or fewer components) shall be ignored. If a **-u** *user* option is given, then only
32278 files being edited by the named user shall be listed. A **-U** option shall be equivalent to
32279 **-u**<*current user*>.

32280 **print** Write out verbose information about the named files, equivalent to *sccs prs*.

32281 **tell** Write a <newline>-separated list of the files being edited to standard output. Takes the
32282 **-b**, **-u**, and **-U** options like **info** and **check**.

32283 **unedit** This is the opposite of an **edit** or a *get -e*. It should be used with caution, since any
32284 changes made since the *get* are lost.

32285 **EXIT STATUS**

32286 The following exit values shall be returned:

32287 0 Successful completion.

32288 >0 An error occurred.

32289 **CONSEQUENCES OF ERRORS**

32290 Default.

32291 **APPLICATION USAGE**

32292 Many of the SCCS utilities take directory names as operands as well as specific file names. The
 32293 pseudo-utilities supported by *sccs* are not described as having this capability, but are not
 32294 prohibited from doing so.

32295 **EXAMPLES**

32296 1. To get a file for editing, edit it and produce a new delta:

32297 `sccs get -e file.c`32298 `ex file.c`32299 `sccs delta file.c`

32300 2. To get a file from another directory:

32301 `sccs -p /usr/src/sccs/s. get cc.c`

32302 or:

32303 `sccs get /usr/src/sccs/s.cc.c`

32304 3. To make a delta of a large number of files in the current directory:

32305 `sccs delta *.c`

32306 4. To get a list of files being edited that are not on branches:

32307 `sccs info -b`

32308 5. To delta everything being edited by the current user:

32309 `sccs delta $(sccs tell -U)`

32310 6. In a makefile, to get source files from an SCCS file if it does not already exist:

32311 `SRCS = <list of source files>`32312 `$(SRCS):`32313 `sccs get $(REL) $@`32314 **RATIONALE**32315 SCCS and its associated utilities are part of the XSI Development Utilities option within the XSI
 32316 extension.

32317 SCCS is an abbreviation for Source Code Control System. It is a maintenance and enhancement
 32318 tracking tool. When a file is put under SCCS, the source code control system maintains the file
 32319 and, when changes are made, identifies and stores them in the file with the original source code
 32320 and/or documentation. As other changes are made, they too are identified and retained in the
 32321 file.

32322 Retrieval of the original and any set of changes is possible. Any version of the file as it develops
 32323 can be reconstructed for inspection or additional modification. History data can be stored with
 32324 each version, documenting why the changes were made, who made them, and when they were
 32325 made.

32326 **FUTURE DIRECTIONS**

32327 None.

32328 **SEE ALSO**32329 *admin, delta, get, make, prs, rmdel, sact, unget, val, what*32330 **CHANGE HISTORY**

32331 First released in Issue 4.

32332 **Issue 6**

32333 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from
32334 “otherwise, the home directory of a user of that name is examined” to “otherwise, the value of
32335 *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined”.

32336 The normative text is reworded to avoid use of the term “must” for application requirements. |

32337 The normative text is reworded to emphasise the term “shall” for implementation requirements. |

32338 **NAME**32339 `sed` — stream editor32340 **SYNOPSIS**32341 `sed [-n] script[file...]`32342 `sed [-n][-e script][-f script_file][file...]`32343 **DESCRIPTION**

32344 The *sed* utility is a stream editor that shall read one or more text files, make editing changes
 32345 according to a script of editing commands, and write the results to standard output. The script
 32346 shall be obtained from either the *script* operand string or a combination of the option-arguments
 32347 from the `-e script` and `-f script_file` options.

32348 **OPTIONS**

32349 The *sed* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 32350 12.2, Utility Syntax Guidelines, except that the order of presentation of the `-e` and `-f` options is
 32351 significant.

32352 The following options shall be supported:

32353 `-e script` Add the editing commands specified by the *script* option-argument to the end of
 32354 the script of editing commands. The *script* option-argument shall have the same
 32355 properties as the *script* operand, described in the OPERANDS section.

32356 `-f script_file` Add the editing commands in the file *script_file* to the end of the script.

32357 `-n` Suppress the default output (in which each line, after it is examined for editing, is
 32358 written to standard output). Only lines explicitly selected for output are written.

32359 Multiple `-e` and `-f` options may be specified. All commands shall be added to the script in the
 32360 order specified, regardless of their origin.

32361 **OPERANDS**

32362 The following operands shall be supported:

32363 *file* A path name of a file whose contents are read and edited. If multiple *file* operands
 32364 are specified, the named files shall be read in the order specified and the
 32365 concatenation shall be edited. If no *file* operands are specified, the standard input
 32366 shall be used.

32367 *script* A string to be used as the script of editing commands. The application shall not
 32368 present a *script* that violates the restrictions of a text file except that the final
 32369 character need not be a <newline> character.

32370 **STDIN**

32371 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
 32372 section.

32373 **INPUT FILES**

32374 The input files shall be text files. The *script_files* named by the `-f` option shall consist of editing
 32375 commands.

32376 **ENVIRONMENT VARIABLES**

32377 The following environment variables shall affect the execution of *sed*:

32378 *LANG* Provide a default value for the internationalization variables that are unset or null.
 32379 If *LANG* is unset or null, the corresponding value from the implementation-
 32380 defined default locale shall be used. If any of the internationalization variables
 32381 contains an invalid setting, the utility shall behave as if none of the variables had

- 32382 been defined.
- 32383 *LC_ALL* If set to a non-empty string value, override the values of all the other
32384 internationalization variables.
- 32385 *LC_COLLATE*
32386 Determine the locale for the behavior of ranges, equivalence classes, and multi-
32387 character collating elements within regular expressions.
- 32388 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
32389 characters (for example, single-byte as opposed to multi-byte characters in
32390 arguments and input files), and the behavior of character classes within regular
32391 expressions.
- 32392 *LC_MESSAGES*
32393 Determine the locale that should be used to affect the format and contents of
32394 diagnostic messages written to standard error.
- 32395 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 32396 **ASYNCHRONOUS EVENTS**
32397 Default.
- 32398 **STDOUT**
32399 The input files shall be written to standard output, with the editing commands specified in the
32400 script applied. If the *-n* option is specified, only those input lines selected by the script shall be
32401 written to standard output.
- 32402 **STDERR**
32403 Used only for diagnostic messages.
- 32404 **OUTPUT FILES**
32405 The output files shall be text files whose formats are dependent on the editing commands given.
- 32406 **EXTENDED DESCRIPTION**
32407 The *script* shall consist of editing commands of the following form:
32408 [*address* [, *address*]] *function*
- 32409 where *function* represents a single-character command verb from the list in **Editing Commands**
32410 in **sed** (on page 3053), followed by any applicable arguments.
- 32411 Zero or more <blank> characters shall be accepted before the first address and before function.
32412 Any number of semicolons shall be accepted before the first address.
- 32413 In default operation, *sed* cyclically shall copy a line of input, less its terminating <newline>, into
32414 a pattern space (unless there is something left after a **D** command), apply in sequence all
32415 commands whose addresses select that pattern space, and at the end of the script copy the
32416 pattern space to standard output (except when *-n* is specified) and delete the pattern space.
32417 Whenever the pattern space is written to standard output or a named file, *sed* shall immediately
32418 follow it with a <newline>.
- 32419 Some of the editing commands use a hold space to save all or part of the pattern space for
32420 subsequent retrieval. The pattern and hold spaces shall each be able to hold at least 8 192 bytes.

32421 **Addresses in sed**

32422 An address is either a decimal number that counts input lines cumulatively across files, a '\$' character that addresses the last line of input, or a context address (which consists of a BRE, as described in **Regular Expressions in sed**, preceded and followed by a delimiter, usually a slash).

32425 An editing command with no addresses shall select every pattern space.

32426 An editing command with one address shall select each pattern space that matches the address.

32427 An editing command with two addresses shall select the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line shall be selected.) Starting at the first line following the selected range, *sed* shall look again for the first address. Thereafter, the process shall be repeated. Omitting either or both of the address components in the following form produces undefined results:

32433 [*address* [, *address*]]

32434 **Regular Expressions in sed**

32435 The *sed* utility shall support the BREs described in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 9.3, Basic Regular Expressions, with the following additions:

- 32437 • In a context address, the construction "`\cBREc`", where *c* is any character other than
32438 backslash or <newline>, shall be identical to "`/BRE/`". If the character designated by *c*
32439 appears following a backslash, then it shall be considered to be that literal character, which
32440 shall not terminate the BRE. For example, in the context address "`\xabc\xdefx`", the
32441 second *x* stands for itself, so that the BRE is "`abcxdef`".
- 32442 • The escape sequence '`\n`' shall match a <newline> embedded in the pattern space. A literal
32443 <newline> character shall not be used in the BRE of a context address or in the substitute
32444 function.
- 32445 • If an RE is empty (that is, no pattern is specified) *sed* shall behave as if the last RE used in the
32446 last command applied (either as an address or as part of a substitute command) was
32447 specified.

32448 **Editing Commands in sed**

32449 In the following list of editing commands, the maximum number of permissible addresses for
32450 each function is indicated by [*0addr*], [*1addr*], or [*2addr*], representing zero, one, or two
32451 addresses.

32452 The argument *text* shall consist of one or more lines. Each embedded <newline> in the text shall
32453 be preceded by a backslash. Other backslashes in text shall be removed, and the following
32454 character shall be treated literally.

32455 The *r* and *w* command verbs, and the *w* flag to the *s* command, take an optional *rfile* (or *wfile*)
32456 parameter, separated from the command verb letter or flag by one or more <blank> characters;
32457 implementations may allow zero separation as an extension.

32458 The argument *rfile* or the argument *wfile* shall terminate the editing command. Each *wfile* shall be
32459 created before processing begins. Implementations shall support at least ten *wfile* arguments in
32460 the script; the actual number (greater than or equal to 10) that shall be supported by the
32461 implementation is unspecified. The use of the *wfile* parameter shall cause that file to be initially
32462 created, if it does not exist, or shall replace the contents of an existing file.

32463 The **b**, **r**, **s**, **t**, **w**, **y**, and **:** command verbs shall accept additional arguments. The following
 32464 synopses indicate which arguments shall be separated from the command verbs by a single
 32465 <space>.

32466 The **a** and **r** commands schedule text for later output. The text specified for the **a** command, and
 32467 the contents of the file specified for the **r** command, shall be written to standard output just
 32468 before the next attempt to fetch a line of input when executing the **N** or **n** commands, or when
 32469 reaching the end of the script. If written when reaching the end of the script, and the **-n** option
 32470 was not specified, the text shall be written after copying the pattern space to standard output.
 32471 The contents of the file specified for the **r** command shall be as of the time the output is written,
 32472 not the time the **r** command is applied. The text shall be output in the order in which the **a** and **r**
 32473 commands were applied to the input.

32474 Command verbs other than **{**, **a**, **b**, **c**, **i**, **r**, **t**, **w**, **:**, and **#** can be followed by a semicolon, optional
 32475 <blank> characters, and another command verb. However, when the **s** command verb is used
 32476 with the **w** flag, following it with another command in this manner produces undefined results.

32477 A function can be preceded by one or more **'!'** characters, in which case the function shall be
 32478 applied if the addresses do not select the pattern space. Zero or more <blank> characters shall be
 32479 accepted before the first **'!'** character. It is unspecified whether <blank> characters can follow a
 32480 **'!'** character, and conforming applications shall not follow a **'!'** character with <blank>
 32481 characters.

32482 **[2addr]{function**
 32483 **function**
 32484 ...
 32485 **}** Execute a list of *sed* functions only when the pattern space is selected. The list of
 32486 *sed* functions shall be surrounded by braces and separated by <newline>s, as
 32487 follows. The braces can be preceded or followed by <blank> characters. The
 32488 functions can be preceded by <blank> characters, but shall not be followed by
 32489 <blank> characters. The <right-brace> shall be preceded by a <newline> and can
 32490 be preceded or followed by <blank> characters.

32491 **[1addr]a**
 32492 **text** Write text to standard output as described previously.

32493 **[2addr]b [label]**
 32494 Branch to the **:** function bearing the *label*. If *label* is not specified, branch to the end
 32495 of the script. The implementation shall support *labels* recognized as unique up to
 32496 at least 8 characters; the actual length (greater than or equal to 8) that shall be
 32497 supported by the implementation is unspecified. It is unspecified whether
 32498 exceeding a label length causes an error or a silent truncation.

32499 **[2addr]c**
 32500 **text** Delete the pattern space. With a 0 or 1 address or at the end of a 2-address range,
 32501 place *text* on the output and start the next cycle.

32502 **[2addr]d** Delete the pattern space and start the next cycle.

32503 **[2addr]D** Delete the initial segment of the pattern space through the first <newline> and
 32504 start the next cycle.

32505 **[2addr]g** Replace the contents of the pattern space by the contents of the hold space.

32506 **[2addr]G** Append to the pattern space a <newline> followed by the contents of the hold
 32507 space.

32508	[2addr]h	Replace the contents of the hold space with the contents of the pattern space.
32509	[2addr]H	Append to the hold space a <newline> followed by the contents of the pattern space.
32510		
32511	[1addr]i\	
32512	<i>text</i>	Write <i>text</i> to standard output.
32513	[2addr]l	(The letter ell.) Write the pattern space to standard output in a visually unambiguous form. The characters listed in the Base Definitions volume of IEEE Std. 1003.1-200x, Table 5-1, Escape Sequences and Associated Actions ('\\', '\a', '\b', '\f', '\r', '\t', '\v') shall be written as the corresponding escape sequence; the '\n' in that table is not applicable. Non-printable characters not in that table shall be written as one three-digit octal number (with a preceding backslash) for each byte in the character (most significant byte first). If the size of a byte on the system is greater than 9 bits, the format used for non-printable characters is implementation-defined.
32514		
32515		
32516		
32517		
32518		
32519		
32520		
32521		
32522		Long lines shall be folded, with the point of folding indicated by writing a backslash followed by a <newline>; the length at which folding occurs is unspecified, but should be appropriate for the output device. The end of each line shall be marked with a '\$'.
32523		
32524		
32525		
32526	[2addr]n	Write the pattern space to standard output if the default output has not been suppressed, and replace the pattern space with the next line of input.
32527		
32528		If no next line of input is available, the n command verb shall branch to the end of the script and quit without starting a new cycle.
32529		
32530	[2addr]N	Append the next line of input to the pattern space, using an embedded <newline> character to separate the appended material from the original material. Note that the current line number changes.
32531		
32532		
32533		If no next line of input is available, the N command verb shall branch to the end of the script and quit without starting a new cycle or copying the pattern space to standard output.
32534		
32535		
32536	[2addr]p	Write the pattern space to standard output.
32537	[2addr]P	Write the pattern space, up to the first <newline>, to standard output.
32538	[1addr]q	Branch to the end of the script and quit without starting a new cycle.
32539	[1addr]r rfile	Copy the contents of <i>rfile</i> to standard output as described previously. If <i>rfile</i> does not exist or cannot be read, it shall be treated as if it were an empty file, causing no error condition.
32540		
32541		
32542	[2addr]s/BRE/replacement/flags	
32543		Substitute the replacement string for instances of the BRE in the pattern space. Any character other than backslash or <newline> can be used instead of a slash to delimit the BRE and the replacement. Within the BRE and the replacement, the BRE delimiter itself can be used as a literal character if it is preceded by a backslash.
32544		
32545		
32546		
32547		
32548		An ampersand ('&') appearing in the replacement shall be replaced by the string matching the BRE. The special meaning of '&' in this context can be suppressed by preceding it by a backslash. The characters "\n", where <i>n</i> is a digit, shall be replaced by the text matched by the corresponding backreference expression. For each backslash ('\')
32549		
32550		
32551		
32552		

32553 encountered in scanning **replacement** from beginning to end, the
 32554 backslash shall be discarded and the following character
 32555 shall lose its special meaning (if any). It is unspecified
 32556 what special meaning is given to any character other than
 32557 '&', '\', or digits.

32558 A line can be split by substituting a <newline> character
 32559 into it. The application shall escape the <newline> in the
 32560 replacement by preceding it by a backslash. A substitution
 32561 shall be considered to have been performed even if the
 32562 replacement string is identical to the string that it
 32563 replaces. Any backslash used to alter the default meaning of
 32564 a subsequent character shall be discarded from the BRE or the
 32565 replacement before evaluating the BRE or using the
 32566 replacement.

32567 The value of *flags* shall be zero or more of:

32568 **n** Substitute for the *n*th occurrence only of the BRE found within the
 32569 pattern space.

32570 **g** Globally substitute for all non-overlapping instances of the BRE
 32571 rather than just the first one. If both **g** and **n** are specified, the results
 32572 are unspecified.

32573 **p** Write the pattern space to standard output if a replacement was
 32574 made.

32575 **w wfile** Write. Append the pattern space to *wfile* if a replacement was made.
 32576 A conforming application shall precede the *wfile* argument with one
 32577 or more <blank> characters. If the *w* flag is not the last flag value
 32578 given in a concatenation of multiple flag values, the results are
 32579 undefined.

32580 **[2addr]t [label]**
 32581 Test. Branch to the : command verb bearing the *label* if any substitutions have been
 32582 made since the most recent reading of an input line or execution of a **t**. If *label* is
 32583 not specified, branch to the end of the script.

32584 **[2addr]w wfile**
 32585 Append (write) the pattern space to *wfile*.

32586 **[2addr]x** Exchange the contents of the pattern and hold spaces.

32587 **[2addr]y/string1/string2/**
 32588 Replace all occurrences of characters in *string1* with the corresponding characters
 32589 in *string2*. If a backslash followed by an 'n' appear in *string1* or *string2*, the two
 32590 characters shall be handled as a single <newline> character. If the number of
 32591 characters in *string1* and *string2* are not equal, or if any of the characters in *string1*
 32592 appear more than once, the results are undefined. Any character other than
 32593 backslash or <newline> can be used instead of slash to delimit the strings. If the
 32594 delimiter is not *n*, within *string1* and *string2*, the delimiter itself can be used as a
 32595 literal character if it is preceded by a backslash. If a backslash character is
 32596 immediately followed by a backslash character in *string1* or *string2*, the two
 32597 backslash characters shall be counted as a single literal backslash character. The
 32598 meaning of a backslash followed by any character that is not 'n', a backslash, or
 32599 the delimiter character is undefined.

32600 [0addr]:label Do nothing. This command bears a *label* to which the **b** and **t** commands branch.

32601 [1addr]= Write the following to standard output:

32602 "%d\n", <current line number>

32603 [0addr] Ignore this empty command.

32604 [0addr]# Ignore the '#' and the remainder of the line (treat them as a comment), with the single exception that if the first two characters in the script are "#n", the default output shall be suppressed; this shall be the equivalent of specifying **-n** on the command line.

32608 EXIT STATUS

32609 The following exit values shall be returned:

32610 0 Successful completion.

32611 >0 An error occurred.

32612 CONSEQUENCES OF ERRORS

32613 Default.

32614 APPLICATION USAGE

32615 Regular expressions match entire strings, not just individual lines, but a <newline> character is matched by '\n' in a *sed* RE; a <newline> character is not allowed by the general definition of regular expression in IEEE Std. 1003.1-200x. Also note that '\n' cannot be used to match a <newline> character at the end of an arbitrary input line; <newline> characters appear in the pattern space as a result of the **N** editing command.

32620 EXAMPLES

32621 This *sed* script simulates the BSD *cat -s* command, squeezing excess blank lines from standard input.

```
32623        sed -n '
32624        # Write non-empty lines.
32625        ./ {
32626            p
32627            d
32628        }
32629        # Write a single empty line, then look for more empty lines.
32630        /^$/    p
32631        # Get next line, discard the held <newline> (empty line),
32632        # and look for more empty lines.
32633        :Empty
32634        /^$/    {
32635            N
32636            s/./ /
32637            b Empty
32638        }
32639        # Write the non-empty line before going back to search
32640        # for the first in a set of empty lines.
32641        p
32642        '
```

32643 RATIONALE

32644 This volume of IEEE Std. 1003.1-200x requires implementations to support at least ten distinct
 32645 *wfiles*, matching historical practice on many implementations. Implementations are encouraged
 32646 to support more, but portable applications should not exceed this limit.

32647 The exit status codes specified here are different from those in System V. System V returns 2 for
 32648 garbled *sed* commands, but returns zero with its usage message or if the input file could not be
 32649 opened. The standard developers considered this to be a bug.

32650 The manner in which the **l** command writes non-printable characters was changed to avoid the
 32651 historical backspace-overstrike method, and other requirements to achieve unambiguous output
 32652 were added. See the RATIONALE for *ed* (on page 2546) for details of the format chosen, which is
 32653 the same as that chosen for *sed*.

32654 This volume of IEEE Std. 1003.1-200x requires implementations to provide pattern and hold
 32655 spaces of at least 8 192 bytes, larger than the 4 000 bytes spaces used by some historical
 32656 implementations, but less than the 20 480 bytes limit used in an early proposal. Implementations
 32657 are encouraged to allocate dynamically larger pattern and hold spaces as needed.

32658 The requirements for acceptance of <blank> and <space> characters in command lines has been
 32659 made more explicit than in early proposals to describe clearly the historical practice and to
 32660 remove confusion about the phrase “protect initial blanks [*sic*] and tabs from the stripping that
 32661 is done on every script line” that appears in much of the historical documentation of the *sed*
 32662 utility description of text. (Not all implementations are known to have stripped <blank>
 32663 characters from text lines, although they all have allowed leading <blank> characters preceding
 32664 the address on a command line.)

32665 The treatment of ‘#’ comments differs from the SVID which only allows a comment as the first
 32666 line of the script, but matches BSD-derived implementations. The comment character is treated
 32667 as a command, and it has the same properties in terms of being accepted with leading <blank>
 32668 characters; the BSD implementation has historically supported this.

32669 Early proposals required that a *script_file* have at least one non-comment line. Some historical
 32670 implementations have behaved in unexpected ways if this were not the case. The standard
 32671 developers considered that this was incorrect behavior and that application developers should
 32672 not have to avoid this feature. A correct implementation of this volume of IEEE Std. 1003.1-200x
 32673 shall permit *script_files* that consist only of comment lines.

32674 Early proposals indicated that if **-e** and **-f** options were intermixed, all **-e** options were
 32675 processed before any **-f** options. This has been changed to process them in the order presented
 32676 because it matches historical practice and is more intuitive.

32677 The treatment of the **p** flag to the **s** command differs between System V and BSD-based systems
 32678 when the default output is suppressed. In the two examples:

```
32679 echo a | sed 's/a/A/p'
```

```
32680 echo a | sed -n 's/a/A/p'
```

32681 This volume of IEEE Std. 1003.1-200x, BSD, System V documentation, and the SVID indicate that
 32682 the first example should write two lines with **A**, whereas the second should write one. Some
 32683 System V systems write the **A** only once in both examples because the **p** flag is ignored if the **-n**
 32684 option is not specified.

32685 This is a case of a diametrical difference between systems that could not be reconciled through
 32686 the compromise of declaring the behavior to be unspecified. The SVID/BSD/System V
 32687 documentation behavior was adopted for this volume of IEEE Std. 1003.1-200x because:

- 32688 • No known documentation for any historic system describes the interaction between the **p**
32689 flag and the **-n** option.
- 32690 • The selected behavior is more correct as there is no technical justification for any interaction
32691 between the **p** flag and the **-n** option. A relationship between **-n** and the **p** flag might imply
32692 that they are only used together, but this ignores valid scripts that interrupt the cyclical
32693 nature of the processing through the use of the **D**, **d**, **q**, or branching commands. Such scripts
32694 rely on the **p** suffix to write the pattern space because they do not make use of the default
32695 output at the “bottom” of the script.
- 32696 • Because the **-n** option makes the **p** flag unnecessary, any interaction would only be useful if
32697 *sed* scripts were written to run both with and without the **-n** option. This is believed to be
32698 unlikely. It is even more unlikely that programmers have coded the **p** flag expecting it to be
32699 unnecessary. Because the interaction was not documented, the likelihood of a programmer
32700 discovering the interaction and depending on it is further decreased.
- 32701 • Finally, scripts that break under the specified behavior produce too much output instead of
32702 too little, which is easier to diagnose and correct.
- 32703 The form of the substitute command that uses the **n** suffix was limited to the first 512 matches in
32704 an early proposal. This limit has been removed because there is no reason an editor processing
32705 lines of {LINE_MAX} length should have this restriction. The command **s/a/A/2047** should be
32706 able to substitute the 2 047th occurrence of **a** on a line.
- 32707 The **b**, **t**, and **:** commands are documented to ignore leading white space, but no mention is
32708 made of trailing white space. Historical implementations of *sed* assigned different locations to
32709 the labels '**x**' and "**x**". This is not useful, and leads to subtle programming errors, but it is
32710 historical practice, and changing it could theoretically break working scripts. Implementors are
32711 encouraged to provide warning messages about labels that are never used or jumps to labels
32712 that do not exist.
- 32713 Historically, the *sed* **!** and **}** editing commands did not permit multiple commands on a single
32714 line using a semicolon as a command delimiter. Implementations are permitted, but not
32715 required, to support this extension.
- 32716 **FUTURE DIRECTIONS**
- 32717 None.
- 32718 **SEE ALSO**
- 32719 *awk*, *ed*, *grep*
- 32720 **CHANGE HISTORY**
- 32721 First released in Issue 2.
- 32722 **Issue 4**
- 32723 Aligned with the ISO/IEC 9945-2: 1993 standard.
- 32724 **Issue 5**
- 32725 FUTURE DIRECTIONS section added.
- 32726 **Issue 6**
- 32727 The following new requirements on POSIX implementations derive from alignment with the
32728 Single UNIX Specification:
- 32729 • Implementations are required to support at least ten *wfile* arguments in an editing command.
- 32730 The EXTENDED DESCRIPTION is changed to align with the IEEE P1003.2b draft standard.

32731 NAME

32732 sh — shell, the standard command language interpreter

32733 SYNOPSIS

32734 sh [-abCefhimnuvx][-o option][+abCefhimnuvx][+o option]
 32735 [command_file [argument...]]

32736 sh -c[-abCefhimnuvx][-o option][+abCefhimnuvx][+o option]command_string
 32737 [command_name [argument...]]

32738 sh -s[-abCefhimnuvx][-o option][+abCefhimnuvx][+o option][argument]

32739 DESCRIPTION

32740 The *sh* utility is a command language interpreter that shall execute commands read from a
 32741 command line string, the standard input, or a specified file. The application shall ensure that the
 32742 commands to be executed are expressed in the language described in Chapter 2 (on page 2235).

32743 Path name expansion does not fail due to the size of a file.

32744 *Notes to Reviewers*

32745 *This section with side shading will not appear in the final copy. - Ed.*

32746 D3, XCU, ERN 215: Text here is unclear. There is nothing under the stat command that permits it
 32747 to fail on a very large file.

32748 Shell input and output redirections have an implementation-defined offset maximum that is
 32749 established in the open file description.

32750 OPTIONS

32751 The *sh* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,
 32752 Utility Syntax Guidelines, with an extension for support of a leading plus sign ('+') as noted
 32753 below.

32754 The **-a**, **-b**, **-C**, **-e**, **-f**, **-m**, **-n**, **-o option**, **-u**, **-v**, and **-x** options are described as part of the *set*
 32755 utility in Section 2.15 (on page 2276). The option letters derived from the *set* special built-in shall
 32756 also be accepted with a leading plus sign ('+') instead of a leading hyphen (meaning the reverse
 32757 case of the option as described in this volume of IEEE Std. 1003.1-200x).

32758 The following additional options shall be supported:

32759 **-c** Read commands from the *command_string* operand. Set the value of special
 32760 parameter 0 (see Section 2.5.2 (on page 2241)) from the value of the *command_name*
 32761 operand and the positional parameters (\$1, \$2, and so on) in sequence from the
 32762 remaining *argument* operands. No commands shall be read from the standard
 32763 input.

32764 **-i** Specify that the shell is *interactive*; see below. An implementation may treat
 32765 specifying the **-i** option as an error if the real user ID of the calling process does
 32766 not equal the effective user ID or if the real group ID does not equal the effective
 32767 group ID.

32768 **-s** Read commands from the standard input.

32769 If there are no operands and the **-c** option is not specified, the **-s** option shall be assumed.

32770 If the **-i** option is present, or if there are no operands and the shell's standard input and standard
 32771 error are attached to a terminal, the shell is considered to be *interactive*.

32772 **OPERANDS**

32773 The following operands shall be supported:

32774 – A single hyphen is treated as the first operand and then ignored. If both ‘-’ and
32775 “—” are given as arguments, or if other operands precede the single hyphen, the
32776 results are undefined.

32777 *argument* The positional parameters (\$1, \$2, and so on) shall be set to *arguments*, if any.

32778 *command_file* The path name of a file containing commands. If the path name contains one or
32779 more slash characters, the implementation attempts to read that file; the file need
32780 not be executable. If the path name does not contain a slash character:

32781 • The implementation shall attempt to read that file from the current working
32782 directory; the file need not be executable.

32783 • If the file is not in the current working directory, the implementation may
32784 perform a search for an executable file using the value of *PATH*, as described in
32785 Section 2.9.1.1 (on page 2257).

32786 Special parameter 0 (see Section 2.5.2 (on page 2241)) shall be set to the value of
32787 *command_file*. If *sh* is called using a synopsis form that omits *command_file*, special
32788 parameter 0 shall be set to the value of the first argument passed to *sh* from its
32789 parent (for example, *argv*[0] for a C program), which is normally a path name used
32790 to execute the *sh* utility.

32791 *command_name*

32792 A string assigned to special parameter 0 when executing the commands in
32793 *command_string*. If *command_name* is not specified, special parameter 0 shall be set
32794 to the value of the first argument passed to *sh* from its parent (for example, *argv*[0]
32795 for a C program), which is normally a path name used to execute the *sh* utility.

32796 *command_string*

32797 A string that shall be interpreted by the shell as one or more commands, as if the
32798 string were the argument to the *system()* function defined in the System Interfaces
32799 volume of IEEE Std. 1003.1-200x. If the *command_string* operand is an empty string,
32800 *sh* shall exit with a zero exit status.

32801 **STDIN**

32802 The standard input shall be used only if one of the following is true:

32803 • The *-s* option is specified.

32804 • The *-c* option is not specified and no operands are specified.

32805 • The script executes one or more commands that require input from standard input (such as a
32806 *read* command that does not redirect its input).

32807 See the INPUT FILES section.

32808 When the shell is using standard input and it invokes a command that also uses standard input,
32809 the shell shall ensure that the standard input file pointer points directly after the command it has
32810 read when the command begins execution. It shall not read ahead in such a manner that any
32811 characters intended to be read by the invoked command are consumed by the shell (whether
32812 interpreted by the shell or not) or that characters that are not read by the invoked command are
32813 not seen by the shell. When the command expecting to read standard input is started
32814 asynchronously by an interactive shell, it is unspecified whether characters are read by the
32815 command or interpreted by the shell.

32816 If the standard input to *sh* is a FIFO or terminal device and is set to non-blocking reads, then *sh*
 32817 shall enable blocking reads on standard input. This shall remain in effect when the command
 32818 completes.

32819 INPUT FILES

32820 The input file shall be a text file, except that line lengths shall be unlimited. If the input file is
 32821 empty or consists solely of blank lines or comments, or both, *sh* shall exit with a zero exit status.

32822 ENVIRONMENT VARIABLES

32823 The following environment variables shall affect the execution of *sh*:

32824 *ENV* This variable, when and only when an interactive shell is invoked, shall be
 32825 subjected to parameter expansion (see Section 2.6.2 (on page 2245)) by the shell,
 32826 and the resulting value shall be used as a path name of a file containing shell
 32827 commands to execute in the current environment. The file need not be executable.
 32828 If the expanded value of *ENV* is not an absolute path name, the results are
 32829 unspecified. *ENV* shall be ignored if the real and effective user IDs or real and
 32830 effective group IDs of the process are different.

32831 *FCEDIT* This variable, when expanded by the shell, determines the default value for the *-e*
 32832 *editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be used
 32833 as the editor. This volume of IEEE Std. 1003.1-200x specifies the effects of this
 32834 variable only for systems supporting the User Portability Utilities option.

32835 *HISTFILE* Determine a path name naming a command history file. If the *HISTFILE* variable is
 32836 not set, the shell may attempt to access or create a file *.sh_history* in the directory
 32837 referred to by the *HOME* environment variable. If the shell cannot obtain both read
 32838 and write access to, or create, the history file, it shall use an unspecified
 32839 mechanism that allows the history to operate properly. (References to history
 32840 "file" in this section shall be understood to mean this unspecified mechanism in
 32841 such cases.) An implementation may choose to access this variable only when
 32842 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt
 32843 to retrieve entries from, or add entries to, the file, as the result of commands issued
 32844 by the user, the file named by the *ENV* variable, or implementation-defined system
 32845 start-up files. (The initialization process for the history file can be dependent on the
 32846 system start-up files, in that they may contain commands that effectively preempt
 32847 the user's settings of *HISTFILE* and *HISTSIZE*. For example, function definition
 32848 commands are recorded in the history file, unless the *set -o nolog* option is set. If
 32849 the system administrator includes function definitions in some system start-up file
 32850 called before the *ENV* file, the history file is initialized before the user gets a chance
 32851 to influence its characteristics.) In some historical shells, the history file is
 32852 initialized just after the *ENV* file has been processed. Therefore, it is
 32853 implementation-defined whether changes made to *HISTFILE* after the history file
 32854 has been initialized are effective. Implementations may choose to disable the
 32855 history list mechanism for users with appropriate privileges who do not set
 32856 *HISTFILE*; the specific circumstances under which this occurs are
 32857 implementation-defined. If more than one instance of the shell is using the same
 32858 history file, it is unspecified how updates to the history file from those shells
 32859 interact. As entries are deleted from the history file, they shall be deleted oldest
 32860 first. It is unspecified when history file entries are physically removed from the
 32861 history file. This volume of IEEE Std. 1003.1-200x specifies the effects of this
 32862 variable only for systems supporting the User Portability Utilities option.

32863 *HISTSIZE* Determine a decimal number representing the limit to the number of previous
 32864 commands that are accessible. If this variable is unset, an unspecified default

32865		greater than or equal to 128 shall be used. The maximum number of commands in the history list is unspecified, but shall be at least 128. An implementation may choose to access this variable only when initializing the history file, as described under <i>HISTFILE</i> . Therefore, it is unspecified whether changes made to <i>HISTSIZE</i> after the history file has been initialized are effective.
32866		
32867		
32868		
32869		
32870	<i>HOME</i>	Determine the path name of the user's home directory. The contents of <i>HOME</i> are used in Tilde Expansion as described in Section 2.6.1 (on page 2244). This volume of IEEE Std. 1003.1-200x specifies the effects of this variable only for systems supporting the User Portability Utilities option.
32871		
32872		
32873		
32874	<i>IFS</i>	<i>Input field separators</i> : a string treated as a list of characters that shall be used for field splitting and to split lines into words with the <i>read</i> command. See Section 2.6.5 (on page 2249). If <i>IFS</i> is not set, the shell shall behave as if the value of <i>IFS</i> were the <space>, <tab>, and <newline> characters. Implementations may ignore the value of <i>IFS</i> in the environment at the time <i>sh</i> is invoked, treating <i>IFS</i> as if it were not set.
32875		
32876		
32877		
32878		
32879		
32880	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. If <i>LANG</i> is unset or null, the corresponding value from the implementation-defined default locale shall be used. If any of the internationalization variables contains an invalid setting, the utility shall behave as if none of the variables had been defined.
32881		
32882		
32883		
32884		
32885	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
32886		
32887	<i>LC_COLLATE</i>	Determine the behavior of range expressions, equivalence classes and multi-character collating elements within pattern matching.
32888		
32889		
32890	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), which characters are defined as letters (character class alpha), and the behavior of character classes within pattern matching.
32891		
32892		
32893		
32894	<i>LC_MESSAGES</i>	Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
32895		
32896		
32897	<i>MAIL</i>	Determine a path name of the user's mailbox file for purposes of incoming mail notification. If this variable is set, the shell shall inform the user if the file named by the variable is created or if its modification time has changed. Informing the user shall be accomplished by writing a string of unspecified format to standard error prior to the writing of the next primary prompt string after the completion of an interval defined by the <i>MAILCHECK</i> variable. The user shall be informed only if <i>MAIL</i> is set and <i>MAILPATH</i> is not set. This volume of IEEE Std. 1003.1-200x specifies the effects of this variable only for systems supporting the User Portability Utilities option.
32898		
32899		
32900		
32901		
32902		
32903		
32904		
32905		
32906	<i>MAILCHECK</i>	Establish a decimal integer value that specifies how often (in seconds) the shell shall check for the arrival of mail in the files specified by the <i>MAILPATH</i> or <i>MAIL</i> variables. The default value shall be 600 seconds. If set to zero, the shell shall check before issuing each primary prompt. This volume of IEEE Std. 1003.1-200x specifies the effects of this variable only for systems supporting the User Portability Utilities option.
32907		
32908		
32909		
32910		
32911		
32912		

- 32913 **MAILPATH** Provide a list of path names and optional messages separated by colons. If this
 32914 variable is set, the shell shall inform the user if any of the files named by the
 32915 variable are created or if any of their modification times change. (See the preceding
 32916 entry for *MAIL* for descriptions of mail arrival and user informing.) Each path
 32917 name can be followed by '%' and a string that shall be subjected to parameter
 32918 expansion and written to standard error when the modification time changes. If a
 32919 '%' character in the path name is preceded by a backslash, it shall be treated as a
 32920 literal '%' in the path name. The default message is unspecified.
- 32921 The *MAILPATH* environment variable takes precedence over the *MAIL* variable.
 32922 This volume of IEEE Std. 1003.1-200x specifies the effects of this variable only for
 32923 systems supporting the User Portability Utilities option.
- 32924 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 32925 **PATH** Establish a string formatted as described in the Base Definitions volume of
 32926 IEEE Std. 1003.1-200x, Chapter 8, Environment Variables, used to effect command
 32927 interpretation; see Section 2.9.1.1 (on page 2257).
- 32928 **PWD** This variable shall represent an absolute path name of the current working
 32929 directory. Assignments to this variable may be ignored unless the value is an
 32930 absolute path name of the current working directory and there are no file name
 32931 components of dot or dot-dot.
- 32932 **ASYNCHRONOUS EVENTS**
- 32933 Default.
- 32934 **STDOUT**
- 32935 See the *STDERR* section.
- 32936 **STDERR**
- 32937 Except as otherwise stated (by the descriptions of any invoked utilities or in interactive mode),
 32938 standard error is used only for diagnostic messages.
- 32939 **OUTPUT FILES**
- 32940 None.
- 32941 **EXTENDED DESCRIPTION**
- 32942 See Chapter 2. The following additional capabilities are supported on systems supporting the
 32943 User Portability Utilities option.
- 32944 **Command History List**
- 32945 When the *sh* utility is being used interactively, it shall maintain a list of commands previously
 32946 entered from the terminal in the file named by the *HISTFILE* environment variable. The type,
 32947 size, and internal format of this file are unspecified. Multiple *sh* processes can share access to the
 32948 file for a user, if file access permissions allow this; see the description of the *HISTFILE*
 32949 environment variable.

32950 **Command Line Editing**

32951 When *sh* is being used interactively from a terminal, the current command and the command
 32952 history (see *fc* (on page 2646)) can be edited using *vi*-mode command line editing. This mode
 32953 uses commands, described below, similar to a subset of those described in the *vi* utility.
 32954 Implementations may offer other command line editing modes corresponding to other editing
 32955 utilities.

32956 The command *set -o vi* shall enable *vi*-mode editing and place *sh* into *vi* insert mode (see
 32957 **Command Line Editing (vi-mode)**). This command also shall disable any other editing mode
 32958 that the implementation may provide. The command *set +o vi* disables *vi*-mode editing.

32959 Certain block-mode terminals may be unable to support shell command line editing. If a
 32960 terminal is unable to provide either edit mode, it need not be possible to *set -o vi* when using the
 32961 shell on this terminal.

32962 In the following sections, the characters *erase*, *interrupt*, *kill*, and *end-of-file* are those set by the
 32963 *stty* utility.

32964 **Command Line Editing (vi-mode)**

32965 With *vi*-mode enabled, *sh* can be switched between insert mode and command mode.

32966 When in insert mode, an entered character shall be inserted into the command line, except as
 32967 noted in **vi Line Editing Insert Mode**. Upon entering *sh* and after termination of the previous
 32968 command, *sh* shall be in insert mode.

32969 Typing an escape character shall switch *sh* into command mode (see **vi Line Editing Command**
 32970 **Mode** (on page 3066)). In command mode, an entered character shall either invoke a defined
 32971 operation, is used as part of a multi-character operation, or is treated as an error. A character that
 32972 is not recognized as part of an editing command shall terminate any specific editing command
 32973 and shall alert the terminal. Typing the *interrupt* character in command mode shall cause *sh* to
 32974 terminate command line editing on the current command line, reissue the prompt on the next
 32975 line of the terminal, and reset the command history (see *fc* (on page 2646)) so that the most
 32976 recently executed command is the previous command (that is, the command that was being
 32977 edited when it was interrupted is not reentered into the history).

32978 In the following sections, the phrase “move the cursor to the beginning of the word” shall mean
 32979 “move the cursor to the first character of the current word” and the phrase “move the cursor to
 32980 the end of the word” shall mean “move the cursor to the last character of the current word”. The
 32981 phrase “beginning of the command line” indicates the point between the end of the prompt
 32982 string issued by the shell (or the beginning of the terminal line, if there is no prompt string) and
 32983 the first character of the command text.

32984 **vi Line Editing Insert Mode**

32985 While in insert mode, any character typed shall be inserted in the current command line, unless
 32986 it is from the following set.

32987 <newline> Execute the current command line being edited.

32988 *erase* Delete the character previous to the current cursor position and move the current
 32989 cursor position back one character. In insert mode, characters shall be erased from
 32990 both the screen and the buffer when backspacing.

32991 *interrupt* Terminate command line editing with the same effects as described for
 32992 interrupting command mode; see **Command Line Editing (vi-mode)**.

32993	<i>kill</i>	Clear all the characters from the input line.
32994	<control>-V	Insert the next character input, even if the character is otherwise a special insert mode character.
32995		
32996	<control>-W	Delete the characters from the one preceding the cursor to the preceding word boundary. The word boundary in this case is the closer to the cursor of either the beginning of the line or a character that is in neither the blank nor punct character classification of the current locale.
32997		
32998		
32999		
33000	<i>end-of-file</i>	Interpreted as the end of input in <i>sh</i> . This interpretation shall occur only at the beginning of an input line. If <i>end-of-file</i> is entered other than at the beginning of the line, the results are unspecified.
33001		
33002		
33003	<ESC>	Place <i>sh</i> into command mode.
33004	vi Line Editing Command Mode	
33005		In command mode for the command line editing feature, decimal digits not beginning with 0 that precede a command letter shall be remembered. Some commands use these decimal digits as a count number that affects the operation.
33006		
33007		
33008		The term <i>motion command</i> represents one of the commands:
33009	<space>	0 b F l W ^ \$; E f T w , B e h t
33010		Any command that modifies the current line shall cause a copy of the current line to be made at the end of the command history, the current line shall become that copy, and the edit is performed on that copy.
33011		
33012		
33013		Any command that is preceded by <i>count</i> shall take a count (the numeric value of any preceding decimal digits). Unless otherwise noted, this count shall cause the specified operation to repeat by the number of times specified by the count. Also unless otherwise noted, a <i>count</i> that is out of range is considered an error condition and shall alert the terminal, but neither the cursor position, nor the command line, shall change.
33014		
33015		
33016		
33017		
33018		The terms <i>word</i> and <i>bigword</i> are used as defined in the <i>vi</i> description. The term <i>save buffer</i> corresponds to the term <i>unnamed buffer</i> in <i>vi</i> .
33019		
33020		The following commands shall be recognized in command mode:
33021	<newline>	Execute the current command line being edited.
33022	<control>-L	Redraw the current command line. Position the cursor at the same location on the new command line.
33023		
33024	#	Insert the character '#' at the beginning of the current command line and treat the current command line as a comment. This line shall be entered into the command history; see <i>fc</i> (on page 2646).
33025		
33026		
33027	=	Display the possible shell word expansions (see Section 2.6 (on page 2244)) of the bigword at the current command line position. These expansions shall be displayed on subsequent terminal lines. If the bigword contains none of the characters '?', '*', or '[', an asterisk('*') shall be implicitly assumed at the end. If any directories are matched, these expansions shall have a '/' character appended. After the expansion, the line shall be redrawn, the cursor is repositioned at the current cursor position, and <i>sh</i> shall be placed in command mode.
33028		
33029		
33030		
33031		
33032		
33033		
33034	\	Perform path name expansion (see Section 2.6.6 (on page 2249)) on the current bigword, up to the largest set of characters that can be matched uniquely. If the
33035		

33036 bigword contains none of the characters '?', '*', or '[', an asterisk ('*') shall
 33037 be implicitly assumed at the end. This maximal expansion then shall replace the
 33038 original bigword in the command line, and the cursor shall be placed after this
 33039 expansion. If the resulting bigword completely and uniquely matches a directory, a
 33040 '/' character shall be inserted directly after the bigword. If some other file is
 33041 completely matched, a single <space> character shall be inserted after the bigword.
 33042 After this operation, *sh* shall be placed in insert mode.

33043 * Perform path name expansion on the current bigword and insert all expansions
 33044 into the command to replace the current bigword, with each expansion separated
 33045 by a single <space> character. If at the end of the line, the current cursor position
 33046 shall be moved to the first column position following the expansions and *sh* shall
 33047 be placed in insert mode. Otherwise, the current cursor position shall be the last
 33048 column position of the first character after the expansions and *sh* shall be placed in
 33049 insert mode. If the current bigword contains none of the characters '?', '*', or
 33050 '[', before the operation, an asterisk shall be implicitly assumed at the end.

33051 @letter Insert the value of the alias named *_letter*. The symbol *letter* represents a single
 33052 alphabetic character from the portable character set; implementations may support
 33053 additional characters as an extension. If the alias *_letter* contains other editing
 33054 commands, these commands shall be performed as part of the insertion. If no alias
 33055 *_letter* is enabled, this command shall have no effect.

33056 [count]~ Convert, if the current character is a lowercase letter, to the equivalent uppercase
 33057 letter and *vice versa*, as prescribed by the current locale. The current cursor position
 33058 then shall be advanced by one character. If the cursor was positioned on the last
 33059 character of the line, the case conversion shall occur, but the cursor shall not
 33060 advance. If the '~' command is preceded by a *count*, that number of characters
 33061 shall be converted, and the cursor shall be advanced to the character position after
 33062 the last character converted. If the *count* is larger than the number of characters
 33063 after the cursor, this shall not be considered an error; the cursor shall advance to
 33064 the last character on the line.

33065 [count]. Repeat the most recent non-motion command, even if it was executed on an earlier
 33066 command line. If the previous command was preceded by a *count*, and no count is
 33067 given on the '.' command, the count from the previous command shall be
 33068 included as part of the repeated command. If the '.' command is preceded by a
 33069 *count*, this shall override any *count* argument to the previous command. The *count*
 33070 specified in the '.' command shall become the count for subsequent '.'
 33071 commands issued without a count.

33072 [number]v Invoke the *vi* editor to edit the current command line in a temporary file. When the
 33073 editor exits, the commands in the temporary file shall be executed. If a *number* is
 33074 prefixed to the command, it specifies the command number in the command
 33075 history to be edited, rather than the current command line.

33076 [count]l (ell)
 33077 [count]<space>
 33078 Move the current cursor position to the next character position. If the cursor was
 33079 positioned on the last character of the line, the terminal shall be alerted and the
 33080 cursor shall not be advanced. If the *count* is larger than the number of characters
 33081 after the cursor, this shall not be considered an error; the cursor shall advance to
 33082 the last character on the line.

33083 [count]h Move the current cursor position to the *count*th (default 1) previous character
 33084 position. If the cursor was positioned on the first character of the line, the terminal

33085		shall be alerted and the cursor shall not be moved. If the count is larger than the
33086		number of characters before the cursor, this shall not be considered an error; the
33087		cursor shall move to the first character on the line.
33088	[count]w	Move to the start of the next word. If the cursor was positioned on the last
33089		character of the line, the terminal shall be alerted and the cursor shall not be
33090		advanced. If the <i>count</i> is larger than the number of words after the cursor, this shall
33091		not be considered an error; the cursor shall advance to the last character on the
33092		line.
33093	[count]W	Move to the start of the next bigword. If the cursor was positioned on the last
33094		character of the line, the terminal shall be alerted and the cursor shall not be
33095		advanced. If the <i>count</i> is larger than the number of bigwords after the cursor, this
33096		shall not be considered an error; the cursor shall advance to the last character on
33097		the line.
33098	[count]e	Move to the end of the current word. If at the end of a word, move to the end of the
33099		next word. If the cursor was positioned on the last character of the line, the
33100		terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger
33101		than the number of words after the cursor, this shall not be considered an error; the
33102		cursor shall advance to the last character on the line.
33103	[count]E	Move to the end of the current bigword. If at the end of a bigword, move to the
33104		end of the next bigword. If the cursor was positioned on the last character of the
33105		line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i>
33106		is larger than the number of bigwords after the cursor, this shall not be considered
33107		an error; the cursor shall advance to the last character on the line.
33108	[count]b	Move to the beginning of the current word. If at the beginning of a word, move to
33109		the beginning of the previous word. If the cursor was positioned on the first
33110		character of the line, the terminal shall be alerted and the cursor shall not be
33111		moved. If the <i>count</i> is larger than the number of words preceding the cursor, this
33112		shall not be considered an error; the cursor shall return to the first character on the
33113		line.
33114	[count]B	Move to the beginning of the current bigword. If at the beginning of a bigword,
33115		move to the beginning of the previous bigword. If the cursor was positioned on the
33116		first character of the line, the terminal shall be alerted and the cursor shall not be
33117		moved. If the <i>count</i> is larger than the number of bigwords preceding the cursor,
33118		this shall not be considered an error; the cursor shall return to the first character on
33119		the line.
33120	^	Move the current cursor position to the first character on the input line that is not a
33121		<blank> character.
33122	\$	Move to the last character position on the current command line.
33123	0	(Zero.) Move to the first character position on the current command line.
33124	[count] 	Move to the <i>count</i> th character position on the current command line. If no number
33125		is specified, move to the first position. The first character position shall be
33126		numbered 1. If the count is larger than the number of characters on the line, this
33127		shall not be considered an error; the cursor shall be placed on the last character on
33128		the line.
33129	[count]fc	Move to the first occurrence of the character 'c' that occurs after the current
33130		cursor position. If the cursor was positioned on the last character of the line, the
33131		terminal shall be alerted and the cursor shall not be advanced. If the character 'c'

33132		does not occur in the line after the current cursor position, the terminal shall be
33133		alerted and the cursor shall not be moved.
33134	[count]Fc	Move to the first occurrence of the character 'c' that occurs before the current
33135		cursor position. If the cursor was positioned on the first character of the line, the
33136		terminal shall be alerted and the cursor shall not be moved. If the character 'c'
33137		does not occur in the line before the current cursor position, the terminal shall be
33138		alerted and the cursor shall not be moved.
33139	[count]tc	Move to the character before the first occurrence of the character 'c' that occurs
33140		after the current cursor position. If the cursor was positioned on the last character
33141		of the line, the terminal shall be alerted and the cursor shall not be advanced. If the
33142		character 'c' does not occur in the line after the current cursor position, the
33143		terminal shall be alerted and the cursor shall not be moved.
33144	[count]Tc	Move to the character after the first occurrence of the character 'c' that occurs
33145		before the current cursor position. If the cursor was positioned on the first
33146		character of the line, the terminal shall be alerted and the cursor shall not be
33147		moved. If the character 'c' does not occur in the line before the current cursor
33148		position, the terminal shall be alerted and the cursor shall not be moved.
33149	[count];	Repeat the most recent f , F , t , or T command. Any number argument on that
33150		previous command shall be ignored. Errors are those described for the repeated
33151		command.
33152	[count],	Repeat the most recent f , F , t , or T command. Any number argument on that
33153		previous command shall be ignored. However, reverse the direction of that
33154		command.
33155	a	Enter insert mode after the current cursor position. Characters that are entered
33156		shall be inserted before the next character.
33157	A	Enter insert mode after the end of the current command line.
33158	i	Enter insert mode at the current cursor position. Characters that are entered are
33159		inserted before the current character.
33160	I	Enter insert mode at the beginning of the current command line.
33161	R	Enter insert mode, replacing characters from the command line beginning at the
33162		current cursor position.
33163	[count]cmotion	
33164		Delete the characters between the current cursor position and the cursor position
33165		that would result from the specified <i>motion</i> command. Then enter insert mode
33166		before the first character following any deleted characters. If <i>count</i> is specified, it
33167		shall be applied to the motion command. A <i>count</i> shall be ignored for the following
33168		motion commands:
33169		0 ^ \$ c
33170		If the <i>motion</i> command is the character 'c', the current command line shall be
33171		cleared and insert mode shall be entered. If the <i>motion</i> command would move the
33172		current cursor position toward the beginning of the command line, the character
33173		under the current cursor position shall not be deleted. If the motion command
33174		would move the current cursor position toward the end of the command line, the
33175		character under the current cursor position shall be deleted. If the <i>count</i> is larger
33176		than the number of characters between the current cursor position and the end of
33177		the command line toward which the motion command would move the cursor,

33178		this shall not be considered an error; all of the remaining characters in the
33179		aforementioned range shall be deleted and insert mode shall be entered. If the
33180		motion command is invalid, the terminal shall be alerted, the cursor shall not be
33181		moved, and no text shall be deleted.
33182	C	Delete from the current character to the end of the line and enter insert mode at the
33183		new end-of-line.
33184	S	Clear the entire current command line and enter insert mode.
33185	[count]rc	Replace the current character with the character 'c'. With a number <i>count</i> ,
33186		replace the current and the following <i>count</i> -1 characters. After this command, the
33187		current cursor position shall be on the last character that was changed. If the <i>count</i>
33188		is larger than the number of characters after the cursor, this shall not be considered
33189		an error; all of the remaining characters shall be changed.
33190	[count]_	Append a <space> character after the current character position and then append
33191		the last bigword in the previous input line after the <space> character. Then enter
33192		insert mode after the last character just appended. With a number <i>count</i> , append
33193		the <i>count</i> th bigword in the previous line.
33194	[count]x	Delete the character at the current cursor position and place the deleted characters
33195		in the save buffer. If the cursor was positioned on the last character of the line, the
33196		character shall be deleted and the cursor position shall be moved to the previous
33197		character (the new last character). If the <i>count</i> is larger than the number of
33198		characters after the cursor, this shall not be considered an error; all the characters
33199		from the cursor to the end of the line shall be deleted.
33200	[count]X	Delete the character before the current cursor position and place the deleted
33201		characters in the save buffer. The character under the current cursor position shall
33202		not change. If the cursor was positioned on the first character of the line, the
33203		terminal shall be alerted, and the X command shall have no effect. If the line
33204		contained a single character, the X command shall have no effect. If the line
33205		contained no characters, the terminal shall be alerted and the cursor shall not be
33206		moved. If the <i>count</i> is larger than the number of characters before the cursor, this
33207		shall not be considered an error; all the characters from before the cursor to the
33208		beginning of the line shall be deleted.
33209	[count]dmotion	
33210		Delete the characters between the current cursor position and the character
33211		position that would result from the <i>motion</i> command. A number <i>count</i> repeats the
33212		<i>motion</i> command <i>count</i> times. If the <i>motion</i> command would move toward the
33213		beginning of the command line, the character under the current cursor position
33214		shall not be deleted. If the <i>motion</i> command is d , the entire current command line
33215		shall be cleared. If the <i>count</i> is larger than the number of characters between the
33216		current cursor position and the end of the command line toward which the motion
33217		command would move the cursor, this shall not be considered an error; all of the
33218		remaining characters in the aforementioned range shall be deleted. The deleted
33219		characters shall be placed in the save buffer.
33220	D	Delete all characters from the current cursor position to the end of the line. The
33221		deleted characters shall be placed in the save buffer.
33222	[count]ymotion	
33223		Yank (that is, copy) the characters from the current cursor position to the position
33224		resulting from the <i>motion</i> command into a save buffer. A number <i>count</i> shall be
33225		applied to the <i>motion</i> command. If the <i>motion</i> command would move toward the

33226		beginning of the command line, the character under the current cursor position shall not be included in the set of yanked characters. If the <i>motion</i> command is <i>y</i> , the entire current command line shall be yanked into the save buffer. The current cursor position shall be unchanged. If the <i>count</i> is larger than the number of characters between the current cursor position and the end of the command line toward which the motion command would move the cursor, this shall not be considered an error; all of the remaining characters in the aforementioned range shall be yanked.
33227		
33228		
33229		
33230		
33231		
33232		
33233		
33234	Y	Yank the characters from the current cursor position to the end of the line into the save buffer. The current character position shall be unchanged.
33235		
33236	[count]p	Put a copy of the current contents of the save buffer after the current cursor position. The current cursor position shall be advanced to the last character put from the save buffer. A <i>count</i> shall indicate how many copies of the save buffer shall be put.
33237		
33238		
33239		
33240	[count]P	Put a copy of the current contents of the save buffer before the current cursor position. The current cursor position shall be moved to the last character put from the save buffer. A <i>count</i> shall indicate how many copies of the save buffer shall be put.
33241		
33242		
33243		
33244	u	Undo the last command that modified the text of the current command line.
33245	U	Undo all changes made to the current command line since first entering command mode on the line.
33246		
33247	[count]k	
33248	[count]-	Replace the current command line with the previous command line in the shell command history. The cursor shall be positioned on the first character of the new command. A count preceding the command shall have the same effect as executing the command <i>count</i> times. If a k or - command retreats past the maximum number of commands in effect for this shell (affected by the <i>HISTSIZE</i> environment variable), the terminal shall be alerted and the command shall have no effect.
33249		
33250		
33251		
33252		
33253		
33254		
33255	[count]j	
33256	[count]+	Replace the current command line with the next command line in the shell command history. The cursor shall be positioned on the first character of the new command. The command history position shall be remembered, and any k or - command, or j or + command, shall decrement or increment that position and then shall fetch the line at the new position. If a j or + command advances past the most recent line in the history, the current command line shall be restored to the contents before the first k or - .
33257		
33258		
33259		
33260		
33261		
33262		
33263	[number]G	Replace the current command line with the contents of the oldest command line stored in the shell command history. With a number <i>number</i> , replace the current command line with the contents of command <i>number</i> in the history.
33264		
33265		
33266	/string<newline>	
33267		Move backward through the command history, searching for the specified <i>string</i> , beginning with the previous command line. If it is not found, the current command line shall be unchanged. If it is found in a previous line, this command shall behave equivalently to a set of k commands to reach that line. If <i>string</i> begins with '^', the characters after the '^' shall be matched only at the beginning of a line.
33268		
33269		
33270		
33271		
33272		

33273 *?string<newline>*
 33274 Move forward through the command history, searching for the specified string. If
 33275 it is not found, the current command line shall be unchanged. If the string is found
 33276 in the current command line, the current cursor position shall be moved to the
 33277 beginning of that string. If it is found in the history, this command shall behave
 33278 equivalently to a set of **j** commands to reach that line. If *string* begins with '^', the
 33279 characters after the '^' shall be matched only at the beginning of a line.

33280 **n** Repeat the most recent / or ? command.

33281 **N** Repeat the most recent / or ? command, reversing the direction of the search.

33282 **EXIT STATUS**
 33283 The following exit values shall be returned:

33284 0 The script to be executed consisted solely of zero or more blank lines or comments, or
 33285 both.

33286 1-125 A non-interactive shell detected a syntax, redirection or variable assignment error.
 33287 127 A specified *command_file* could not be found by a non-interactive shell.

33288 Otherwise, the shell shall return the exit status of the last command it invoked or attempted to
 33289 invoke (see also the *exit* utility in Section 2.15 (on page 2276)).

33290 **CONSEQUENCES OF ERRORS**
 33291 See Section 2.8.1 (on page 2255).

33292 **APPLICATION USAGE**
 33293 Standard input and standard error are the files that determine whether a shell is interactive
 33294 when **-i** is not specified. For example:

33295 `sh > file`

33296 and:

33297 `sh 2> file`

33298 create interactive and non-interactive shells, respectively. Although both accept terminal input,
 33299 the results of error conditions are different, as described in Section 2.8.1 (on page 2255); in the
 33300 second example a redirection error encountered by a special built-in utility aborts the shell.

33301 A portable application must protect its first operand, if it starts with a plus sign, by preceding it
 33302 with the "--" argument that denotes the end of the options.

33303 Applications should note that the standard *PATH* to the shell cannot be assumed to be either
 33304 **/bin/sh** or **/usr/bin/sh**, and should be determined by interrogation of the *PATH* returned by
 33305 *getconf PATH*, ensuring that the returned path name is an absolute path name and not a shell
 33306 built in.

33307 For example, to determine the location of the standard *sh* utility:

33308 `command -v sh`

33309 On some systems this might return:

33310 `/usr/xpg4/bin/sh`

33311 Furthermore, on systems that support executable scripts (the "#!" construct), it is
 33312 recommended that applications using executable scripts install them using *getconf -v* to
 33313 determine the shell path name and update the "#!" script appropriately as it is being installed
 33314 (for example, with *sed*). For example:

```

33315      #
33316      # Installation time script to install correct POSIX shell path name
33317      #
33318      # Get list of paths to check
33319      #
33320      Sifs=$IFS
33321      IFS=:
33322      set $(getconf PATH)
33323      IFS=$Sifs
33324      #
33325      # Check each path for 'sh'
33326      #
33327      for i in $@
33328      do
33329          if [ -f ${i}/sh ];
33330          then
33331              Pshell=${i}/sh
33332          fi
33333      done
33334      #
33335      # This is the list of scripts to update. They should be of the
33336      # form '${name}.source' and will be transformed to '${name}'.
33337      # Each script should begin:
33338      #
33339      # !INSTALLSHELLPATH -p
33340      #
33341      scripts="a b c"
33342      #
33343      # Transform each script
33344      #
33345      for i in ${scripts}
33346      do
33347          sed -e "s|INSTALLSHELLPATH|${Pshell}|" < ${i}.source > ${i}
33348      done

```

33349 EXAMPLES

- 33350 1. Execute a shell command from a string:


```
33351     sh -c "cat myfile"
```
- 33352 2. Execute a shell script from a file in the current directory:


```
33353     sh my_shell_cmds
```

33354 RATIONALE

33355 The *sh* utility and the *set* special built-in utility share a common set of options.

33356 The KornShell ignores the contents of *IFS* upon entry to the script. A conforming application
 33357 cannot rely on importing *IFS*. One justification for this, beyond security considerations, is to
 33358 assist possible future shell compilers. Allowing *IFS* to be imported from the environment
 33359 prevents many optimizations that might otherwise be performed via dataflow analysis of the
 33360 script itself.

33361 The text in the STDIN section about non-blocking reads concerns an instance of *sh* that has been
 33362 invoked, probably by a C-language program, with standard input that has been opened using

33363 the O_NONBLOCK flag; see *open()* in the System Interfaces volume of IEEE Std. 1003.1-200x. If
33364 the shell did not reset this flag, it would immediately terminate because no input data would be
33365 available yet and that would be considered the same as end-of-file.

33366 The options associated with a *restricted shell* (command name *rsh* and the *-r* option) were
33367 excluded because the standard developers considered that the implied level of security could
33368 not be achieved and they did not want to raise false expectations.

33369 On systems that support set-user-ID scripts, a historical trapdoor has been to link a script to the
33370 name *-i*. When it is called by a sequence such as

33371 `sh -`

33372 or by:

33373 `#! usr/bin/sh -`

33374 the historical systems have assumed that no option letters follow. Thus, this volume of
33375 IEEE Std. 1003.1-200x allows the single hyphen to mark the end of the options, in addition to the
33376 use of the regular "—" argument, because it was considered that the older practice was so
33377 pervasive. An alternative approach is taken by the KornShell, where real and effective
33378 user/group IDs must match for an interactive shell; this behavior is specifically allowed by this
33379 volume of IEEE Std. 1003.1-200x.

33380 **Note:** There are other problems with set-user-ID scripts that the two approaches described
33381 here do not resolve.

33382 The default messages for the various *MAIL*-related messages are unspecified because they vary
33383 across implementations. Typical messages are:

33384 `"you have mail\n"`

33385 or:

33386 `"you have new mail\n"`

33387 It is important that the descriptions of command line editing refer to the same shell as that in
33388 IEEE Std. 1003.1-200x so that interactive users can also be application programmers without
33389 having to deal with programmatic differences in their two environments. It is also essential that
33390 the utility name *sh* be specified because this explicit utility name is too firmly rooted in historical
33391 practice of application programs for it to change.

33392 Consideration was given to mandating a diagnostic message when attempting to set *vi*-mode on
33393 terminals that do not support command line editing. However, it is not historical practice for the
33394 shell to be cognizant of all terminal types and thus be able to detect inappropriate terminals in
33395 all cases. Implementations are encouraged to supply diagnostics in this case whenever possible,
33396 rather than leaving the user in a state where editing commands work incorrectly.

33397 In early proposals, the KornShell-derived *emacs* mode of command line editing was included,
33398 even though the *emacs* editor itself was not. The community of *emacs* proponents was adamant
33399 that the full *emacs* editor not be included in this volume of IEEE Std. 1003.1-200x because they
33400 were concerned that an attempt to standardize this very powerful environment would
33401 encourage vendors to ship versions conforming strictly to this volume of IEEE Std. 1003.1-200x,
33402 but lacking the extensibility required by the community. The author of the original *emacs*
33403 program also expressed his desire to omit the program. Furthermore, there were a number of
33404 historical systems that did not include *emacs*, or included it without supporting it, but there were
33405 very few that did not include and support *vi*. The shell *emacs* command line editing mode was
33406 finally omitted from this volume of IEEE Std. 1003.1-200x because it became apparent that the
33407 KornShell version and the editor being distributed with the GNU system had diverged in some

33408 respects. The author of *emacs* requested that the POSIX *emacs* mode either be deleted or have a
 33409 significant number of unspecified conditions. Although the KornShell author agreed to consider
 33410 changes to bring the shell into alignment, the standard developers decided to defer specification
 33411 at this time, rather than attempting to agree on a specific subset of *emacs* late within the
 33412 development of this volume of IEEE Std. 1003.1-200x. It is assumed that the *emacs* and KornShell
 33413 developers will converge on a definition acceptable to both groups, and this may be used as a
 33414 model for a future version of this volume of IEEE Std. 1003.1-200x. In the interim,
 33415 implementations are free to offer additional command line editing modes based on the exact
 33416 models of editors their users are most comfortable with.

33417 Early proposals had the following list entry in **vi Line Editing Insert Mode** (on page 3065):

33418 \ If followed by the *erase* or *kill* character, that character shall be inserted into the input line.
 33419 Otherwise, the backslash itself shall be inserted into the input line.

33420 However, this is not actually a feature of *sh* command line editing insert mode, but one of some
 33421 historical terminal line drivers. Some conforming implementations continue to do this when the
 33422 *stty ixtextn* flag is set.

33423 FUTURE DIRECTIONS

33424 None.

33425 SEE ALSO

33426 *cd*, *echo*, *pwd*, *test*, *umask*, the System Interfaces volume of IEEE Std. 1003.1-200x, *dup()*, *exec*,
 33427 *exit()*, *fork()*, *pipe()*, *signal()*, *system()*, *ulimit()*, *umask()*, *wait()*

33428 CHANGE HISTORY

33429 First released in Issue 2.

33430 Issue 4

33431 Aligned with the ISO/IEC 9945-2: 1993 standard.

33432 Description of the shell command language and special built-ins moved to Chapter 2 (on page
 33433 2235).

33434 Issue 5

33435 FUTURE DIRECTIONS section added.

33436 Text is added to the DESCRIPTION for the Large File Summit proposal.

33437 Issue 6

33438 The Open Group corrigenda item U029/2 has been applied, correcting the second SYNOPSIS.

33439 The Open Group corrigenda item U027/3 has been applied, correcting a typographical error.

33440 The following new requirements on POSIX implementations derive from alignment with the
 33441 Single UNIX Specification:

33442 • The option letters derived from the set special built-in are also accepted with a leading plus
 33443 sign (' + ').

33444 • Large file extensions are added:

33445 — Path name expansion does not fail due to the size of a file.

33446 — Shell input and output redirections have an implementation-defined offset maximum
 33447 that is established in the open file description.

33448 In the ENVIRONMENT VARIABLES section, the text “user’s home directory” is updated to
 33449 “directory referred to by the *HOME* environment variable”.

33450 Descriptions for the *ENV* and *PWD* environment variables are included to align with the
33451 IEEE P1003.2b draft standard.
33452 The normative text is reworded to avoid use of the term “must” for application requirements.

33453 **NAME**

33454 sleep — suspend execution for an interval

33455 **SYNOPSIS**33456 sleep *time*33457 **DESCRIPTION**33458 The *sleep* utility shall suspend execution for at least the integral number of seconds specified by
33459 the *time* operand.33460 **OPTIONS**

33461 None.

33462 **OPERANDS**

33463 The following operand shall be supported:

33464 *time* A non-negative decimal integer specifying the number of seconds for which to
33465 suspend execution.33466 **STDIN**

33467 Not used.

33468 **INPUT FILES**

33469 None.

33470 **ENVIRONMENT VARIABLES**33471 The following environment variables shall affect the execution of *sleep*:33472 *LANG* Provide a default value for the internationalization variables that are unset or null.
33473 If *LANG* is unset or null, the corresponding value from the implementation-
33474 defined default locale shall be used. If any of the internationalization variables
33475 contains an invalid setting, the utility shall behave as if none of the variables had
33476 been defined.33477 *LC_ALL* If set to a non-empty string value, override the values of all the other
33478 internationalization variables.33479 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
33480 characters (for example, single-byte as opposed to multi-byte characters in
33481 arguments).33482 *LC_MESSAGES*33483 Determine the locale that should be used to affect the format and contents of
33484 diagnostic messages written to standard error.33485 *XSI* *NLS_PATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.33486 **ASYNCHRONOUS EVENTS**33487 If the *sleep* utility receives a SIGALRM signal, one of the following actions shall be taken:

- 33488 1. Terminate normally with a zero exit status.
-
- 33489 2. Effectively ignore the signal.
-
- 33490 3. Provide the default behavior for signals described in the ASYNCHRONOUS EVENTS
-
- 33491 section of Section 1.11 (on page 2224). This could include terminating with a non-zero exit
-
- 33492 status.

33493 The *sleep* utility shall take the standard action for all other signals.

33494 **STDOUT**

33495 Not used.

33496 **STDERR**

33497 Used only for diagnostic messages.

33498 **OUTPUT FILES**

33499 None.

33500 **EXTENDED DESCRIPTION**

33501 None.

33502 **EXIT STATUS**

33503 The following exit values shall be returned:

33504 0 The execution was successfully suspended for at least *time* seconds, or a SIGALRM signal
33505 was received. See the ASYNCHRONOUS EVENTS section.

33506 >0 An error occurred.

33507 **CONSEQUENCES OF ERRORS**

33508 Default.

33509 **APPLICATION USAGE**

33510 None.

33511 **EXAMPLES**33512 The *sleep* utility can be used to execute a command after a certain amount of time, as in:33513 `(sleep 105; command) &`

33514 or to execute a command every so often, as in:

```
33515 while true
33516 do
33517     command
33518     sleep 37
33519 done
```

33520 **RATIONALE**

33521 The exit status is allowed to be zero when *sleep* is interrupted by the SIGALRM signal because
33522 most implementations of this utility rely on the arrival of that signal to notify them that the
33523 requested finishing time has been successfully attained. Such implementations thus do not
33524 distinguish this situation from the successful completion case. Other implementations are
33525 allowed to catch the signal and go back to sleep until the requested time expires or to provide
33526 the normal signal termination procedures.

33527 As with all other utilities that take integral operands and do not specify subranges of allowed
33528 values, *sleep* is required by this volume of IEEE Std. 1003.1-200x to deal with *time* requests of up
33529 to 2 147 483 647 seconds. This may mean that some implementations have to make multiple calls
33530 to the delay mechanism of the underlying operating system if its argument range is less than
33531 this.

33532 **FUTURE DIRECTIONS**

33533 None.

33534 **SEE ALSO**33535 *wait*, the System Interfaces volume of IEEE Std. 1003.1-200x, *alarm()*, *sleep()*

33536 **CHANGE HISTORY**

33537 First released in Issue 2.

33538 **Issue 4**

33539 Aligned with the ISO/IEC 9945-2:1993 standard.

33540 NAME

33541 sort — sort, merge, or sequence check text files

33542 SYNOPSIS

33543 sort [-m][-o *output*][-bdfinru][-t *char*][-k *keydef*]... [*file*...]

33544 sort -c [-bdfinru][-t *char*][-k *keydef*]... *file*

33545 DESCRIPTION

33546 The *sort* utility shall perform one of the following functions:

- 33547 1. Sort lines of all the named files together and write the result to the specified output.
- 33548 2. Merge lines of all the named (presorted) files together and write the result to the specified
33549 output.
- 33550 3. Check that a single input file is correctly presorted.

33551 Comparisons shall be based on one or more sort keys extracted from each line of input (or, if no
33552 sort keys are specified, the entire line up to, but not including, the terminating <newline>
33553 character), and shall be performed using the collating sequence of the current locale.

33554 OPTIONS

33555 The *sort* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
33556 12.2, Utility Syntax Guidelines, and the **-k** *keydef* option should follow the **-b**, **-d**, **-f**, **-i**, **-n**, and
33557 **-r** options.

33558 The following options shall be supported:

- 33559 **-c** Check that the single input file is ordered as specified by the arguments and the
33560 collating sequence of the current locale. No output shall be produced; only the exit
33561 code shall be affected.
- 33562 **-m** Merge only; the input file shall be assumed to be already sorted.
- 33563 **-o** *output* Specify the name of an output file to be used instead of the standard output. This
33564 file can be the same as one of the input *files*.
- 33565 **-u** Unique: suppress all but one in each set of lines having equal keys. If used with
33566 the **-c** option, check that there are no lines with duplicate keys, in addition to
33567 checking that the input file is sorted.

33568 The following options shall override the default ordering rules. When ordering options appear
33569 independent of any key field specifications, the requested field ordering rules shall be applied
33570 globally to all sort keys. When attached to a specific key (see **-k**), the specified ordering options
33571 shall override all global ordering options for that key.

- 33572 **-d** Specify that only <blank> characters and alphanumeric characters, according to
33573 the current setting of *LC_CTYPE*, shall be significant in comparisons. The behavior
33574 is undefined for a sort key to which **-i** or **-n** also applies.
- 33575 **-f** Consider all lowercase characters that have uppercase equivalents, according to
33576 the current setting of *LC_CTYPE*, to be the uppercase equivalent for the purposes
33577 of comparison.
- 33578 **-i** Ignore all characters that are non-printable, according to the current setting of
33579 *LC_CTYPE*.
- 33580 **-n** Restrict the sort key to an initial numeric string, consisting of optional <blank>
33581 characters, optional minus sign, and zero or more digits with an optional radix
33582 character and thousands separators (as defined in the current locale), which shall

- 33583 be sorted by arithmetic value. An empty digit string shall be treated as zero.
 33584 Leading zeros and signs on zeros shall not affect ordering.
- 33585 **-r** Reverse the sense of comparisons.
- 33586 The treatment of field separators can be altered using the options:
- 33587 **-b** Ignore leading <blank> characters when determining the starting and ending
 33588 positions of a restricted sort key. If the **-b** option is specified before the first **-k**
 33589 option, it shall be applied to all **-k** options. Otherwise, the **-b** option can be
 33590 attached independently to each **-k** *field_start* or *field_end* option-argument (see
 33591 below).
- 33592 **-t char** Use *char* as the field separator character; *char* shall not be considered to be part of a
 33593 field (although it can be included in a sort key). Each occurrence of *char* shall be
 33594 significant (for example, <*char*><*char*> delimits an empty field). If **-t** is not
 33595 specified, <blank> characters shall be used as default field separators; each
 33596 maximal non-empty sequence of <blank> characters that follows a non-<blank>
 33597 character shall be a field separator.
- 33598 Sort keys can be specified using the options:
- 33599 **-k keydef** The *keydef* argument is a restricted sort key field definition. The format of this
 33600 definition is:
- 33601 *field_start*[*type*][, *field_end*[*type*]]
- 33602 where *field_start* and *field_end* define a key field restricted to a portion of the line
 33603 (see the EXTENDED DESCRIPTION section), and *type* is a modifier from the list of
 33604 characters 'b', 'd', 'f', 'i', 'n', 'r'. The 'b' modifier shall behave like the
 33605 **-b** option, but applies only to the *field_start* or *field_end* to which it is attached. The
 33606 other modifiers shall behave like the corresponding options, but shall apply only
 33607 to the key field to which they are attached; they shall have this effect if specified
 33608 with *field_start*, *field_end*, or both. If any modifier is attached to a *field_start* or to a
 33609 *field_end*, no option shall apply to either. Implementations shall support at least
 33610 nine occurrences of the **-k** option, which shall be significant in command line
 33611 order. If no **-k** option is specified, a default sort key of the entire line shall be used.
- 33612 When there are multiple key fields, later keys shall be compared only after all
 33613 earlier keys compare equal. Except when the **-u** option is specified, lines that
 33614 otherwise compare equal shall be ordered as if none of the options **-d**, **-f**, **-i**, **-n**, or
 33615 **-k** were present (but with **-r** still in effect, if it was specified) and with all bytes in
 33616 the lines significant to the comparison. The order in which lines that still compare
 33617 equal are written is unspecified.
- 33618 **OPERANDS**
- 33619 The following operand shall be supported:
- 33620 *file* A path name of a file to be sorted, merged, or checked. If no *file* operands are
 33621 specified, or if a *file* operand is '-', the standard input shall be used.
- 33622 **STDIN**
- 33623 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
 33624 See the INPUT FILES section.

33625 **INPUT FILES**

33626 The input files shall be text files, except that the *sort* utility shall add a <newline> character to
 33627 the end of a file ending with an incomplete last line.

33628 **ENVIRONMENT VARIABLES**

33629 The following environment variables shall affect the execution of *sort*:

33630 *LANG* Provide a default value for the internationalization variables that are unset or null.
 33631 If *LANG* is unset or null, the corresponding value from the implementation-
 33632 defined default locale shall be used. If any of the internationalization variables
 33633 contains an invalid setting, the utility shall behave as if none of the variables had
 33634 been defined.

33635 *LC_ALL* If set to a non-empty string value, override the values of all the other
 33636 internationalization variables.

33637 *LC_COLLATE*
 33638 Determine the locale for ordering rules.

33639 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 33640 characters (for example, single-byte as opposed to multi-byte characters in
 33641 arguments and input files) and the behavior of character classification for the *-b*,
 33642 *-d*, *-f*, *-i*, and *-n* options.

33643 *LC_MESSAGES*
 33644 Determine the locale that should be used to affect the format and contents of
 33645 diagnostic messages written to standard error.

33646 *LC_NUMERIC*
 33647 Determine the locale for the definition of the radix character and thousands
 33648 separator for the *-n* option.

33649 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

33650 **ASYNCHRONOUS EVENTS**

33651 Default.

33652 **STDOUT**

33653 Unless the *-o* or *-c* options are in effect, the standard output shall contain the sorted input.

33654 **STDERR**

33655 Used for diagnostic messages. A warning message about correcting an incomplete last line of an
 33656 input file may be generated, but need not affect the final exit status.

33657 **OUTPUT FILES**

33658 If the *-o* option is in effect, the sorted input shall be written to the file *output*.

33659 **EXTENDED DESCRIPTION**

33660 The notation:

33661 *-k field_start[type][,field_end[type]]*

33662 shall define a key field that begins at *field_start* and ends at *field_end* inclusive, unless *field_start*
 33663 falls beyond the end of the line or after *field_end*, in which case the key field is empty. A missing
 33664 *field_end* shall mean the last character of the line.

33665 A field comprises a maximal sequence of non-separating characters and, in the absence of option
 33666 *-t*, any preceding field separator.

33667 The *field_start* portion of the *keydef* option-argument shall have the form:

33668 *field_number*[*.first_character*]

33669 Fields and characters within fields shall be numbered starting with 1. The *field_number* and
33670 *first_character* pieces, interpreted as positive decimal integers, shall specify the first character to
33671 be used as part of a sort key. If *.first_character* is omitted, it shall refer to the first character of the
33672 field.

33673 The *field_end* portion of the *keydef* option-argument shall have the form:

33674 *field_number*[*.last_character*]

33675 The *field_number* shall be as described above for *field_start*. The *last_character* piece, interpreted
33676 as a non-negative decimal integer, shall specify the last character to be used as part of the sort
33677 key. If *last_character* evaluates to zero or *.last_character* is omitted, it shall refer to the last
33678 character of the field specified by *field_number*.

33679 If the **-b** option or **b** type modifier is in effect, characters within a field shall be counted from the
33680 first non-<blank> character in the field. (This shall apply separately to *first_character* and
33681 *last_character*.)

33682 EXIT STATUS

33683 The following exit values shall be returned:

33684 0 All input files were output successfully, or **-c** was specified and the input file was correctly
33685 sorted.

33686 1 Under the **-c** option, the file was not ordered as specified, or if the **-c** and **-u** options were
33687 both specified, two input lines were found with equal keys.

33688 >1 An error occurred.

33689 CONSEQUENCES OF ERRORS

33690 Default.

33691 APPLICATION USAGE

33692 The default value for **-t**, <blank> character, has different properties from, for example,
33693 **-t"<space>"**. If a line contains:

33694 <space><space>foo

33695 the following treatment would occur with default separation as opposed to specifically selecting
33696 a <space> character:

33697

Field	Default	-t "<space>"
1	<space><space>foo	<i>empty</i>
2	<i>empty</i>	<i>empty</i>
3	<i>empty</i>	foo

33698

33699

33700

33701 The leading field separator itself is included in a field when **-t** is not used. For example, this
33702 command returns an exit status of zero, meaning the input was already sorted:

33703 `sort -c -k 2 <<eof`

33704 `y<tab>b`

33705 `x<space>a`

33706 `eof`

33707 (assuming that a <tab> character precedes the <space> character in the current collating
33708 sequence). The field separator is not included in a field when it is explicitly set via **-t**. This is
33709 historical practice and allows usage such as:

```

33710      sort -t "|" -k 2n <<eof
33711      Atlanta|425022|Georgia
33712      Birmingham|284413|Alabama
33713      Columbia|100385|South Carolina
33714      eof

```

33715 where the second field can be correctly sorted numerically without regard to the non-numeric
33716 field separator.

33717 The wording in the OPTIONS section clarifies that the **-b**, **-d**, **-f**, **-i**, **-n**, and **-r** options have to
33718 come before the first sort key specified if they are intended to apply to all specified keys. The
33719 way it is described in this volume of IEEE Std. 1003.1-200x matches historical practice, not
33720 historical documentation. The results are unspecified if these options are specified after a **-k**
33721 option.

33722 The **-f** option might not work as expected in locales where there is not a one-to-one mapping
33723 between an uppercase and a lowercase letter.

33724 EXAMPLES

33725 1. The following command sorts the contents of **infile** with the second field as the sort key:

```
33726      sort -k 2,2 infile
```

33727 2. The following command sorts, in reverse order, the contents of **infile1** and **infile2**, placing
33728 the output in **outfile** and using the second character of the second field as the sort key
33729 (assuming that the first character of the second field is the field separator):

```
33730      sort -r -o outfile -k 2.2,2.2 infile1 infile2
```

33731 3. The following command sorts the contents of **infile1** and **infile2** using the second non-
33732 <blank> character of the second field as the sort key:

```
33733      sort -k 2.2b,2.2b infile1 infile2
```

33734 4. The following command prints the System V password file (user database) sorted by the
33735 numeric user ID (the third colon-separated field):

```
33736      sort -t : -k 3,3n /etc/passwd
```

33737 5. The following command prints the lines of the already sorted file **infile**, suppressing all
33738 but one occurrence of lines having the same third field:

```
33739      sort -um -k 3.1,3.0 infile
```

33740 RATIONALE

33741 Examples in some historical documentation state that options **-um** with one input file keep the
33742 first in each set of lines with equal keys. This behavior was deemed to be an implementation
33743 artifact and was not standardized.

33744 The **-z** option was omitted; it is not standard practice on most systems and is inconsistent with
33745 using *sort* to sort several files individually and then merge them together. The text concerning **-z**
33746 in historical documentation appeared to require implementations to determine the proper buffer
33747 length during the sort phase of operation, but not during the merge.

33748 The **-y** option was omitted because of non-portability. The **-M** option, present in System V, was
33749 omitted because of non-portability in international usage.

33750 An undocumented **-T** option exists in some implementations. It is used to specify a directory for
33751 intermediate files. Implementations are encouraged to support the use of the *TMPDIR*
33752 environment variable instead of adding an option to support this functionality.

- 33753 The **-k** option was added to satisfy two objections. First, the zero-based counting used by *sort* is
33754 not consistent with other utility conventions. Second, it did not meet syntax guideline
33755 requirements.
- 33756 Historical documentation indicates that “setting **-n** implies **-b**”. The description of **-n** already
33757 states that optional leading <blank>s are tolerated in doing the comparison. If **-b** is enabled,
33758 rather than implied, by **-n**, this has unusual side effects. When a character offset is used in a
33759 column of numbers (for example, to sort modulo 100), that offset is measured relative to the
33760 most significant digit, not to the column. Based upon a recommendation from the author of the
33761 original *sort* utility, the **-b** implication has been omitted from this volume of
33762 IEEE Std. 1003.1-200x, and an application wishing to achieve the previously mentioned side
33763 effects has to code the **-b** flag explicitly.
- 33764 **FUTURE DIRECTIONS**
- 33765 None.
- 33766 **SEE ALSO**
- 33767 *comm, join, uniq*, the System Interfaces volume of IEEE Std. 1003.1-200x, *toupper()*
- 33768 **CHANGE HISTORY**
- 33769 First released in Issue 2.
- 33770 **Issue 4**
- 33771 Aligned with the ISO/IEC 9945-2:1993 standard.
- 33772 **Issue 6**
- 33773 IEEE PASC Interpretation 1003.2 #174 is applied, updating the DESCRIPTION of comparisons.
- 33774 IEEE PASC Interpretation 1003.2 #168 is applied.

33775 NAME

33776 split — split files into pieces

33777 SYNOPSIS

33778 UP `split [-l line_count][-a suffix_length][file[name]]`33779 `split -b n[k|m][-a suffix_length][file[name]]`

33780

33781 DESCRIPTION

33782 The *split* utility shall read an input file and write one or more output files. The default size of
 33783 each output file shall be 1 000 lines. The size of the output files can be modified by specification
 33784 of the `-b` or `-l` options. Each output file shall be created with a unique suffix. The suffix shall
 33785 consist of exactly *suffix_length* lowercase letters from the POSIX locale. The letters of the suffix
 33786 shall be used as if they were a base-26 digit system, with the first suffix to be created consisting
 33787 of all 'a' characters, the second with a 'b' replacing the last 'a', and so on, until a name of all
 33788 'z' characters is created. By default, the names of the output files shall be 'x', followed by a
 33789 two-character suffix from the character set as described above, starting with "aa", "ab", "ac",
 33790 and so on, and continuing until the suffix "zz", for a maximum of 676 files.

33791 If the number of files required exceeds the maximum allowed by the suffix length provided,
 33792 such that the last allowable file would be larger than the requested size, the *split* utility shall fail
 33793 after creating the last file with a valid suffix; *split* shall not delete the files it created with valid
 33794 suffixes. If the file limit is not exceeded, the last file created shall contain the remainder of the
 33795 input file, and may be smaller than the requested size.

33796 OPTIONS

33797 The *split* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 33798 12.2, Utility Syntax Guidelines.

33799 The following options shall be supported:

33800 `-a suffix_length`

33801 Use *suffix_length* letters to form the suffix portion of the file names of the split file.
 33802 If `-a` is not specified, the default suffix length shall be two. If the sum of the *name*
 33803 operand and the *suffix_length* option-argument would create a file name exceeding
 33804 {NAME_MAX} bytes, an error shall result; *split* shall exit with a diagnostic
 33805 message and no files shall be created.

33806 `-b n` Split a file into pieces *n* bytes in size.

33807 `-b nk` Split a file into pieces *n**1024 bytes in size.

33808 `-b nm` Split a file into pieces *n**1 048 576 bytes in size.

33809 `-l line_count` Specify the number of lines in each resulting file piece. The *line_count* argument is
 33810 an unsigned decimal integer. The default is 1 000. If the input does not end with a
 33811 <newline> character, the partial line shall be included in the last output file.

33812 OPERANDS

33813 The following operands shall be supported:

33814 *file* The path name of the ordinary file to be split. If no input file is given or *file* is '-',
 33815 the standard input shall be used.

33816 *name* The prefix to be used for each of the files resulting from the split operation. If no
 33817 *name* argument is given, 'x' shall be used as the prefix of the output files. The
 33818 combined length of the basename of *prefix* and *suffix_length* cannot exceed
 33819 {NAME_MAX} bytes. See the OPTIONS section.

33820 **STDIN**

33821 See the INPUT FILES section.

33822 **INPUT FILES**

33823 Any file can be used as input.

33824 **ENVIRONMENT VARIABLES**33825 The following environment variables shall affect the execution of *split*:

33826 *LANG* Provide a default value for the internationalization variables that are unset or null.
33827 If *LANG* is unset or null, the corresponding value from the implementation-
33828 defined default locale shall be used. If any of the internationalization variables
33829 contains an invalid setting, the utility shall behave as if none of the variables had
33830 been defined.

33831 *LC_ALL* If set to a non-empty string value, override the values of all the other
33832 internationalization variables.

33833 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
33834 characters (for example, single-byte as opposed to multi-byte characters in
33835 arguments and input files).

33836 *LC_MESSAGES*

33837 Determine the locale that should be used to affect the format and contents of
33838 diagnostic messages written to standard error.

33839 *NSLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

33840 **ASYNCHRONOUS EVENTS**

33841 Default.

33842 **STDOUT**

33843 Not used.

33844 **STDERR**

33845 Used only for diagnostic messages.

33846 **OUTPUT FILES**

33847 The output files contain portions of the original input file; otherwise, unchanged.

33848 **EXTENDED DESCRIPTION**

33849 None.

33850 **EXIT STATUS**

33851 The following exit values shall be returned:

33852 0 Successful completion.

33853 >0 An error occurred.

33854 **CONSEQUENCES OF ERRORS**

33855 Default.

33856 **APPLICATION USAGE**

33857 None.

33858 **EXAMPLES**33859 In the following examples **foo** is a text file that contains 5 000 lines.33860 1. Create five files, **xaa**, **xab**, **xac**, **xad**, and **xae**:33861 `split foo`33862 2. Create five files, but the suffixed portion of the created files consists of three letters, **xaaa**,
33863 **xaab**, **xaac**, **xaad**, and **xaae**:33864 `split -a 3 foo`33865 3. Create three files with four-letter suffixes and a supplied prefix, **bar_aaaa**, **bar_aaab**, and
33866 **bar_aaac**:33867 `split -a 4 -l 2000 foo bar_`33868 4. Create as many files as are necessary to contain at most 20*1 024 bytes, each with the
33869 default prefix of **x** and a five-letter suffix:33870 `split -a 5 -b 20k foo`33871 **RATIONALE**33872 The **-b** option was added to provide a mechanism for splitting files other than by lines. While
33873 most uses of the **-b** option are for transmitting files over networks, some believed it would have
33874 additional uses.33875 The **-a** option was added to overcome the limitation of being able to create only 676 files.33876 Consideration was given to deleting this utility, using the rationale that the function provided
33877 by this utility is available via the *csplit* utility (see *csplit* (on page 2492)). Upon reconsideration of
33878 the purpose of the User Portability Extension, it was decided to retain both this utility and the
33879 *csplit* utility because users use both utilities and have historical expectations of their behavior.
33880 Furthermore, the splitting on byte boundaries in *split* cannot be duplicated with the historical
33881 *csplit*.33882 The text “*split* shall not delete the files it created with valid suffixes” would normally be
33883 assumed, but since the related utility, *csplit*, does delete files under some circumstances, the
33884 historical behavior of *split* is made explicit to avoid misinterpretation.33885 **FUTURE DIRECTIONS**

33886 None.

33887 **SEE ALSO**33888 *csplit*33889 **CHANGE HISTORY**

33890 First released in Issue 2.

33891 **Issue 4**

33892 Aligned with the ISO/IEC 9945-2: 1993 standard.

33893 **Issue 6**

33894 This utility is now marked as part of the User Portability Utilities option.

33895 The APPLICATION USAGE section is added.

33896 The obsolescent SYNOPSIS is removed.

33897 **NAME**

33898 strings — find printable strings in files

33899 **SYNOPSIS**33900 UP strings [-a][-t *format*][-n *number*][*file...*]

33901

33902 **DESCRIPTION**

33903 The *strings* utility shall look for printable strings in regular files and shall write those strings to
 33904 standard output. A printable string is any sequence of four (by default) or more printable
 33905 characters terminated by a <newline> or NUL character. Additional implementation-defined
 33906 strings may be written; see *localedef*.

33907 **OPTIONS**

33908 The *strings* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 33909 12.2, Utility Syntax Guidelines.

33910 The following options shall be supported:

33911 **-a** Scan files in their entirety. If **-a** is not specified, it is implementation-defined what
 33912 portion of each file is scanned for strings.

33913 **-n *number*** Specify the minimum string length, where the *number* argument is a positive
 33914 decimal integer. The default shall be 4.

33915 **-t *format*** Write each string preceded by its byte offset from the start of the file. The format
 33916 shall be dependent on the single character used as the *format* option-argument:

33917 d The offset shall be written in decimal.

33918 o The offset shall be written in octal.

33919 x The offset shall be written in hexadecimal.

33920 **OPERANDS**

33921 The following operand shall be supported:

33922 ***file*** A path name of a regular file to be used as input. If no *file* operand is specified, the
 33923 *strings* utility shall read from the standard input.

33924 **STDIN**

33925 See the INPUT FILES section.

33926 **INPUT FILES**

33927 The input files named by the utility arguments or the standard input shall be regular files of any
 33928 format.

33929 **ENVIRONMENT VARIABLES**

33930 The following environment variables shall affect the execution of *strings*:

33931 ***LANG*** Provide a default value for the internationalization variables that are unset or null.
 33932 If *LANG* is unset or null, the corresponding value from the implementation-
 33933 defined default locale shall be used. If any of the internationalization variables
 33934 contains an invalid setting, the utility shall behave as if none of the variables had
 33935 been defined.

33936 ***LC_ALL*** If set to a non-empty string value, override the values of all the other
 33937 internationalization variables.

33938 ***LC_CTYPE*** Determine the locale for the interpretation of sequences of bytes of text data as
 33939 characters (for example, single-byte as opposed to multi-byte characters in

- 33940 arguments and input files) and to identify printable strings.
- 33941 **LC_MESSAGES**
- 33942 Determine the locale that should be used to affect the format and contents of
- 33943 diagnostic messages written to standard error.
- 33944 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 33945 **ASYNCHRONOUS EVENTS**
- 33946 Default.
- 33947 **STDOUT**
- 33948 Strings found shall be written to the standard output, one per line.
- 33949 When the **-t** option is not specified, the format of the output shall be:
- 33950 "%s", <string>
- 33951 With the **-t o** option, the format of the output shall be:
- 33952 "%o %s", <byte offset>, <string>
- 33953 With the **-t x** option, the format of the output shall be:
- 33954 "%x %s", <byte offset>, <string>
- 33955 With the **-t d** option, the format of the output shall be:
- 33956 "%d %s", <byte offset>, <string>
- 33957 **STDERR**
- 33958 Used only for diagnostic messages.
- 33959 **OUTPUT FILES**
- 33960 None.
- 33961 **EXTENDED DESCRIPTION**
- 33962 None.
- 33963 **EXIT STATUS**
- 33964 The following exit values shall be returned:
- 33965 0 Successful completion.
- 33966 >0 An error occurred.
- 33967 **CONSEQUENCES OF ERRORS**
- 33968 Default.
- 33969 **APPLICATION USAGE**
- 33970 By default the data area (as opposed to the text, "bss" or header areas) of a binary executable file
- 33971 is scanned. Implementations document which areas are scanned.
- 33972 Some historical implementations do not require NUL or <newline> character terminators for
- 33973 strings to permit those languages that do not use NUL as a string terminator to have their strings
- 33974 written.
- 33975 **EXAMPLES**
- 33976 None.

33977 **RATIONALE**

33978 Apart from rationalizing the option syntax and slight difficulties with object and executable
33979 binary files, *strings* is specified to match historical practice closely. The **-a** and **-n** options were
33980 introduced to replace the non-conforming **-** and *-number* options.

33981 The **-o** option historically means different things on different implementations. Some use it to
33982 mean “*offset in decimal*”, while others use it as “*offset in octal*”. Instead of trying to decide which
33983 way would be least objectionable, the **-t** option was added. It was originally named **-O** to mean
33984 “*offset*”, but was changed to **-t** to be consistent with *od*.

33985 The ISO C standard function *isprint()* is restricted to a domain of **unsigned char**. This volume of
33986 IEEE Std. 1003.1-200x requires implementations to write strings as defined by the current locale.

33987 **FUTURE DIRECTIONS**

33988 None.

33989 **SEE ALSO**

33990 *nm*

33991 **CHANGE HISTORY**

33992 First released in Issue 4.

33993 **Issue 6**

33994 This utility is now marked as part of the User Portability Utilities option.

33995 The obsolescent SYNOPSIS is removed.

33996 The normative text is reworded to avoid use of the term “*must*” for application requirements.

33997 NAME

33998 strip — remove unnecessary information from executable files (DEVELOPMENT)

33999 SYNOPSIS

34000 SD strip file...

34001

34002 DESCRIPTION

34003 The *strip* utility shall remove from executable files named by the *file* operands any information
34004 the implementor deems unnecessary for execution of those files. The nature of that information
34005 is unspecified. The effect of *strip* shall be similar to the use of the *-s* option to *cc*, *c99*, or *fort77*.

34006 OPTIONS

34007 None.

34008 OPERANDS

34009 The following operand shall be supported:

34010 *file* A path name referring to an executable file.

34011 STDIN

34012 Not used.

34013 INPUT FILES

34014 The input files shall be in the form of executable files successfully produced by any compiler
34015 defined by this volume of IEEE Std. 1003.1-200x.

34016 ENVIRONMENT VARIABLES

34017 The following environment variables shall affect the execution of *strip*:

34018 *LANG* Provide a default value for the internationalization variables that are unset or null.
34019 If *LANG* is unset or null, the corresponding value from the implementation-
34020 defined default locale shall be used. If any of the internationalization variables
34021 contains an invalid setting, the utility shall behave as if none of the variables had
34022 been defined.

34023 *LC_ALL* If set to a non-empty string value, override the values of all the other
34024 internationalization variables.

34025 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
34026 characters (for example, single-byte as opposed to multi-byte characters in
34027 arguments).

34028 *LC_MESSAGES*

34029 Determine the locale that should be used to affect the format and contents of
34030 diagnostic messages written to standard error.

34031 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

34032 ASYNCHRONOUS EVENTS

34033 Default.

34034 STDOUT

34035 Not used.

34036 STDERR

34037 Used only for diagnostic messages.

34038 OUTPUT FILES

34039 The *strip* utility shall produce executable files of unspecified format.

34040 EXTENDED DESCRIPTION

34041 None.

34042 EXIT STATUS

34043 The following exit values shall be returned:

34044 0 Successful completion.

34045 >0 An error occurred.

34046 CONSEQUENCES OF ERRORS

34047 Default.

34048 APPLICATION USAGE

34049 None.

34050 EXAMPLES

34051 None.

34052 RATIONALE

34053 Historically, this utility has been used to remove the symbol table from an executable file. It was
34054 included since it is known that the amount of symbolic information can amount to several
34055 megabytes; the ability to remove it in a portable manner was deemed important, especially for
34056 smaller systems.

34057 The behavior of *strip* is said to be the same as the `-s` option to a compiler. While the end result is
34058 essentially the same, it is not required to be identical. The same effect can be achieved with
34059 either `-s` during a compile or a *strip* on the final object file.

34060 FUTURE DIRECTIONS

34061 None.

34062 SEE ALSO

34063 *ar*, *c99*, *fort77*

34064 CHANGE HISTORY

34065 First released in Issue 2.

34066 Issue 4

34067 Aligned with the ISO/IEC 9945-2:1993 standard.

34068 Issue 6

34069 This utility is now marked as part of the Software Development Utilities option.

34070 NAME

34071 stty — set the options for a terminal

34072 SYNOPSIS

34073 stty [-a | -g]

34074 stty *operands*

34075 DESCRIPTION

34076 The *stty* utility shall set or report on terminal I/O characteristics for the device that is its
 34077 standard input. Without options or operands specified, it shall report the settings of certain
 34078 characteristics, usually those that differ from implementation-defined defaults. Otherwise, it
 34079 shall modify the terminal state according to the specified operands. Detailed information about
 34080 the modes listed in the first five groups below are described in the Base Definitions volume of
 34081 IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface. Operands in the Combination
 34082 Modes group (see **Combination Modes** (on page 3099)) are implemented using operands in the
 34083 previous groups. Some combinations of operands are mutually-exclusive on some terminal
 34084 types; the results of using such combinations are unspecified.

34085 Typical implementations of this utility require a communications line configured to use the
 34086 **termios** interface defined in the System Interfaces volume of IEEE Std. 1003.1-200x. On systems
 34087 where none of these lines are available, and on lines not currently configured to support the
 34088 **termios** interface, some of the operands need not affect terminal characteristics.

34089 OPTIONS

34090 The *stty* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section
 34091 12.2, Utility Syntax Guidelines.

34092 The following options shall be supported:

34093 **-a** Write to standard output all the current settings for the terminal.

34094 **-g** Write to standard output all the current settings in an unspecified form that can be
 34095 used as arguments to another invocation of the *stty* utility on the same system. The
 34096 form used shall not contain any characters that would require quoting to avoid
 34097 word expansion by the shell; see Section 2.6 (on page 2244).

34098 OPERANDS

34099 The following operands shall be supported to set the terminal characteristics.

34100 Control Modes

34101 **parenb** (**-parenb**) Enable (disable) parity generation and detection. This has the effect of setting
 34102 (not setting) PARENB in the **termios** *c_flag* field, as defined in the Base
 34103 Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal
 34104 Interface.

34105 **parodd** (**-parodd**) Select odd (even) parity. This shall have the effect of setting (not setting)
 34106 PARODD in the **termios** *c_flag* field, as defined in the Base Definitions
 34107 volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.

34108 **cs5 cs6 cs7 cs8** Select character size, if possible. This shall have the effect of setting CS5, CS6,
 34109 CS7, and CS8, respectively, in the **termios** *c_flag* field, as defined in the Base
 34110 Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal
 34111 Interface.

34112 *number* Set terminal baud rate to the number given, if possible. If the baud rate is set
 34113 to zero, the modem control lines shall not be longer asserted. This shall have
 34114 the effect of setting the input and output **termios** baud rate values as defined

34115		in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General
34116		Terminal Interface.
34117	ispeed <i>number</i>	Set terminal input baud rate to the number given, if possible. If the input baud
34118		rate is set to zero, the input baud rate shall be specified by the value of the
34119		output baud rate. This shall have the effect of setting the input termios baud
34120		rate values as defined in the Base Definitions volume of IEEE Std. 1003.1-200x,
34121		Chapter 11, General Terminal Interface.
34122	ospeed <i>number</i>	Set terminal output baud rate to the number given, if possible. If the output
34123		baud rate is set to zero, the modem control lines shall no longer be asserted.
34124		This shall have the effect of setting the output termios baud rate values as
34125		defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11,
34126		General Terminal Interface.
34127	hupcl (-hupcl)	Stop asserting modem control lines (do not stop asserting modem control
34128		lines) on last close. This shall have the effect of setting (not setting) HUPCL in
34129		the termios <i>c_cflag</i> field, as defined in the Base Definitions volume of
34130		IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34131	hup (-hup)	Same as hupcl(-hupcl) .
34132	cstopb (-cstopb)	Use two (one) stop bits per character. This shall have the effect of setting (not
34133		setting) CSTOPB in the termios <i>c_cflag</i> field, as defined in the Base Definitions
34134		volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34135	cread (-cread)	Enable (disable) the receiver. This shall have the effect of setting (not setting)
34136		CREAD in the termios <i>c_cflag</i> field, as defined in the Base Definitions volume
34137		of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34138	local (-local)	Assume a line without (with) modem control. This shall have the effect of
34139		setting (not setting) CLOCAL in the termios <i>c_cflag</i> field, as defined in the
34140		Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General
34141		Terminal Interface.
34142		It is unspecified whether <i>stty</i> shall report an error if an attempt to set a Control Mode fails.
34143	Input Modes	
34144	ignbrk (-ignbrk)	Ignore (do not ignore) break on input. This shall have the effect of setting (not
34145		setting) IGNBRK in the termios <i>c_iflag</i> field, as defined in the Base Definitions
34146		volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34147	brkint (-brkint)	Signal (do not signal) INTR on break. This shall have the effect of setting (not
34148		setting) BRKINT in the termios <i>c_iflag</i> field, as defined in the Base Definitions
34149		volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34150	ignpar (-ignpar)	Ignore (do not ignore) bytes with parity errors. This shall have the effect of
34151		setting (not setting) IGNPAR in the termios <i>c_iflag</i> field, as defined in the Base
34152		Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal
34153		Interface.
34154	parmrk (-parmrk)	
34155		Mark (do not mark) parity errors. This shall have the effect of setting (not
34156		setting) PARMRK in the termios <i>c_iflag</i> field, as defined in the Base
34157		Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal
34158		Interface.

34159	inpck (-inpck)	Enable (disable) input parity checking. This shall have the effect of setting (not setting) INPCK in the termios <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34160		
34161		
34162	istrip (-istrip)	Strip (do not strip) input characters to seven bits. This shall have the effect of setting (not setting) ISTRIP in the termios <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34163		
34164		
34165		
34166	inlcr (-inlcr)	Map (do not map) NL to CR on input. This shall have the effect of setting (not setting) INLCR in the termios <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34167		
34168		
34169	igncr (-igncr)	Ignore (do not ignore) CR on input. This shall have the effect of setting (not setting) IGNCR in the termios <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34170		
34171		
34172	icrnl (-icrnl)	Map (do not map) CR to NL on input. This shall have the effect of setting (not setting) ICRNL in the termios <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34173		
34174		
34175	ixon (-ixon)	Enable (disable) START/STOP output control. Output from the system is stopped when the system receives STOP and started when the system receives START. This shall have the effect of setting (not setting) IXON in the termios <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34176		
34177		
34178		
34179		
34180 XSI	ixany (-ixany)	Allow any character to restart output. This shall have the effect of setting (not setting) IXANY in the termios <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34181		
34182		
34183	ixoff (-ixoff)	Request that the system send (not send) STOP characters when the input queue is nearly full and START characters to resume data transmission. This shall have the effect of setting (not setting) IXOFF in the termios <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34184		
34185		
34186		
34187		
34188	Output Modes	
34189	opost (-opost)	Post-process output (do not post-process output; ignore all other output modes). This shall have the effect of setting (not setting) OPOST in the termios <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34190		
34191		
34192		
34193 XSI	ocrnl (-ocrnl)	Map (do not map) CR to NL on output. This shall have the effect of setting (not setting) OCRNL in the termios <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34194		
34195		
34196		
34197	onocr (-onocr)	Do not (do) output CR at column zero. This shall have the effect of setting (not setting) ONOCR in the termios <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34198		
34199		
34200	onlret (-onlret)	The terminal newline key performs (does not perform) the CR function. This shall have the effect of setting (not setting) ONLRET in the termios <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34201		
34202		
34203		

34204	ofill (-ofill)	Use fill characters (use timing) for delays. This shall have the effect of setting (not setting) OFILL in the termios <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34205		
34206		
34207		
34208	ofdel (-ofdel)	Fill characters are DELs (NULs). This shall have the effect of setting (not setting) OFDEL in the termios <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34209		
34210		
34211	cr0 cr1 cr2 cr3	Select the style of delay for CRs. This shall have the effect of setting (not setting) CRDLY to CR1, CR2, CR3, or CR4, respectively, in the termios <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34212		
34213		
34214		
34215	nl0 nl1	Select the style of delay for NL. This has the effect of setting (not setting) NLDLY to NL0 or NL1, respectively, in the termios <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34216		
34217		
34218		
34219	tab0 tab1 tab2 tab3	Select the style of delay for horizontal tabs. This shall have the effect of setting (not setting) TABDLY to TAB0, TAB1, TAB2, or TAB3, respectively, in the termios <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface. Note that TAB3 has the effect of expanding <tab>s to <space>s.
34220		
34221		
34222		
34223		
34224		
34225	tabs (-tabs)	
34226		Synonym for tab0 (tab3).
34227	bs0 bs1	Select the style of delay for backspaces. This shall have the effect of setting (not setting) BSDLY to BS0 or BS1, respectively, in the termios <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34228		
34229		
34230		
34231	ff0 ff1	Select the style of delay for form-feeds. This shall have the effect of setting (not setting) FFDLY to FF0 or FF1, respectively, in the termios <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34232		
34233		
34234		
34235	vt0 vt1	Select the style of delay for vertical-tabs. This shall have the effect of setting (not setting) VTDLY to VT0 or VT1, respectively, in the termios <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34236		
34237		
34238		
34239	Local Modes	
34240	isig (-isig)	Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SUSP. This shall have the effect of setting (not setting) ISIG in the termios <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34241		
34242		
34243		
34244	icanon (-icanon)	Enable (disable) canonical input (ERASE and KILL processing). This shall have the effect of setting (not setting) ICANON in the termios <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface.
34245		
34246		
34247		
34248	ixten (-ixten)	
34249		Enable (disable) any implementation-defined special control characters not

34250 currently controlled by **icanon**, **isig**, **ixon**, or **ixoff**. This shall have the effect of
 34251 setting (not setting) IEXTEN in the **termios** *c_lflag* field, as defined in the Base
 34252 Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal
 34253 Interface.

34254 **echo** (**-echo**) Echo back (do not echo back) every character typed. This shall have the effect
 34255 of setting (not setting) ECHO in the **termios** *c_lflag* field, as defined in the Base
 34256 Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal
 34257 Interface.

34258 **echoe** (**-echoe**) The ERASE character visually erases (does not erase) the last character in the
 34259 current line from the display, if possible. This shall have the effect of setting
 34260 (not setting) ECHOE in the **termios** *c_lflag* field, as defined in the Base
 34261 Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal
 34262 Interface.

34263 **echok** (**-echok**) Echo (do not echo) NL after KILL character. This shall have the effect of
 34264 setting (not setting) ECHOK in the **termios** *c_lflag* field, as defined in the Base
 34265 Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal
 34266 Interface.

34267 **echonl** (**-echonl**) Echo (do not echo) NL, even if **echo** is disabled. This shall have the effect of
 34268 setting (not setting) ECHONL in the **termios** *c_lflag* field, as defined in the
 34269 Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General
 34270 Terminal Interface.

34271 **noflsh** (**-noflsh**) Disable (enable) flush after INTR, QUIT, SUSP. This shall have the effect of
 34272 setting (not setting) NOFLSH in the **termios** *c_lflag* field, as defined in the Base
 34273 Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal
 34274 Interface.

34275 **tostop** (**-tostop**) Send SIGTTOU for background output. This shall have the effect of setting
 34276 (not setting) TOSTOP in the **termios** *c_lflag* field, as defined in the Base
 34277 Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal
 34278 Interface.

34279 **Special Control Character Assignments**

34280 *<control>-character string*
 34281 Set *<control>-character* to *string*. If *<control>-character* is one of the character sequences in
 34282 the first column of the following table, the corresponding Base Definitions volume of
 34283 IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface control character from the
 34284 second column shall be recognized. This has the effect of setting the corresponding element
 34285 of the **termios** *c_cc* array (see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter
 34286 13, Headers, **<termios.h>**).

34287

Table 4-19 Control Character Names in *stty*

34288

34289

34290

34291

34292

34293

34294

34295

34296

34297

Control Character	c_cc Subscript	Description
eof	VEOF	EOF character
eol	VEOL	EOL character
erase	VERASE	ERASE character
intr	VINTR	INTR character
kill	VKILL	KILL character
quit	VQUIT	QUIT character
susp	VSUSP	SUSP character
start	VSTART	START character
stop	VSTOP	STOP character

34298

34299

34300

34301

34302

34303

34304

If *string* is a single character, the control character shall be set to that character. If *string* is the two-character sequence "^_" or the string *undef*, the control character shall be set to `_POSIX_VDISABLE`, if it is in effect for the device; if `_POSIX_VDISABLE` is not in effect for the device, it shall be treated as an error. In the POSIX locale, if *string* is a two-character sequence beginning with circumflex ('^'), and the second character is one of those listed in the "^c" column of the following table, the control character shall be set to the corresponding character value in the Value column of the table.

34305

Table 4-20 Circumflex Control Characters in *stty*

34306

34307

34308

34309

34310

34311

34312

34313

34314

34315

34316

34317

^c	Value	^c	Value	^c	Value
a, A	<SOH>	l, L	<FF>	w, W	<ETB>
b, B	<STX>	m, M	<CR>	x, X	<CAN>
c, C	<ETX>	n, N	<SO>	y, Y	
d, D	<EOT>	o, O	<SI>	z, Z	<SUB>
e, E	<ENQ>	p, P	<DLE>	[<ESC>
f, F	<ACK>	q, Q	<DC1>	\	<FS>
g, G	<BEL>	r, R	<DC2>]	<GS>
h, H	<BS>	s, S	<DC3>	^	<RS>
i, I	<HT>	t, T	<DC4>	_	<US>
j, J	<LF>	u, U	<NAK>	?	
k, K	<VT>	v, V	<SYN>		

34318

min number

34319

time number

34320

Set the value of **min** or **time** to *number*. MIN and TIME are used in non-canonical mode input processing (**icanon**).

34321

34322

Combination Modes

34323

saved settings

34324

Set the current terminal characteristics to the saved settings produced by the **-g** option.

34325

evenp or **parity**

34326

Enable **parenb** and **cs7**; disable **parodd**.

34327

oddp

34328

Enable **parenb**, **cs7**, and **parodd**.

34329

-parity, **-evenp**, or **-oddp**

34330

Disable **parenb**, and set **cs8**.

- 34331 XSI **raw** (**-raw** or **cooked**)
 34332 Enable (disable) raw input and output. Raw mode shall be equivalent to setting:
- ```
34333 stty cs8 erase ^- kill ^- intr ^- \

 34334 quit ^- eof ^- eol ^- -post -inpck
```
- 34335 **nl** (**-nl**)  
 34336 Enable (disable) **icrnl**. In addition, **-nl** unsets **inlcr** and **igncr**.
- 34337 **ek** Reset ERASE and KILL characters back to system defaults.
- 34338 **sane** Reset all modes to some reasonable, unspecified, values.
- 34339 **STDIN**  
 34340 Although no input is read from standard input, standard input is used to get the current  
 34341 terminal I/O characteristics and to set new terminal I/O characteristics.
- 34342 **INPUT FILES**  
 34343 None.
- 34344 **ENVIRONMENT VARIABLES**  
 34345 The following environment variables shall affect the execution of *stty*:
- 34346 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 34347 If *LANG* is unset or null, the corresponding value from the implementation-  
 34348 defined default locale shall be used. If any of the internationalization variables  
 34349 contains an invalid setting, the utility shall behave as if none of the variables had  
 34350 been defined.
- 34351 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 34352 internationalization variables.
- 34353 **LC\_CTYPE** This variable determines the locale for the interpretation of sequences of bytes of  
 34354 text data as characters (for example, single-byte as opposed to multi-byte  
 34355 characters in arguments) and which characters are in the class **print**.
- 34356 **LC\_MESSAGES**  
 34357 Determine the locale that should be used to affect the format and contents of  
 34358 diagnostic messages written to standard error.
- 34359 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 34360 **ASYNCHRONOUS EVENTS**  
 34361 Default.
- 34362 **STDOUT**  
 34363 If operands are specified, no output shall be produced.
- 34364 If the **-g** option is specified, *stty* shall write to standard output the current settings in a form that  
 34365 can be used as arguments to another instance of *stty* on the same system.
- 34366 If the **-a** option is specified, all of the information as described in the OPERANDS section shall  
 34367 be written to standard output. Unless otherwise specified, this information shall be written as  
 34368 <space>-separated tokens in an unspecified format, on one or more lines, with an unspecified  
 34369 number of tokens per line. Additional information may be written.
- 34370 If no options or operands are specified, an unspecified subset of the information written for the  
 34371 **-a** option shall be written.
- 34372 If speed information is written as part of the default output, or if the **-a** option is specified and if  
 34373 the terminal input speed and output speed are the same, the speed information shall be written



34374 as follows:

34375 "speed %d baud;", <speed>

34376 Otherwise, speeds shall be written as:

34377 "ispeed %d baud; ospeed %d baud;", <ispeed>, <ospeed>

34378 In locales other than the POSIX locale, the word **baud** may be changed to something more  
34379 appropriate in those locales.

34380 If control characters are written as part of the default output, or if the **-a** option is specified,  
34381 control characters shall be written as:

34382 "%s = %s;", <control-character name>, <value>

34383 where <value> is either the character, or some visual representation of the character if it is non-  
34384 printable, or the string *undef* if the character is disabled.

#### 34385 **STDERR**

34386 Used only for diagnostic messages.

#### 34387 **OUTPUT FILES**

34388 None.

#### 34389 **EXTENDED DESCRIPTION**

34390 None.

#### 34391 **EXIT STATUS**

34392 The following exit values shall be returned:

34393 0 The terminal options were read or set successfully.

34394 >0 An error occurred.

#### 34395 **CONSEQUENCES OF ERRORS**

34396 Default.

#### 34397 **APPLICATION USAGE**

34398 The **-g** flag is designed to facilitate the saving and restoring of terminal state from the shell level.  
34399 For example, a program may:

```
34400 saveterm="$(stty -g)" # save terminal state
34401 stty (new settings) # set new state
34402 ... # ...
34403 stty $saveterm # restore terminal state
```

34404 Since the format is unspecified, the saved value is not portable across systems.

34405 Since the **-a** format is so loosely specified, scripts that save and restore terminal settings should  
34406 use the **-g** option.

#### 34407 **EXAMPLES**

34408 None.

#### 34409 **RATIONALE**

34410 The original *stty* description was taken directly from System V and reflected the System V  
34411 terminal driver **termio**. It has been modified to correspond to the terminal driver **termios**.

34412 Output modes are specified only for XSI-conformant systems. All implementations are expected  
34413 to provide *stty* operands corresponding to all of the output modes they support.

34414 The *stty* utility is primarily used to tailor the user interface of the terminal, such as selecting the  
34415 preferred ERASE and KILL characters. As an application programming utility, *stty* can be used  
34416 within shell scripts to alter the terminal settings for the duration of the script.

34417 The **termios** section states that individual disabling of control characters is possible through the  
34418 option `_POSIX_VDISABLE`. If enabled, two conventions currently exist for specifying this:  
34419 System V uses "`^-`", and BSD uses *undef*. Both are accepted by *stty* in this volume of  
34420 IEEE Std. 1003.1-200x. The other BSD convention of using the letter '`u`' was rejected because it  
34421 conflicts with the actual letter '`u`', which is an acceptable value for a control character.

34422 Early proposals did not specify the mapping of "`^c`" to control characters because the control  
34423 characters were not specified in the POSIX locale character set description file requirements. The  
34424 control character set is now specified in the Base Definitions volume of IEEE Std. 1003.1-200x,  
34425 Chapter 3, Definitions so the historical mapping is specified. Note that although the mapping  
34426 corresponds to control-character key assignments on many terminals that use the  
34427 ISO/IEC 646:1991 standard (or ASCII) character encodings, the mapping specified here is to the  
34428 control characters, not their keyboard encodings.

34429 Since **termios** supports separate speeds for input and output, two new options were added to  
34430 specify each distinctly.

34431 Some historical implementations use standard input to get and set terminal characteristics;  
34432 others use standard output. Since input from a login TTY is usually restricted to the owner while  
34433 output to a TTY is frequently open to anyone, using standard input provides fewer chances of  
34434 accidentally (or maliciously) altering the terminal settings of other users. Using standard input  
34435 also allows *stty -a* and *stty -g* output to be redirected for later use. Therefore, usage of standard  
34436 input is required by this volume of IEEE Std. 1003.1-200x.

#### 34437 **FUTURE DIRECTIONS**

34438 None.

#### 34439 **SEE ALSO**

34440 The Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface

#### 34441 **CHANGE HISTORY**

34442 First released in Issue 2.

#### 34443 **Issue 4**

34444 Aligned with the ISO/IEC 9945-2:1993 standard.

#### 34445 **Issue 5**

34446 The description of **tabs** is clarified.

34447 **FUTURE DIRECTIONS** section added.

#### 34448 **Issue 6**

34449 The legacy items **iucl(-iucl)**, **xcase**, **olcuc(-olcuc)**, **lcase(-lcase)**, and **LCASE(-LCASE)**, are  
34450 removed.

## 34451 NAME

34452 tabs — set terminal tabs

## 34453 SYNOPSIS

34454 UP XSI tabs [ -n | -a | -a2 | -c | -c2 | -c3 | -f | -p | -s | -u ][ +m[n] ] [-T type]

34455 tabs [-T type][ +[n] ] n1[,n2,...]

34456

## 34457 DESCRIPTION

34458 The *tabs* utility shall display a series of characters that first clears the hardware terminal tab  
 34459 XSI settings and then initializes the tab stops at the specified positions and optionally adjusts the  
 34460 margin.

34461 The phrase “tab-stop position *N*” shall be taken to mean that, from the start of a line of output,  
 34462 tabbing to position *N* shall cause the next character output to be in the (*N*+1)th column position  
 34463 on that line. The maximum number of tab stops allowed is terminal-dependent.

34464 It need not be possible to implement *tabs* on certain terminals. If the terminal type obtained from  
 34465 the *TERM* environment variable or *-T* option represents such a terminal, an appropriate  
 34466 diagnostic message shall be written to standard error and *tabs* shall exit with a status greater  
 34467 than zero.

## 34468 OPTIONS

34469 The *tabs* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 34470 XSI 12.2, Utility Syntax Guidelines, except for various extensions: the options *-a2*, *-c2*, and *-c3* are  
 34471 multi-character.

34472 The following options shall be supported:

34473 *-n* Specify repetitive tab stops separated by a uniform number of column positions, *n*,  
 34474 where *n* is a single-digit decimal number. The default usage of *tabs* with no  
 34475 arguments shall be equivalent to *tabs-8*. When *-0* is used, the tab stops shall be  
 34476 cleared and no new ones set.

34477 XSI *-a* 1,10,16,36,72  
 34478 Assembler, applicable to some mainframes.

34479 XSI *-a2* 1,10,16,40,72  
 34480 Assembler, applicable to some mainframes.

34481 XSI *-c* 1,8,12,16,20,55  
 34482 COBOL, normal format.

34483 XSI *-c2* 1,6,10,14,49  
 34484 COBOL, compact format (columns 1-6 omitted).

34485 XSI *-c3* 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67  
 34486 COBOL compact format (columns 1-6 omitted), with more tabs than *-c2*.

34487 XSI *-f* 1,7,11,15,19,23  
 34488 FORTRAN

34489 XSI *-p* 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61  
 34490 PL/1

34491 XSI *-s* 1,10,55  
 34492 SNOBOL

34493 XSI *-u* 1,12,20,44  
 34494 Assembler, applicable to some mainframes.

34495            -T *type*        Indicate the type of terminal. If this option is not supplied and the *TERM* variable  
 34496                                    is unset or null, an unspecified default terminal type shall be used. The setting of  
 34497                                    *type* shall take precedence over the value in *TERM*.

34498 **OPERANDS**

34499            The following operand shall be supported:

34500            *n1*[,*n2*,...]    A single command line argument that consists of tab-stop values separated using  
 34501                                    either commas or <blank> characters. The application shall ensure that the tab-  
 34502                                    stop values are positive decimal integers in strictly ascending order. If any number  
 34503                                    (except the first one) is preceded by a plus sign, it is taken as an increment to be  
 34504                                    added to the previous value. For example, the tab lists 1,10,20,30 and 1,10,+10,+10  
 34505                                    are considered to be identical.

34506 **STDIN**

34507            Not used.

34508 **INPUT FILES**

34509            None.

34510 **ENVIRONMENT VARIABLES**

34511            The following environment variables shall affect the execution of *tabs*:

34512            *LANG*            Provide a default value for the internationalization variables that are unset or null.  
 34513                                    If *LANG* is unset or null, the corresponding value from the implementation-  
 34514                                    defined default locale shall be used. If any of the internationalization variables  
 34515                                    contains an invalid setting, the utility shall behave as if none of the variables had  
 34516                                    been defined.

34517            *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
 34518                                    internationalization variables.

34519            *LC\_CTYPE*    Determine the locale for the interpretation of sequences of bytes of text data as  
 34520                                    characters (for example, single-byte as opposed to multi-byte characters in  
 34521                                    arguments).

34522            *LC\_MESSAGES*

34523                                    Determine the locale that should be used to affect the format and contents of  
 34524                                    diagnostic messages written to standard error.

34525 XSI        *NLSPATH*    Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

34526            *TERM*            Determine the terminal type. If this variable is unset or null, and if the -T option is  
 34527                                    not specified, an unspecified default terminal type shall be used.

34528 **ASYNCHRONOUS EVENTS**

34529            Default.

34530 **STDOUT**

34531            If standard output is a terminal, the appropriate sequence to clear and set the tab stops may be  
 34532                                    written to standard output in an unspecified format. If standard output is not a terminal,  
 34533                                    undefined results occur.

34534 **STDERR**

34535            Used only for diagnostic messages.

34536 **OUTPUT FILES**

34537 None.

34538 **EXTENDED DESCRIPTION**

34539 None.

34540 **EXIT STATUS**

34541 The following exit values shall be returned:

34542 0 Successful completion.

34543 &gt;0 An error occurred.

34544 **CONSEQUENCES OF ERRORS**

34545 Default.

34546 **APPLICATION USAGE**34547 This utility makes use of the terminal's hardware tabs and the *stty tabs* option.

34548 This utility is not recommended for application use.

34549 Some integrated display units might not have escape sequences to set tab stops, but may be set  
 34550 by internal system calls. On these terminals, *tabs* works if standard output is directed to the  
 34551 terminal; if output is directed to another file, however, *tabs* fails.

34552 **EXAMPLES**

34553 None.

34554 **RATIONALE**

34555 Consideration was given to having the *tput* utility handle all of the functions described in *tabs*.  
 34556 However, the separate *tabs* utility was retained because it seems more intuitive to use a  
 34557 command named *tabs* than *tput* with a new option. The POSIX Shell and Utilities *tput* does not  
 34558 support setting or clearing tabs, and no known historical version of *tabs* supports the capability  
 34559 of setting arbitrary tab stops.

34560 The System V *tabs* interface is very complex; the version in this volume of IEEE Std. 1003.1-200x  
 34561 has a reduced feature list. There was considerable sentiment for specifying only a means of  
 34562 resetting the tabs back to a known state—presumably the “standard” of tabs every eight  
 34563 positions. The following features were omitted:

- 34564 • Setting tab stops tailored for certain programming languages; the standard developers were  
 34565 concerned that it would be difficult to decide which languages to include and where the tabs  
 34566 should be.
- 34567 • Setting tab stops via the first line in a file, using *—file*. Since even the SVID has no complete  
 34568 explanation of this feature, it is doubtful that it is in widespread use.
- 34569 • Setting the left margin using *+mn*. As this does not work with all terminal types, it was  
 34570 omitted.

34571 In an early proposal, a *-t tablist* option was added for consistency with *expand*; this was later  
 34572 removed when inconsistencies with the historical list of tabs were identified.

34573 Consideration was given to adding a *-p* option that would output the current tab settings so  
 34574 that they could be saved and then later restored. This was not accepted because querying the tab  
 34575 stops of the terminal is not a capability in historical *terminfo* or *termcap* facilities and might not be  
 34576 supported on a wide range of terminals.

34577 **FUTURE DIRECTIONS**

34578 None.

34579 **SEE ALSO**34580 *expand, stty, unexpand*34581 **CHANGE HISTORY**

34582 First released in Issue 2.

34583 **Issue 4**

34584 Aligned with the ISO/IEC 9945-2:1993 standard.

34585 **Issue 6**

34586 This utility is now marked as part of the User Portability Utilities option.

34587 The normative text is reworded to avoid use of the term “must” for application requirements.

34588 **NAME**

34589 tail — copy the last part of a file

34590 **SYNOPSIS**34591 tail [-f][ -c *number*| -n *number*][*file*]34592 **DESCRIPTION**34593 The *tail* utility shall copy its input file to the standard output beginning at a designated place.

34594 Copying shall begin at the point in the file indicated by the *-c number* or *-n number* options. The  
 34595 option-argument *number* shall be counted in units of lines or bytes, according to the options *-n*  
 34596 and *-c*. Both line and byte counts start from 1.

34597 Tails relative to the end of the file may be saved in an internal buffer, and thus may be limited in  
 34598 length. Such a buffer, if any, is no smaller than {LINE\_MAX}\*10 bytes.

34599 **OPTIONS**

34600 The *tail* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 34601 12.2, Utility Syntax Guidelines.

34602 The following options shall be supported:

34603 *-c number* The application shall ensure that the *number* option-argument is a decimal integer  
 34604 whose sign affects the location in the file, measured in bytes, to begin the copying:

34605

34606

34607

34608

| Sign        | Copying Starts                         |
|-------------|----------------------------------------|
| +           | Relative to the beginning of the file. |
| -           | Relative to the end of the file.       |
| <i>none</i> | Relative to the end of the file.       |

34609 The origin for counting shall be 1; that is, *-c +1* represents the first byte of the file,  
 34610 *-c -1* the last.

34611 *-f* If the input file is a regular file or if the *file* operand specifies a FIFO, do not  
 34612 terminate after the last line of the input file has been copied, but read and copy  
 34613 further bytes from the input file when they become available. If no *file* operand is  
 34614 specified and standard input is a pipe, the *-f* option shall be ignored. If the input  
 34615 file is not a FIFO, pipe, or regular file, it is unspecified whether or not the *-f* option  
 34616 shall be ignored.

34617 *-n number* This option is equivalent to *-c number*, except the starting location in the file shall  
 34618 be measured in lines instead of bytes. The origin for counting shall be 1; that is, *-n*  
 34619 *+1* represents the first line of the file, *-n -1* the last.

34620 If neither *-c* nor *-n* is specified, *-n 10* shall be assumed.34621 **OPERANDS**

34622 The following operand shall be supported:

34623 *file* A path name of an input file. If no *file* operands are specified, the standard input  
 34624 shall be used.

34625 **STDIN**

34626 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
 34627 section.

34628 **INPUT FILES**

34629           If the `-c` option is specified, the input file can contain arbitrary data; otherwise, the input file  
34630           shall be a text file.

34631 **ENVIRONMENT VARIABLES**

34632           The following environment variables shall affect the execution of *tail*:

34633           *LANG*       Provide a default value for the internationalization variables that are unset or null.  
34634                       If *LANG* is unset or null, the corresponding value from the implementation-  
34635                       defined default locale shall be used. If any of the internationalization variables  
34636                       contains an invalid setting, the utility shall behave as if none of the variables had  
34637                       been defined.

34638           *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
34639                       internationalization variables.

34640           *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
34641                       characters (for example, single-byte as opposed to multi-byte characters in  
34642                       arguments and input files).

34643           *LC\_MESSAGES*

34644                       Determine the locale that should be used to affect the format and contents of  
34645                       diagnostic messages written to standard error.

34646 XSI       *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

34647 **ASYNCHRONOUS EVENTS**

34648           Default.

34649 **STDOUT**

34650           The designated portion of the input file shall be written to standard output.

34651 **STDERR**

34652           Used only for diagnostic messages.

34653 **OUTPUT FILES**

34654           None.

34655 **EXTENDED DESCRIPTION**

34656           None.

34657 **EXIT STATUS**

34658           The following exit values shall be returned:

34659           0   Successful completion.

34660           >0 An error occurred.

34661 **CONSEQUENCES OF ERRORS**

34662           Default.



34663 **APPLICATION USAGE**

34664 The `-c` option should be used with caution when the input is a text file containing multi-byte  
34665 characters; it may produce output that does not start on a character boundary.

34666 Although the input file to *tail* can be any type, the results might not be what would be expected  
34667 on some character special device files or on file types not described by the System Interfaces  
34668 volume of IEEE Std. 1003.1-200x. Since this volume of IEEE Std. 1003.1-200x does not specify the  
34669 block size used when doing input, *tail* need not read all of the data from devices that only  
34670 perform block transfers.

34671 **EXAMPLES**

34672 The `-f` option can be used to monitor the growth of a file that is being written by some other  
34673 process. For example, the command:

```
34674 tail -f fred
```

34675 prints the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between  
34676 the time *tail* is initiated and killed. As another example, the command:

```
34677 tail -f -c 15 fred
```

34678 prints the last 15 bytes of the file **fred**, followed by any bytes that are appended to **fred** between  
34679 the time *tail* is initiated and killed.

34680 **RATIONALE**

34681 This version of *tail* was created to allow conformance to the Utility Syntax Guidelines. The  
34682 historical `-b` option was omitted because of the general non-portability of block-sized units of  
34683 text. The `-c` option historically meant “characters”, but this volume of IEEE Std. 1003.1-200x  
34684 indicates that it means “bytes”. This was selected to allow reasonable implementations when  
34685 multi-byte characters are possible; it was not named `-b` to avoid confusion with the historical  
34686 `-b`.

34687 The origin of counting both lines and bytes is 1, matching all widespread historical  
34688 implementations.

34689 The restriction on the internal buffer is a compromise between the historical System V  
34690 implementation of 4 096 bytes and the BSD 32 768 bytes.

34691 The `-f` option has been implemented as a loop that sleeps for 1 second and copies any bytes that  
34692 are available. This is sufficient, but if more efficient methods of determining when new data are  
34693 available are developed, implementations are encouraged to use them.

34694 Historical documentation indicates that *tail* ignores the `-f` option if the input file is a pipe (pipe  
34695 and FIFO on systems that support FIFOs). On BSD-based systems, this has been true; on System  
34696 V-based systems, this was true when input was taken from standard input, but it did not ignore  
34697 the `-f` flag if a FIFO was named as the *file* operand. Since the `-f` option is not useful on pipes and  
34698 all historical implementations ignore `-f` if no *file* operand is specified and standard input is a  
34699 pipe, this volume of IEEE Std. 1003.1-200x requires this behavior. However, since the `-f` option is  
34700 useful on a FIFO, this volume of IEEE Std. 1003.1-200x also requires that if standard input is a  
34701 FIFO or a FIFO is named, the `-f` option shall not be ignored. Although historical behavior does  
34702 not ignore the `-f` option for other file types, this is unspecified so that implementations are  
34703 allowed to ignore the `-f` option if it is known that the file cannot be extended.

34704 This was changed to the current form based on comments noting that `-c` was almost never used  
34705 without specifying a number and that there was no need to specify `-l` if `-n number` was given.

34706 **FUTURE DIRECTIONS**

34707 None.

34708 **SEE ALSO**34709 *head*34710 **CHANGE HISTORY**

34711 First released in Issue 2.

34712 **Issue 4**

34713 Aligned with the ISO/IEC 9945-2:1993 standard.

34714 **Issue 6**

34715 The obsolescent SYNOPSIS lines and associated text are removed.

34716 The normative text is reworded to avoid use of the term “must” for application requirements.

## 34717 NAME

34718 talk — talk to another user

## 34719 SYNOPSIS

34720 UP `talk address [terminal]`

34721

## 34722 DESCRIPTION

34723 The *talk* utility is a two-way, screen-oriented communication program.34724 When first invoked, *talk* shall send a message similar to:34725 Message from *<unspecified string>*34726 talk: connection requested by *your\_address*34727 talk: respond with: talk *your\_address*34728 to the specified *address*. At this point, the recipient of the message can reply by typing:34729 `talk your_address`34730 Once communication is established, the two parties can type simultaneously, with their output  
34731 displayed in separate regions of the screen. Characters shall be processed as follows:

- 34732 • Typing the alert character shall alert the recipient's terminal.
- 34733 • Typing `<control>-L` shall cause the sender's screen regions to be refreshed.
- 34734 • Typing the erase and kill characters shall affect the sender's terminal in the manner described  
34735 by the **termios** interface in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11,  
34736 General Terminal Interface.
- 34737 • Typing the interrupt or end-of-file characters shall terminate the local *talk* utility. Once the  
34738 *talk* session has been terminated on one side, the other side of the *talk* session shall be notified  
34739 that the *talk* session has been terminated and shall be able to do nothing except exit.
- 34740 • Typing characters from *LC\_CTYPE* classifications **print** or **space** shall cause those characters  
34741 to be sent to the recipient's terminal.
- 34742 • When and only when the *stty ixten* local mode is enabled, the existence and processing of  
34743 additional special control characters and multi-byte or single-byte functions shall be  
34744 implementation-defined.
- 34745 • Typing other non-printable characters shall cause implementation-defined sequences of  
34746 printable characters to be sent to the recipient's terminal.

34747 Permission to be a recipient of a *talk* message can be denied or granted by use of the *mesg* utility.  
34748 However, a user's privilege may further constrain the domain of accessibility of other users'  
34749 terminals. The *talk* utility shall fail when the user lacks the appropriate privileges to perform the  
34750 requested action.

34751 Certain block-mode terminals do not have all the capabilities necessary to support the  
34752 simultaneous exchange of messages required for *talk*. When this type of exchange cannot be  
34753 supported on such terminals, the implementation may support an exchange with reduced levels  
34754 of simultaneous interaction or it may report an error describing the terminal-related deficiency.

## 34755 OPTIONS

34756 None.

34757 **OPERANDS**

34758 The following operands shall be supported:

34759 *address* The recipient of the *talk* session. One form of *address* is the *<user name>*, as returned  
 34760 by the *who* utility. Other address formats and how they are handled are  
 34761 unspecified.

34762 *terminal* If the recipient is logged in more than once, the *terminal* argument can be used to  
 34763 indicate the appropriate terminal name. If *terminal* is not specified, the *talk* message  
 34764 shall be displayed on one or more accessible terminals in use by the recipient. The  
 34765 format of *terminal* shall be the same as that returned by the *who* utility.

34766 **STDIN**

34767 Characters read from standard input shall be copied to the recipient's terminal in an unspecified  
 34768 manner. If standard input is not a terminal, *talk* shall write a diagnostic message and exit with a  
 34769 non-zero status.

34770 **INPUT FILES**

34771 None.

34772 **ENVIRONMENT VARIABLES**

34773 The following environment variables shall affect the execution of *talk*:

34774 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 34775 If *LANG* is unset or null, the corresponding value from the implementation-  
 34776 defined default locale shall be used. If any of the internationalization variables  
 34777 contains an invalid setting, the utility shall behave as if none of the variables had  
 34778 been defined.

34779 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 34780 internationalization variables.

34781 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 34782 characters (for example, single-byte as opposed to multi-byte characters in  
 34783 arguments and input files). If the recipient's locale does not use an *LC\_CTYPE*  
 34784 equivalent to the sender's, the results are undefined.

34785 *LC\_MESSAGES*

34786 Determine the locale that should be used to affect the format and contents of  
 34787 diagnostic messages written to standard error and informative messages written to  
 34788 standard output.

34789 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

34790 *TERM* Determine the name of the invoker's terminal type. If this variable is unset or null,  
 34791 an unspecified default terminal type shall be used.

34792 **ASYNCHRONOUS EVENTS**

34793 When the *talk* utility receives a SIGINT signal, the utility shall terminate and exit with a zero  
 34794 status. It shall take the standard action for all other signals.

34795 **STDOUT**

34796 If standard output is a terminal, characters copied from the recipient's standard input may be  
 34797 written to standard output. Standard output also may be used for diagnostic messages. If  
 34798 standard output is not a terminal, *talk* shall exit with a non-zero status.

34799 **STDERR**

34800 None.

34801 **OUTPUT FILES**

34802 None.

34803 **EXTENDED DESCRIPTION**

34804 None.

34805 **EXIT STATUS**

34806 The following exit values shall be returned:

34807 0 Successful completion.

34808 >0 An error occurred or *talk* was invoked on a terminal incapable of supporting it.34809 **CONSEQUENCES OF ERRORS**

34810 Default.

34811 **APPLICATION USAGE**

34812 Because the handling of non-printable, non-`<space>` characters is tied to the *stty* description of

34813 **ixten**, implementation extensions within the terminal driver can be accessed. For example,

34814 some implementations provide line editing functions with certain control character sequences.

34815 **EXAMPLES**

34816 None.

34817 **RATIONALE**

34818 The *write* utility was included in this volume of IEEE Std. 1003.1-200x since it can be

34819 implemented on all terminal types. The *talk* utility, which cannot be implemented on certain

34820 terminals, was considered to be a “better” communications interface. Both of these programs are

34821 in widespread use on historical implementations. Therefore, both utilities have been specified.

34822 All references to networking abilities (*talking* to a user on another system) were removed as

34823 being outside the scope of this volume of IEEE Std. 1003.1-200x.

34824 Historical BSD and System V versions of *talk* terminate both of the conversations when either

34825 user breaks out of the session. This can lead to adverse consequences if a user unwittingly

34826 continues to enter text that is interpreted by the shell when the other terminates the session.

34827 Therefore, the version of *talk* specified by this volume of IEEE Std. 1003.1-200x requires both

34828 users to terminate their end of the session explicitly.

34829 Only messages sent to the terminal of the invoking user can be internationalized in any way:

- 34830 • The original “Message from *<unspecified string>* ...” message sent to the terminal of the
- 34831 recipient cannot be internationalized because the environment of the recipient is as yet
- 34832 inaccessible to the *talk* utility. The environment of the invoking party is irrelevant.

- 34833 • Subsequent communication between the two parties cannot be internationalized because the
- 34834 two parties may specify different languages in their environment (and non-portable
- 34835 characters cannot be mapped from one language to another).

- 34836 • Neither party can be required to communicate in a language other than C and/or the one
- 34837 specified by their environment because unavailable terminal hardware support (for example,
- 34838 fonts) may be required.

34839 The text in the STDOUT section reflects the usage of the verb “display” in this section; some *talk*

34840 implementations actually use standard output to write to the terminal, but this volume of

34841 IEEE Std. 1003.1-200x does not require that to be the case.

34842 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*  
34843 require that they all use or accept the same format.

34844 The handling of non-printable characters is partially implementation-defined because the details  
34845 of mapping them to printable sequences is not needed by the user. Historical implementations,  
34846 for security reasons, disallow the transmission of non-printable characters that may send  
34847 commands to the other terminal.

34848 **FUTURE DIRECTIONS**

34849 None.

34850 **SEE ALSO**

34851 *msg*, *who*, *write*, the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General  
34852 Terminal Interface

34853 **CHANGE HISTORY**

34854 First released in Issue 4.

34855 **Issue 6**

34856 This utility is now marked as part of the User Portability Utilities option.

34857 **NAME**

34858           tee — duplicate standard input

34859 **SYNOPSIS**34860           tee [-ai][*file...*]34861 **DESCRIPTION**34862           The *tee* utility shall copy standard input to standard output, making a copy in zero or more files.34863           The *tee* utility shall not buffer output.34864           If the **-a** option is not specified, output files shall be written (see Section 1.7.1.4 (on page 2209)).34865 **OPTIONS**34866           The *tee* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
34867           12.2, Utility Syntax Guidelines.

34868           The following options shall be supported:

34869           **-a**           Append the output to the files.34870           **-i**           Ignore the SIGINT signal.34871 **OPERANDS**

34872           The following operands shall be supported:

34873           *file*           A path name of an output file. Processing of at least 13 *file* operands shall be  
34874           supported.34875 **STDIN**

34876           The standard input can be of any type.

34877 **INPUT FILES**

34878           None.

34879 **ENVIRONMENT VARIABLES**34880           The following environment variables shall affect the execution of *tee*:34881           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
34882           If *LANG* is unset or null, the corresponding value from the implementation-  
34883           defined default locale shall be used. If any of the internationalization variables  
34884           contains an invalid setting, the utility shall behave as if none of the variables had  
34885           been defined.34886           **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
34887           internationalization variables.34888           **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as  
34889           characters (for example, single-byte as opposed to multi-byte characters in  
34890           arguments).34891           **LC\_MESSAGES**34892           Determine the locale that should be used to affect the format and contents of  
34893           diagnostic messages written to standard error.34894 **XSI**           **NLS\_PATH**       Determine the location of message catalogs for the processing of *LC\_MESSAGES*.34895 **ASYNCHRONOUS EVENTS**34896           Default, except that if the **-i** option was specified, SIGINT shall be ignored.

34897 **STDOUT**

34898       The standard output shall be a copy of the standard input.

34899 **STDERR**

34900       Used only for diagnostic messages.

34901 **OUTPUT FILES**

34902       If any *file* operands are specified, the standard input shall be copied to each named file.

34903 **EXTENDED DESCRIPTION**

34904       None.

34905 **EXIT STATUS**

34906       The following exit values shall be returned:

34907       0   The standard input was successfully copied to all output files.

34908       >0  An error occurred.

34909 **CONSEQUENCES OF ERRORS**

34910       If a write to any successfully opened *file* operand fails, writes to other successfully opened *file* operands and standard output shall continue, but the exit status shall be non-zero. Otherwise, the default actions specified in Section 1.11 (on page 2224) apply.

34913 **APPLICATION USAGE**

34914       The *tee* utility is usually used in a pipeline, to make a copy of the output of some utility.

34915       The *file* operand is technically optional, but *tee* is no more useful than *cat* when none is specified.

34916 **EXAMPLES**

34917       Save an unsorted intermediate form of the data in a pipeline:

34918       ... | tee unsorted | sort > sorted

34919 **RATIONALE**

34920       The buffering requirement means that *tee* is not allowed to use ISO C standard fully buffered or line-buffered writes. It does not mean that *tee* has to do 1-byte reads followed by 1-byte writes.

34922       It should be noted that early versions of BSD ignore any invalid options and accept a single '-' as an alternative to -i. They also print a message if unable to open a file:

34924       "tee: cannot access %s\n", <pathname>

34925       Historical implementations ignore write errors. This is explicitly not permitted by this volume of IEEE Std. 1003.1-200x.

34927       Some historical implementations use O\_APPEND when providing append mode; others use the *lseek()* function to seek to the end of file after opening the file without O\_APPEND. This volume of IEEE Std. 1003.1-200x requires functionality equivalent to using O\_APPEND; see Section 1.7.1.4 (on page 2209).

34931 **FUTURE DIRECTIONS**

34932       None.

34933 **SEE ALSO**

34934       *cat*

34935 **CHANGE HISTORY**

34936       First released in Issue 2.



- 34937 **Issue 4**
- 34938 Aligned with the ISO/IEC 9945-2: 1993 standard.
- 34939 **Issue 6**
- 34940 IEEE PASC Interpretation 1003.2 #168 is applied.

34941 **NAME**

34942 test — evaluate expression

34943 **SYNOPSIS**34944 test [*expression*]34945 [ [*expression*] ]34946 **DESCRIPTION**

34947 The *test* utility shall evaluate the *expression* and indicates the result of the evaluation by its exit  
 34948 status. An exit status of zero indicates that the expression evaluated as true and an exit status of  
 34949 1 indicates that the expression evaluated as false.

34950 In the second form of the utility, which uses "[ ]" rather than *test*, the application shall ensure  
 34951 that the square brackets are separate arguments.

34952 **OPTIONS**

34953 The *test* utility shall not recognize the "--" argument in the manner specified by guideline 10 in  
 34954 the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

34955 No options shall be supported.

34956 **OPERANDS**

34957 The application shall ensure that all operators and elements of primaries are presented as  
 34958 separate arguments to the *test* utility.

34959 The following primaries can be used to construct *expression*:

34960 **-b file** True if *file* exists and is a block special file.

34961 **-c file** True if *file* exists and is a character special file.

34962 **-d file** True if *file* exists and is a directory.

34963 **-e file** True if *file* exists.

34964 **-f file** True if *file* exists and is a regular file.

34965 **-g file** True if *file* exists and its set group ID flag is set.

34966 **-h file** True if *file* exists and is a symbolic link.

34967 **-n string** True if the length of *string* is non-zero.

34968 **-p file** True if *file* is a named pipe (FIFO).

34969 **-r file** True if *file* exists and is readable. True shall indicate that permission to read from  
 34970 *file* will be granted, as defined in Section 1.7.1.4 (on page 2209).

34971 **-s file** True if *file* exists and has a size greater than zero.

34972 **-t file\_descriptor**

34973 True if the file whose file descriptor number is *file\_descriptor* is open and is  
 34974 associated with a terminal.

34975 **-u file** True if *file* exists and its set-user-ID flag is set.

34976 **-w file** True if *file* exists and is writable. True shall indicate that permission to write from  
 34977 *file* will be granted, as defined in Section 1.7.1.4 (on page 2209).

34978 **-x file** True if *file* exists and is executable. True if *file* exists and is executable. True shall  
 34979 indicate that permission to execute *file* will be granted, as defined in Section 1.7.1.4  
 34980 (on page 2209). If *file* is a directory, true shall indicate that permission to search *file*  
 34981 will be granted.

- 34982      **-z string**      True if the length of string *string* is zero.
- 34983      **string**            True if the string *string* is not the null string.
- 34984      **s1 = s2**            True if the strings *s1* and *s2* are identical.
- 34985      **s1 != s2**           True if the strings *s1* and *s2* are not identical.
- 34986      **n1 -eq n2**          True if the integers *n1* and *n2* are algebraically equal.
- 34987      **n1 -ne n2**          True if the integers *n1* and *n2* are not algebraically equal.
- 34988      **n1 -gt n2**          True if the integer *n1* is algebraically greater than the integer *n2*.
- 34989      **n1 -ge n2**          True if the integer *n1* is algebraically greater than or equal to the integer *n2*.
- 34990      **n1 -lt n2**          True if the integer *n1* is algebraically less than the integer *n2*.
- 34991      **n1 -le n2**          True if the integer *n1* is algebraically less than or equal to the integer *n2*.
- 34992 XSI      **expression1 -a expression2**  
 34993                    True if both *expression1* and *expression2* are true. The **-a** binary primary is left  
 34994                    associative. It has a higher precedence than **-o**.
- 34995 XSI      **expression1 -o expression2**  
 34996                    True if either *expression1* or *expression2* is true. The **-o** binary primary is left  
 34997                    associative.
- 34998      With the exception of the **-h file** primary, if a *file* argument is a symbolic link, *test* shall evaluate  
 34999      the expression by resolving the symbolic link and using the file referenced by the link.
- 35000      These primaries can be combined with the following operators:
- 35001      **! expression**      True if *expression* is false.
- 35002 XSI      **( expression )**      True if *expression* is true. The parentheses can be used to alter the normal  
 35003                    precedence and associativity.
- 35004      The primaries with two elements of the form:
- 35005      *-primary\_operator primary\_operand*
- 35006      are known as *unary primaries*. The primaries with three elements in either of the two forms:
- 35007      *primary\_operand -primary\_operator primary\_operand*
- 35008      *primary\_operand primary\_operator primary\_operand*
- 35009      are known as *binary primaries*. Additional implementation-defined operators and  
 35010      *primary\_operators* may be provided by implementations. They shall be of the form *-operator*  
 35011      where the first character of *operator* is not a digit.
- 35012      The algorithm for determining the precedence of the operators and the return value that shall be  
 35013      generated is based on the number of arguments presented to *test*. (However, when using the  
 35014      "[...]" form, the right-bracket final argument shall not be counted in this algorithm.)
- 35015      In the following list, \$1, \$2, \$3, and \$4 represent the arguments presented to *test*:
- 35016      0 arguments:      Exit false (1).
- 35017      1 argument:        Exit true (0) if \$1 is not null; otherwise, exit false.
- 35018      2 arguments:      • If \$1 is '!', exit true if \$2 is null, false if \$2 is not null.
- 35019                      • If \$1 is a unary primary, exit true if the unary test is true, false if the unary  
 35020                      test is false.

|           |                              |                                                                                                                                                                                                                                                                                                                                                                |
|-----------|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 35021     |                              | <ul style="list-style-type: none"><li>• Otherwise, produce unspecified results.</li></ul>                                                                                                                                                                                                                                                                      |
| 35022     | 3 arguments:                 | <ul style="list-style-type: none"><li>• If \$2 is a binary primary, perform the binary test of \$1 and \$3.</li></ul>                                                                                                                                                                                                                                          |
| 35023     |                              | <ul style="list-style-type: none"><li>• If \$1 is '!', negate the two-argument test of \$2 and \$3.</li></ul>                                                                                                                                                                                                                                                  |
| 35024     |                              | <ul style="list-style-type: none"><li>• If \$1 is '( ' and \$3 is ') ', perform the unary test of \$2.</li></ul>                                                                                                                                                                                                                                               |
| 35025     |                              | <ul style="list-style-type: none"><li>• Otherwise, produce unspecified results.</li></ul>                                                                                                                                                                                                                                                                      |
| 35026     | 4 arguments:                 | <ul style="list-style-type: none"><li>• If \$1 is '!', negate the three-argument test of \$2, \$3, and \$4.</li></ul>                                                                                                                                                                                                                                          |
| 35027 XSI |                              | <ul style="list-style-type: none"><li>• If \$1 is '( ' and \$4 is ') ', perform the two-argument test of \$2 and \$3.</li></ul>                                                                                                                                                                                                                                |
| 35028     |                              | <ul style="list-style-type: none"><li>• Otherwise, the results are unspecified.</li></ul>                                                                                                                                                                                                                                                                      |
| 35029 XSI | >4 arguments:                | The results are unspecified. On XSI-conformant systems, combinations of primaries and operators shall be evaluated using the precedence and associativity rules described previously. In addition, the string comparison binary primaries '=' and '!=' shall have a higher precedence than any unary primary.                                                  |
| 35030     |                              |                                                                                                                                                                                                                                                                                                                                                                |
| 35031     |                              |                                                                                                                                                                                                                                                                                                                                                                |
| 35032     |                              |                                                                                                                                                                                                                                                                                                                                                                |
| 35033     |                              |                                                                                                                                                                                                                                                                                                                                                                |
| 35034     | <b>STDIN</b>                 |                                                                                                                                                                                                                                                                                                                                                                |
| 35035     |                              | Not used.                                                                                                                                                                                                                                                                                                                                                      |
| 35036     | <b>INPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                |
| 35037     |                              | None.                                                                                                                                                                                                                                                                                                                                                          |
| 35038     | <b>ENVIRONMENT VARIABLES</b> |                                                                                                                                                                                                                                                                                                                                                                |
| 35039     |                              | The following environment variables shall affect the execution of <i>test</i> :                                                                                                                                                                                                                                                                                |
| 35040     | <i>LANG</i>                  | Provide a default value for the internationalization variables that are unset or null. If <i>LANG</i> is unset or null, the corresponding value from the implementation-defined default locale shall be used. If any of the internationalization variables contains an invalid setting, the utility shall behave as if none of the variables had been defined. |
| 35041     |                              |                                                                                                                                                                                                                                                                                                                                                                |
| 35042     |                              |                                                                                                                                                                                                                                                                                                                                                                |
| 35043     |                              |                                                                                                                                                                                                                                                                                                                                                                |
| 35044     |                              |                                                                                                                                                                                                                                                                                                                                                                |
| 35045     | <i>LC_ALL</i>                | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                       |
| 35046     |                              |                                                                                                                                                                                                                                                                                                                                                                |
| 35047     | <i>LC_CTYPE</i>              | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).                                                                                                                                                                                      |
| 35048     |                              |                                                                                                                                                                                                                                                                                                                                                                |
| 35049     |                              |                                                                                                                                                                                                                                                                                                                                                                |
| 35050     | <i>LC_MESSAGES</i>           |                                                                                                                                                                                                                                                                                                                                                                |
| 35051     |                              | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                                   |
| 35052     |                              |                                                                                                                                                                                                                                                                                                                                                                |
| 35053 XSI | <i>NLSPATH</i>               | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                          |
| 35054     | <b>ASYNCHRONOUS EVENTS</b>   |                                                                                                                                                                                                                                                                                                                                                                |
| 35055     |                              | Default.                                                                                                                                                                                                                                                                                                                                                       |
| 35056     | <b>STDOUT</b>                |                                                                                                                                                                                                                                                                                                                                                                |
| 35057     |                              | Not used.                                                                                                                                                                                                                                                                                                                                                      |
| 35058     | <b>STDERR</b>                |                                                                                                                                                                                                                                                                                                                                                                |
| 35059     |                              | Used only for diagnostic messages.                                                                                                                                                                                                                                                                                                                             |

35060 **OUTPUT FILES**

35061 None.

35062 **EXTENDED DESCRIPTION**

35063 None.

35064 **EXIT STATUS**

35065 The following exit values shall be returned:

35066 0 *expression* evaluated to true.35067 1 *expression* evaluated to false or *expression* was missing.

35068 &gt;1 An error occurred.

35069 **CONSEQUENCES OF ERRORS**

35070 Default.

35071 **APPLICATION USAGE**

35072 Scripts should be careful when dealing with user-supplied input that could be confused with  
 35073 primaries and operators. Unless the application writer knows all the cases that produce input to  
 35074 the script, invocations like:

35075 `test "$1" -a "$2"`

35076 should be written as:

35077 `test "$1" && test "$2"`

35078 to avoid problems if a user supplied values such as \$1 set to '!' and \$2 set to the null string.  
 35079 That is, in cases where maximal portability is of concern, replace:

35080 `test expr1 -a expr2`

35081 with:

35082 `test expr1 && test expr2`

35083 and replace:

35084 `test expr1 -o expr2`

35085 with:

35086 `test expr1 || test expr2`

35087 but note that, in *test*, `-a` has higher precedence than `-o` while `&&` and `||` have equal  
 35088 precedence in the shell.

35089 Parentheses or braces can be used in the shell command language to effect grouping.

35090 Parentheses must be escaped when using *sh*; for example:35091 `test \( expr1 -a expr2 \) -o expr3`

35092 This command is not always portable outside XSI-conformant systems. The following form can  
 35093 be used instead:

35094 `( test expr1 && test expr2 ) || test expr3`

35095 The two commands:

35096 `test "$1"`35097 `test ! "$1"`

35098 could not be used reliably on some historical systems. Unexpected results would occur if such a  
 35099 *string* expression were used and \$1 expanded to '!', '( ', or a known unary primary. Better  
 35100 constructs are:

```
35101 test -n "$1"
35102 test -z "$1"
```

35103 respectively.

35104 Historical systems have also been unreliable given the common construct:

```
35105 test "$response" = "expected string"
```

35106 One of the following is a more reliable form:

```
35107 test "X$response" = "Xexpected string"
35108 test "expected string" = "$response"
```

35109 Note that the second form assumes that *expected string* could not be confused with any unary  
 35110 primary. If *expected string* starts with '- ', '( ', '! ', or even '= ', the first form should be used  
 35111 instead. Using the preceding rules without the XSI marked extensions, any of the three  
 35112 comparison forms is reliable, given any input. (However, note that the strings are quoted in all  
 35113 cases.)

35114 Because the string comparison binary primaries, '= ' and '! =', have a higher precedence than  
 35115 any unary primary in the greater than 4 argument case, unexpected results can occur if  
 35116 arguments are not properly prepared. For example, in:

```
35117 test -d $1 -o -d $2
```

35118 If \$1 evaluates to a possible directory name of '= ', the first three arguments are considered a  
 35119 string comparison, which shall cause a syntax error when the second -d is encountered. One of  
 35120 the following forms prevents this; the second is preferred:

```
35121 test \(-d "$1" \) -o \(-d "$2" \)
35122 test -d "$1" || test -d "$2"
```

35123 Also in the greater than 4 argument case:

```
35124 test "$1" = "bat" -a "$2" = "ball"
```

35125 Syntax errors occur if \$1 evaluates to '( ' or '! '. One of the following forms prevents this; the  
 35126 third is preferred:

```
35127 test "X$1" = "Xbat" -a "X$2" = "Xball"
35128 test "$1" = "bat" && test "$2" = "ball"
35129 test "X$1" = "Xbat" && test "X$2" = "Xball"
```

### 35130 EXAMPLES

35131 1. Exit if there are not two or three arguments (two variations):

```
35132 if [$# -ne 2 -a $# -ne 3]; then exit 1; fi
35133 if [$# -lt 2 -o $# -gt 3]; then exit 1; fi
```

35134 2. Perform a *mkdir* if a directory does not exist:

```
35135 test ! -d tempdir && mkdir tempdir
```

35136 3. Wait for a file to become non-readable:

```
35137 while test -r thefile
35138 do
```

```

35139 sleep 30
35140 done
35141 echo '"thefile" is no longer readable'
35142
35143 4. Perform a command if the argument is one of three strings (two variations):
35144
35145 if ["$1" = "pear"] || ["$1" = "grape"] || ["$1" = "apple"]
35146 then
35147 command
35148 fi
35149
35150 case "$1" in
35151 pear|grape|apple) command ;;
35152 esac

```

#### 35150 RATIONALE

35151 The KornShell-derived conditional command (double bracket [[]]) was removed from the shell  
 35152 command language description in an early proposal. Objections were raised that the real  
 35153 problem is misuse of the *test* command (D), and putting it into the shell is the wrong way to fix  
 35154 the problem. Instead, proper documentation and a new shell reserved word (!) are sufficient.

35155 Tests that require multiple *test* operations can be done at the shell level using individual  
 35156 invocations of the *test* command and shell logicals, rather than using the error-prone *-o* flag of  
 35157 *test*.

35158 XSI-conformant systems support more than four arguments.

35159 XSI-conformant systems support the combining of primaries with the following constructs:

35160 *expression1 -a expression2*

35161 True if both *expression1* and *expression2* are true.

35162 *expression1 -o expression2*

35163 True if at least one of *expression1* and *expression2* are true.

35164 (*expression*)

35165 True if *expression* is true.

35166 In evaluating these more complex combined expressions, the following precedence rules are  
 35167 used:

- 35168 • The unary primaries have higher precedence than the algebraic binary primaries.
- 35169 • The unary primaries have lower precedence than the string binary primaries.
- 35170 • The unary and binary primaries have higher precedence than the unary *string* primary.
- 35171 • The ! operator has higher precedence than the *-a* operator, and the *-a* operator has higher  
 35172 precedence than the *-o* operator.
- 35173 • The *-a* and *-o* operators are left associative.
- 35174 • The parentheses can be used to alter the normal precedence and associativity.

35175 The BSD and System V versions of *-f* are not the same. The BSD definition was:

35176 *-f file* True if *file* exists and is not a directory.

35177 The SVID version (true if the file exists and is a regular file) was chosen for this volume of  
 35178 IEEE Std. 1003.1-200x because its use is consistent with the *-b*, *-c*, *-d*, and *-p* operands (*file*  
 35179 exists and is a specific file type).

35180 The `-e` primary, possessing similar functionality to that provided by the C shell, was added  
 35181 because it provides the only way for a shell script to find out if a file exists without trying to  
 35182 open the file. Since implementations are allowed to add additional file types, a portable script  
 35183 cannot use:

```
35184 test -b foo -o -c foo -o -d foo -o -f foo -o -p foo
```

35185 to find out if `foo` is an existing file.) On historical BSD systems, the existence of a file could be  
 35186 determined by:

```
35187 test -f foo -o -d foo
```

35188 but there was no easy way to determine that an existing file was a regular file. An early proposal  
 35189 used the KornShell `-a` primary (with the same meaning), but this was changed to `-e` because  
 35190 there were concerns about the high probability of humans confusing the `-a` primary with the `-a`  
 35191 binary operator.

35192 The following option was not included because it was undocumented in most implementations,  
 35193 has been removed from some implementations (including System V), and the functionality is  
 35194 provided by the shell (see Section 2.6.2 (on page 2245).

35195 `-l string` The length of the string *string*.

35196 The `-b`, `-c`, `-g`, `-p`, `-u`, and `-x` operands are derived from the SVID; historical BSD does not  
 35197 provide them. The `-k` operand is derived from System V; historical BSD does not provide it.

35198 On historical BSD systems, `test -w directory` always returned false because `test` tried to open the  
 35199 directory for writing, which always fails.

35200 Some additional primaries newly invented or from the KornShell appeared in an early proposal  
 35201 as part of the conditional command (`[[ ]]`): `s1 > s2`, `s1 < s2`, `str = pattern`, `str != pattern`, `f1 -nt f2`, `f1`  
 35202 `-ot f2`, and `f1 -ef f2`. They were not carried forward into the `test` utility when the conditional  
 35203 command was removed from the shell because they have not been included in the `test` utility  
 35204 built into historical implementations of the `sh` utility.

35205 The `-t file_descriptor` primary is shown with a mandatory argument because the grammar is  
 35206 ambiguous if it can be omitted. Historical implementations have allowed it to be omitted,  
 35207 providing a default of 1.

#### 35208 FUTURE DIRECTIONS

35209 None.

#### 35210 SEE ALSO

35211 *find*

#### 35212 CHANGE HISTORY

35213 First released in Issue 2.

#### 35214 Issue 4

35215 Aligned with the ISO/IEC 9945-2:1993 standard.

#### 35216 Issue 5

35217 FUTURE DIRECTIONS section added.

#### 35218 Issue 6

35219 The `-h` operand is added for symbolic links, and access permission requirements are clarified for  
 35220 the `-r`, `-w`, and `-x` operands to align with the IEEE P1003.2b draft standard.

35221 The normative text is reworded to avoid use of the term “must” for application requirements.



35222 **NAME**

35223           time — time a simple command

35224 **SYNOPSIS**35225 UP       time [-p] *utility* [*argument...*]

35226

35227 **DESCRIPTION**

35228       The *time* utility shall invoke the utility named by the *utility* operand with arguments supplied as  
 35229       the *argument* operands and write a message to standard error that lists timing statistics for the  
 35230       utility. The message shall include the following information:

- 35231           • The elapsed (real) time between invocation of *utility* and its termination.
- 35232           • The User CPU time, equivalent to the sum of the *tms\_utime* and *tms\_cutime* fields returned by  
 35233           the *times()* function defined in the System Interfaces volume of IEEE Std. 1003.1-200x for the  
 35234           process in which *utility* is executed.
- 35235           • The System CPU time, equivalent to the sum of the *tms\_stime* and *tms\_cstime* fields returned  
 35236           by the *times()* function for the process in which *utility* is executed.

35237       The precision of the timing shall be no less than the granularity defined for the size of the clock  
 35238       tick unit on the system, but the results shall be reported in terms of standard time units (for  
 35239       example, 0.02 seconds, 00:00:00.02, 1m33.75s, 365.21 seconds), not numbers of clock ticks.

35240       When *time* is used as part of a pipeline, the times reported are unspecified, except when it is the  
 35241       sole command within a grouping command (see Section 2.9.4.1 (on page 2261)) in that pipeline.  
 35242       For example, the commands on the left are unspecified; those on the right report on utilities **a**  
 35243       and **c**, respectively:

```
35244 time a | b | c { time a } | b | c
35245 a | b | time c a | b | (time c)
```

35246 **OPTIONS**

35247       The *time* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 35248       12.2, Utility Syntax Guidelines.

35249       The following option shall be supported:

- 35250       **-p**           Write the timing output to standard error in the format shown in the **STDERR**  
 35251       section.

35252 **OPERANDS**

35253       The following operands shall be supported:

- 35254       *utility*       The name of a utility that is to be invoked. If the *utility* operand names any of the  
 35255       special built-in utilities in Section 2.15 (on page 2276), the results are undefined.
- 35256       *argument*     Any string to be supplied as an argument when invoking the utility named by the  
 35257       *utility* operand.

35258 **STDIN**

35259       Not used.

35260 **INPUT FILES**

35261       None.

35262 **ENVIRONMENT VARIABLES**

35263 The following environment variables shall affect the execution of *time*:

35264 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 35265 If *LANG* is unset or null, the corresponding value from the implementation-  
 35266 defined default locale shall be used. If any of the internationalization variables  
 35267 contains an invalid setting, the utility shall behave as if none of the variables had  
 35268 been defined.

35269 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 35270 internationalization variables.

35271 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 35272 characters (for example, single-byte as opposed to multi-byte characters in  
 35273 arguments).

35274 *LC\_MESSAGES*  
 35275 Determine the locale that should be used to affect the format and contents of  
 35276 diagnostic and informative messages written to standard error.

35277 *LC\_NUMERIC*  
 35278 Determine the locale for numeric formatting.

35279 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

35280 *PATH* Determine the search path that shall be used to locate the utility to be invoked; see  
 35281 the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 8, Environment  
 35282 Variables.

35283 **ASYNCHRONOUS EVENTS**

35284 Default.

35285 **STDOUT**

35286 Not used.

35287 **STDERR**

35288 The standard error shall be used to write the timing statistics. If *-p* is specified, the following  
 35289 format shall be used in the POSIX locale:

35290 "real %f\nuser %f\nsys %f\n", <real seconds>, <user seconds>,  
 35291 <system seconds>

35292 where each floating-point number shall be expressed in seconds. The precision used may be less  
 35293 than the default six digits of %f, but shall be sufficiently precise to accommodate the size of the  
 35294 clock tick on the system (for example, if there were 60 clock ticks per second, at least two digits  
 35295 shall follow the radix character). The number of digits following the radix character shall be no  
 35296 less than one, even if this always results in a trailing zero. The implementation may append  
 35297 white space and additional information following the format shown here.

35298 **OUTPUT FILES**

35299 None.

35300 **EXTENDED DESCRIPTION**

35301 None.

35302 **EXIT STATUS**

35303 If the *utility* utility is invoked, the exit status of *time* shall be the exit status of *utility*; otherwise,  
 35304 the *time* utility shall exit with one of the following values:

- 35305 1-125 An error occurred in the *time* utility.
- 35306 126 The utility specified by *utility* was found but could not be invoked.
- 35307 127 The utility specified by *utility* could not be found.

### 35308 CONSEQUENCES OF ERRORS

35309 Default.

### 35310 APPLICATION USAGE

35311 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if  
 35312 an error occurs so that applications can distinguish “failure to find a utility” from “invoked  
 35313 utility exited with an error indication”. The value 127 was chosen because it is not commonly  
 35314 used for other meanings; most utilities use small values for “normal error conditions” and the  
 35315 values above 128 can be confused with termination due to receipt of a signal. The value 126 was  
 35316 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some  
 35317 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
 35318 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
 35319 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
 35320 any other reason.

### 35321 EXAMPLES

35322 It is frequently desirable to apply *time* to pipelines or lists of commands. This can be done by  
 35323 placing pipelines and command lists in a single file; this file can then be invoked as a utility, and  
 35324 the *time* applies to everything in the file.

35325 Alternatively, the following command can be used to apply *time* to a complex command:

```
35326 time sh -c 'complex-command-line'
```

### 35327 RATIONALE

35328 The *time* utility when originally proposed for this volume of IEEE Std. 1003.1-200x, was rejected  
 35329 because it was not useful for portable applications:

- 35330 • The underlying CPU definitions from the System Interfaces volume of IEEE Std. 1003.1-200x  
 35331 are vague, so the numeric output could not be compared accurately between systems or even  
 35332 between invocations.
- 35333 • The creation of portable benchmark programs was outside the scope this volume of  
 35334 IEEE Std. 1003.1-200x.

35335 However, *time* does fit in the scope of user portability. Human judgement can be applied to the  
 35336 analysis of the output, and it could be very useful in hands-on debugging of applications or in  
 35337 providing subjective measures of system performance. Hence it has been included in this  
 35338 volume of IEEE Std. 1003.1-200x.

35339 The default output format has been left unspecified because historical implementations differ  
 35340 greatly in their style of depicting this numeric output. The **-p** option was invented to provide  
 35341 scripts a common means of obtaining this information.

35342 In the KornShell, *time* is a shell reserved word that can be used to time an entire pipeline, rather  
 35343 than just a simple command. The POSIX definition has been worded to allow this  
 35344 implementation. Consideration was given to invalidating this approach because of the historical  
 35345 model from the C shell and System V shell. However, since the System V *time* utility historically  
 35346 has not produced accurate results in pipeline timing (because the constituent processes are not  
 35347 all owned by the same parent process, as allowed by POSIX), it did not seem worthwhile to  
 35348 break historical KornShell usage.

35349 The term *utility* is used, rather than *command*, to highlight the fact that shell compound  
35350 commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility*  
35351 includes user application programs and shell scripts, not just the standard utilities.

35352 **FUTURE DIRECTIONS**

35353 None.

35354 **SEE ALSO**

35355 *sh*, the System Interfaces volume of IEEE Std. 1003.1-200x, *times()*

35356 **CHANGE HISTORY**

35357 First released in Issue 2.

35358 **Issue 4**

35359 Aligned with the ISO/IEC 9945-2:1993 standard.

35360 **Issue 6**

35361 This utility is now marked as part of the User Portability Utilities option.

35362 **NAME**

35363 touch — change file access and modification times

35364 **SYNOPSIS**35365 touch [-acm][ -r *ref\_file* | -t *time*] *file*...35366 **DESCRIPTION**

35367 The *touch* utility shall change the modification times, access times, or both of files. The  
 35368 modification time shall be equivalent to the value of the *st\_mtime* member of the **stat** structure  
 35369 for a file, as described in the System Interfaces volume of IEEE Std. 1003.1-200x; the access time  
 35370 shall be equivalent to the value of *st\_atime*.

35371 The time used can be specified by the **-t** *time* option-argument, the corresponding time fields of  
 35372 the file referenced by the **-r** *ref\_file* option-argument, or the *date\_time* operand, as specified in the  
 35373 following sections. If none of these are specified, *touch* shall use the current time (the value  
 35374 returned by the equivalent of the *time()* function defined in the System Interfaces volume of  
 35375 IEEE Std. 1003.1-200x).

35376 For each *file* operand, *touch* shall perform actions equivalent to the following functions defined  
 35377 in the System Interfaces volume of IEEE Std. 1003.1-200x:

- 35378 1. If *file* does not exist, a *creat()* function call is made with the *file* operand used as the *path*  
 35379 argument and the value of the bitwise-inclusive OR of S\_IRUSR, S\_IWUSR, S\_IRGRP,  
 35380 S\_IWGRP, S\_IROTH, and S\_IWOTH used as the *mode* argument.
- 35381 2. The *utime()* function is called with the following arguments:
  - 35382 a. The *file* operand is used as the *path* argument.
  - 35383 b. The **utimbuf** structure members *actime* and *modtime* are determined as described in  
 35384 the OPTIONS section.

35385 **OPTIONS**

35386 The *touch* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 35387 12.2, Utility Syntax Guidelines.

35388 The following options shall be supported:

- |                |                           |                                                                                                                             |
|----------------|---------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| 35389<br>35390 | <b>-a</b>                 | Change the access time of <i>file</i> . Do not change the modification time unless <b>-m</b> is also specified.             |
| 35391<br>35392 | <b>-c</b>                 | Do not create a specified <i>file</i> if it does not exist. Do not write any diagnostic messages concerning this condition. |
| 35393<br>35394 | <b>-m</b>                 | Change the modification time of <i>file</i> . Do not change the access time unless <b>-a</b> is also specified.             |
| 35395<br>35396 | <b>-r</b> <i>ref_file</i> | Use the corresponding time of the file named by the path name <i>ref_file</i> instead of the current time.                  |
| 35397<br>35398 | <b>-t</b> <i>time</i>     | Use the specified <i>time</i> instead of the current time. The option-argument shall be a decimal number of the form:       |
| 35399          |                           | [ [ <i>CC</i> ] <i>YY</i> ] <i>MMDDhhmm</i> [ . <i>SS</i> ]                                                                 |
| 35400          |                           | where each two digits represents the following:                                                                             |
| 35401          | <i>MM</i>                 | The month of the year [01-12].                                                                                              |
| 35402          | <i>DD</i>                 | The day of the month [01-31].                                                                                               |

35403            *hh*        The hour of the day [00-23].  
 35404            *mm*        The minute of the hour [00-59].  
 35405            *CC*        The first two digits of the year (the century).  
 35406            *YY*        The second two digits of the year.  
 35407            *SS*        The second of the minute [00-61].  
 35408            Both *CC* and *YY* shall be optional. If neither is given, the current year shall be  
 35409            assumed. If *YY* is specified, but *CC* is not, *CC* shall be derived as follows:

35410  
 35411  
 35412

| If <i>YY</i> is: | <i>CC</i> becomes: |
|------------------|--------------------|
| 69-99            | 19                 |
| 00-68            | 20                 |

35413            The resulting time shall be affected by the value of the *TZ* environment variable. If  
 35414            the resulting time value precedes the Epoch, *touch* shall exit immediately with an  
 35415            error status. The range of valid times past the Epoch is implementation-defined,  
 35416            but it shall extend to at least the time 0 hours, 0 minutes, 0 seconds, January 1,  
 35417            2038, Coordinated Universal Time. Some systems may not be able to represent  
 35418            dates beyond the January 18, 2038, because they use **signed int** as a time holder.

35419            The range for *SS* is (00-61) rather than (00-59) because of leap seconds. If *SS* is 60 or  
 35420            61, and the resulting time, as affected by the *TZ* environment variable, does not  
 35421            refer to a leap second, the resulting time shall be one or two seconds after a time  
 35422            where *SS* is 59. If *SS* is not given a value, it is assumed to be zero.

35423            If neither the **-a** nor **-m** options were specified, *touch* shall behave as if both the **-a** and **-m**  
 35424            options were specified.

#### 35425 OPERANDS

35426            The following operands shall be supported:

35427            *file*        A path name of a file whose times shall be modified.

#### 35428 STDIN

35429            Not used.

#### 35430 INPUT FILES

35431            None.

#### 35432 ENVIRONMENT VARIABLES

35433            The following environment variables shall affect the execution of *touch*:

35434            *LANG*        Provide a default value for the internationalization variables that are unset or null.  
 35435            If *LANG* is unset or null, the corresponding value from the implementation-  
 35436            defined default locale shall be used. If any of the internationalization variables  
 35437            contains an invalid setting, the utility shall behave as if none of the variables had  
 35438            been defined.

35439            *LC\_ALL*      If set to a non-empty string value, override the values of all the other  
 35440            internationalization variables.

35441            *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 35442            characters (for example, single-byte as opposed to multi-byte characters in  
 35443            arguments).

35444            *LC\_MESSAGES*

35445            Determine the locale that should be used to affect the format and contents of

- 35446 diagnostic messages written to standard error.
- 35447 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 35448 **TZ** Determine the timezone to be used for interpreting the *time* option-argument.
- 35449 **ASYNCHRONOUS EVENTS**
- 35450 Default.
- 35451 **STDOUT**
- 35452 Not used.
- 35453 **STDERR**
- 35454 Used only for diagnostic messages.
- 35455 **OUTPUT FILES**
- 35456 None.
- 35457 **EXTENDED DESCRIPTION**
- 35458 None.
- 35459 **EXIT STATUS**
- 35460 The following exit values shall be returned:
- 35461 0 The utility executed successfully and all requested changes were made.
- 35462 >0 An error occurred.
- 35463 **CONSEQUENCES OF ERRORS**
- 35464 Default.
- 35465 **APPLICATION USAGE**
- 35466 The interpretation of time is taken to be *seconds since the Epoch* (see the Base Definitions volume of IEEE Std. 1003.1-200x, Section 4.12, Seconds Since the Epoch). It should be noted that implementations conforming to the System Interfaces volume of IEEE Std. 1003.1-200x do not take leap seconds into account when computing seconds since the Epoch. When *SS=60* is used, the resulting time always refers to 1 plus *seconds since the Epoch* for a time when *SS=59*.
- 35471 Although the *-t time* option-argument specifies values in 1969, the access time and modification time fields are defined in terms of seconds since the Epoch (midnight on 1 January 1970 UTC). Therefore, depending on the value of *TZ* when *touch* is run, there is never more than a few valid hours in 1969 and there need not be any valid times in 1969.
- 35475 One ambiguous situation occurs if *-t time* is not specified, *-r ref\_file* is not specified, and the first operand is an eight or ten-digit decimal number. A portable script can avoid this problem by using:
- 35476 `touch -- file`
- 35477
- 35478 or:
- 35479 `touch ./file`
- 35480
- 35481 in this case.
- 35482 **EXAMPLES**
- 35483 None.
- 35484 **RATIONALE**
- 35485 The functionality of *touch* is described almost entirely through references to functions in the System Interfaces volume of IEEE Std. 1003.1-200x. In this way, there is no duplication of effort required for describing such side effects as the relationship of user IDs to the user database,

- 35488 permissions, and so on.
- 35489 There are some significant differences between the *touch* utility in this volume of  
35490 IEEE Std. 1003.1-200x and those in System V and BSD systems. They are upward-compatible for  
35491 historical applications from both implementations:
- 35492 1. In System V, an ambiguity exists when a path name that is a decimal number leads the  
35493 operands; it is treated as a time value. In BSD, no *time* value is allowed; files may only be  
35494 *touched* to the current time. The `-t time` construct solves these problems for future portable  
35495 applications (note that the `-t` option is not historical practice).
  - 35496 2. The inclusion of the century digits, *CC*, is also new. Note that a ten-digit *time* value is  
35497 treated as if *YY*, and not *CC*, were specified. The caveat about the range of dates following  
35498 the Epoch was included as recognition that some implementations are not able to  
35499 represent dates beyond 18 January 2038 because they use **signed int** as a time holder.
- 35500 The `-r` option was added because several comments requested this capability. This option was  
35501 named `-f` in an early proposal, but was changed because the `-f` option is used in the BSD version  
35502 of *touch* with a different meaning.
- 35503 At least one historical implementation of *touch* incremented the exit code if `-c` was specified and  
35504 the file did not exist. This volume of IEEE Std. 1003.1-200x requires exit status zero if no errors  
35505 occur.
- 35506 **FUTURE DIRECTIONS**
- 35507 Applications should use the `-r` or `-t` options.
- 35508 **SEE ALSO**
- 35509 *date*, the System Interfaces volume of IEEE Std. 1003.1-200x, *creat()*, *time()*, `<sys/stat.h>`
- 35510 **CHANGE HISTORY**
- 35511 First released in Issue 2.
- 35512 **Issue 4**
- 35513 Aligned with the ISO/IEC 9945-2:1993 standard.
- 35514 **Issue 6**
- 35515 The obsolescent *date\_time* operand is removed.
- 35516 The Open Group corrigenda item U027/1 has been applied. This extends the range of valid time  
35517 past the Epoch to at least the time 0 hours, 0 minutes, 0 seconds, January 1, 2038, Coordinated  
35518 Universal Time. This is a new requirement on POSIX implementations.
- 35519 **Notes to Reviewers**
- 35520 *This section with side shading will not appear in the final copy. - Ed.*
- 35521 Should leap seconds be 00-61? c9x infers that it is only 00-60, and astronomers confirm that  
35522 double leap seconds do not occur.



35523 **NAME**

35524 tput — change terminal characteristics

35525 **SYNOPSIS**35526 UP tput [-T *type*] *operand...*

35527

35528 **DESCRIPTION**

35529 The *tput* utility shall display terminal-dependent information. The manner in which this  
 35530 information is retrieved is unspecified. The information displayed shall clear the terminal screen,  
 35531 initialize the user's terminal, or reset the user's terminal, depending on the operand given. The  
 35532 exact consequences of displaying this information are unspecified.

35533 **OPTIONS**

35534 The *tput* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 35535 12.2, Utility Syntax Guidelines.

35536 The following option shall be supported:

35537 **-T *type*** Indicate the type of terminal. If this option is not supplied and the *TERM* variable  
 35538 is unset or null, an unspecified default terminal type shall be used. The setting of  
 35539 *type* shall take precedence over the value in *TERM*.

35540 **OPERANDS**

35541 The following strings shall be supported as operands by the implementation in the POSIX locale:

35542 **clear** Display the clear-screen sequence.

35543 **init** Display the sequence that initializes the user's terminal in an implementation-  
 35544 defined manner.

35545 **reset** Display the sequence that resets the user's terminal in an implementation-defined  
 35546 manner.

35547 If a terminal does not support any of the operations described by these operands, this shall not  
 35548 be considered an error condition.

35549 **STDIN**

35550 Not used.

35551 **INPUT FILES**

35552 None.

35553 **ENVIRONMENT VARIABLES**

35554 The following environment variables shall affect the execution of *tput*:

35555 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 35556 If *LANG* is unset or null, the corresponding value from the implementation-  
 35557 defined default locale shall be used. If any of the internationalization variables  
 35558 contains an invalid setting, the utility shall behave as if none of the variables had  
 35559 been defined.

35560 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 35561 internationalization variables.

35562 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 35563 characters (for example, single-byte as opposed to multi-byte characters in  
 35564 arguments).

35565 **LC\_MESSAGES**

35566 Determine the locale that should be used to affect the format and contents of

35567 diagnostic messages written to standard error.

35568 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

35569 **TERM** Determine the terminal type. If this variable is unset or null, and if the **-T** option is  
35570 not specified, an unspecified default terminal type shall be used.

35571 **ASYNCHRONOUS EVENTS**

35572 Default.

35573 **STDOUT**

35574 If standard output is a terminal device, it may be used for writing the appropriate sequence to  
35575 clear the screen or reset or initialize the terminal. If standard output is not a terminal device,  
35576 undefined results occur.

35577 **STDERR**

35578 Used only for diagnostic messages.

35579 **OUTPUT FILES**

35580 None.

35581 **EXTENDED DESCRIPTION**

35582 None.

35583 **EXIT STATUS**

35584 The following exit values shall be returned:

35585 0 The requested string was written successfully.

35586 1 Unspecified.

35587 2 Usage error.

35588 3 No information is available about the specified terminal type.

35589 4 The specified operand is invalid.

35590 >4 An error occurred.

35591 **CONSEQUENCES OF ERRORS**

35592 If one of the operands is not available for the terminal, *tput* continues processing the remaining  
35593 operands.

35594 **APPLICATION USAGE**

35595 The difference between resetting and initializing a terminal is left unspecified, as they vary  
35596 greatly based on hardware types. In general, resetting is a more severe action.

35597 Some terminals use control characters to perform the stated functions, and on such terminals it  
35598 might make sense to use *tput* to store the initialization strings in a file or environment variable  
35599 for later use. However, because other terminals might rely on system calls to do this work, the  
35600 standard output cannot be used in a portable manner, such as the following non-portable  
35601 constructs:

35602 ClearVar='tput clear'  
35603 tput reset | mailx -s "Wake Up" ddg

35604 **EXAMPLES**

35605 1. Initialize the terminal according to the type of terminal in the environmental variable  
35606 *TERM*. This command can be included in a **.profile** file.

35607 tput init

35608 2. Reset a 450 terminal.

35609 `tput -T 450 reset`

35610 **RATIONALE**

35611 The list of operands was reduced to a minimum for the following reasons:

35612 • The only features chosen were those that were likely to be used by human users interacting  
35613 with a terminal.

35614 • Specifying the full *terminfo* set was not considered desirable, but the standard developers did  
35615 not want to select among operands.

35616 • This volume of IEEE Std. 1003.1-200x does not attempt to provide applications with  
35617 sophisticated terminal handling capabilities, as that falls outside of its assigned scope and  
35618 intersects with the responsibilities of other standards bodies.

35619 The difference between resetting and initializing a terminal is left unspecified as this varies  
35620 greatly based on hardware types. In general, resetting is a more severe action.

35621 The exit status of 1 is historically reserved for finding out if a Boolean operand is not set.  
35622 Although the operands were reduced to a minimum, the exit status of 1 should still be reserved  
35623 for the Boolean operands, for those sites that wish to support them.

35624 **FUTURE DIRECTIONS**

35625 None.

35626 **SEE ALSO**

35627 *stty*, *tabs*

35628 **CHANGE HISTORY**

35629 First released in Issue 4.

35630 **Issue 6**

35631 This utility is now marked as part of the User Portability Utilities option.

35632 **NAME**

35633 tr — translate characters

35634 **SYNOPSIS**35635 tr [-c | -C][-s] *string1 string2*35636 tr -s [-c | -C] *string1*35637 tr -d [-c | -C] *string1*35638 tr -ds [-c | -C] *string1 string2*35639 **DESCRIPTION**

35640 The *tr* utility shall copy the standard input to the standard output with substitution or deletion  
 35641 of selected characters. The options specified and the *string1* and *string2* operands shall control  
 35642 translations that occur while copying characters and single-character collating elements.

35643 **OPTIONS**

35644 The *tr* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,  
 35645 Utility Syntax Guidelines.

35646 The following options shall be supported:

35647 **-c** Complement the set of values specified by *string1*. See the EXTENDED  
 35648 DESCRIPTION section.

35649 **-C** Complement the set of characters specified by *string1*. See the EXTENDED  
 35650 DESCRIPTION section.

35651 **-d** Delete all occurrences of input characters that are specified by *string1*.

35652 **-s** Replace instances of repeated characters with a single character, as described in the  
 35653 EXTENDED DESCRIPTION section.

35654 **OPERANDS**

35655 The following operands shall be supported:

35656 *string1, string2*

35657 Translation control strings. Each string shall represent a set of characters to be  
 35658 converted into an array of characters used for the translation. For a detailed  
 35659 description of how the strings are interpreted, see the EXTENDED DESCRIPTION  
 35660 section.

35661 **STDIN**

35662 The standard input can be any type of file.

35663 **INPUT FILES**

35664 None.

35665 **ENVIRONMENT VARIABLES**

35666 The following environment variables shall affect the execution of *tr*:

35667 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 35668 If *LANG* is unset or null, the corresponding value from the implementation-  
 35669 defined default locale shall be used. If any of the internationalization variables  
 35670 contains an invalid setting, the utility shall behave as if none of the variables had  
 35671 been defined.

35672 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 35673 internationalization variables.

|       |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 35674 | <i>LC_COLLATE</i>           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35675 |                             | Determine the locale for the behavior of range expressions and equivalence classes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 35676 | <i>LC_CTYPE</i>             | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments) and the behavior of character classes.                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35677 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35678 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35679 | <i>LC_MESSAGES</i>          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35680 |                             | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 35681 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35682 | XSI <i>NLSPATH</i>          | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 35683 | <b>ASYNCHRONOUS EVENTS</b>  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35684 |                             | Default.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 35685 | <b>STDOUT</b>               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35686 |                             | The <i>tr</i> output shall be identical to the input, with the exception of the specified transformations.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35687 | <b>STDERR</b>               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35688 |                             | Used only for diagnostic messages.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35689 | <b>OUTPUT FILES</b>         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35690 |                             | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 35691 | <b>EXTENDED DESCRIPTION</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35692 |                             | The operands <i>string1</i> and <i>string2</i> (if specified) define two arrays of characters. The constructs in the following list can be used to specify characters or single-character collating elements. If any of the constructs result in multi-character collating elements, <i>tr</i> shall exclude, without a diagnostic, those multi-character elements from the resulting array.                                                                                                                                                                                                                                                            |
| 35693 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35694 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35695 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35696 | <i>character</i>            | Any character not described by one of the conventions below represents itself.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 35697 | <i>\octal</i>               | Octal sequences can be used to represent characters with specific coded values. An octal sequence shall consist of a backslash followed by the longest sequence of one, two, or three-octal-digit characters (01234567). The sequence shall cause the value whose encoding is represented by the one, two, or three-digit octal integer to be placed into the array. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is implementation-defined. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading ' <i>\</i> ' for each byte. |
| 35698 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35699 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35700 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35701 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35702 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35703 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35704 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35705 | <i>\character</i>           | The backslash-escape sequences in the Base Definitions volume of IEEE Std. 1003.1-200x, Table 5-1, Escape Sequences and Associated Actions (' <i>\</i> <i>\</i> ', ' <i>\a</i> ', ' <i>\b</i> ', ' <i>\f</i> ', ' <i>\n</i> ', ' <i>\r</i> ', ' <i>\t</i> ', ' <i>\v</i> ') shall be supported. The results of using any other character, other than an octal digit, following the backslash are unspecified.                                                                                                                                                                                                                                           |
| 35706 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35707 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35708 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35709 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35710 | <i>c-c</i>                  | Represents the range of collating elements between the range endpoints (as long as neither endpoint is an octal sequence of the form <i>\octal</i> ), inclusive, as defined by the current setting of the <i>LC_COLLATE</i> locale category. The application shall ensure that the starting endpoint precedes the second endpoint in the current collation order. The characters or collating elements in the range shall be placed in the array in ascending collation sequence.                                                                                                                                                                       |
| 35711 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35712 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35713 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35714 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35715 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35716 |                             | If either or both of the range endpoints are octal sequences of the form <i>\octal</i> , this shall represent the range of specific coded values between the two range endpoints, inclusive.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 35717 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35718 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

|           |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 35719     | [:class:]               | Represents all characters belonging to the defined character class, as defined by the current setting of the <i>LC_CTYPE</i> locale category. The following character class names shall be accepted when specified in <i>string1</i> :                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 35720     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35721     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35722     |                         | <b>alnum</b> <b>blank</b> <b>digit</b> <b>lower</b> <b>punct</b> <b>upper</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 35723     |                         | <b>alpha</b> <b>cntrl</b> <b>graph</b> <b>print</b> <b>space</b> <b>xdigit</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 35724 XSI |                         | In addition, character class expressions of the form [: <i>name</i> :] shall be recognized in those locales where the <i>name</i> keyword has been given a <b>charclass</b> definition in the <i>LC_CTYPE</i> category.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35725     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35726     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35727     |                         | When both the <b>-d</b> and <b>-s</b> options are specified, any of the character class names shall be accepted in <i>string2</i> . Otherwise, only character class names <b>lower</b> or <b>upper</b> are valid in <i>string2</i> and then only if the corresponding character class ( <b>upper</b> and <b>lower</b> , respectively) is specified in the same relative position in <i>string1</i> . Such a specification shall be interpreted as a request for case conversion. When [: <i>lower</i> :] appears in <i>string1</i> and [: <i>upper</i> :] appears in <i>string2</i> , the arrays shall contain the characters from the <b>toupper</b> mapping in the <i>LC_CTYPE</i> category of the current locale. When [: <i>upper</i> :] appears in <i>string1</i> and [: <i>lower</i> :] appears in <i>string2</i> , the arrays shall contain the characters from the <b>tolower</b> mapping in the <i>LC_CTYPE</i> category of the current locale. The first character from each mapping pair shall be in the array for <i>string1</i> and the second character from each mapping pair shall be in the array for <i>string2</i> in the same relative position. |
| 35728     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35729     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35730     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35731     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35732     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35733     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35734     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35735     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35736     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35737     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35738     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35739     |                         | Except for case conversion, the characters specified by a character class expression shall be placed in the array in an unspecified order.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 35740     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35741     |                         | If the name specified for <i>class</i> does not define a valid character class in the current locale, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 35742     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35743     | [= <i>equiv</i> =]      | Represents all characters or collating elements belonging to the same equivalence class as <i>equiv</i> , as defined by the current setting of the <i>LC_COLLATE</i> locale category. An equivalence class expression shall be allowed only in <i>string1</i> , or in <i>string2</i> when it is being used by the combined <b>-d</b> and <b>-s</b> options. The characters belonging to the equivalence class shall be placed in the array in an unspecified order.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 35744     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35745     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35746     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35747     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35748     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35749     | [ <i>x</i> * <i>n</i> ] | Represents <i>n</i> repeated occurrences of the character <i>x</i> . Because this expression is used to map multiple characters to one, it is only valid when it occurs in <i>string2</i> . If <i>n</i> is omitted or is zero, it shall be interpreted as large enough to extend the <i>string2</i> -based sequence to the length of the <i>string1</i> -based sequence. If <i>n</i> has a leading zero, it shall be interpreted as an octal value. Otherwise, it shall be interpreted as a decimal value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 35750     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35751     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35752     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35753     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35754     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35755     |                         | When the <b>-d</b> option is not specified:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 35756     |                         | • Each input character found in the array specified by <i>string1</i> shall be replaced by the character in the same relative position in the array specified by <i>string2</i> . When the array specified by <i>string2</i> is shorter than the one specified by <i>string1</i> , the results are unspecified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35757     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35758     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35759     |                         | • If the <b>-C</b> option is specified, the complements of the characters specified by <i>string1</i> (the set of all characters in the current character set, as defined by the current setting of <i>LC_CTYPE</i> , except for those actually specified in the <i>string1</i> operand) shall be placed in the array in ascending collation sequence, as defined by the current setting of <i>LC_COLLATE</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35760     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35761     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35762     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35763     |                         | • If the <b>-c</b> option is specified, the complement of the values specified by <i>string1</i> shall be placed in the array in ascending order by binary value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 35764     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

35765 • Because the order in which characters specified by character class expressions or equivalence  
 35766 class expressions is undefined, such expressions should only be used if the intent is to map  
 35767 several characters into one. An exception is case conversion, as described previously.

35768 When the `-d` option is specified:

- 35769 • Input characters found in the array specified by *string1* shall be deleted.
- 35770 • When the `-C` option is specified with `-d`, all characters except those specified by *string1* shall  
 35771 be deleted. The contents of *string2* are ignored, unless the `-s` option is also specified.
- 35772 • When the `-c` option is specified with `-d`, all values except those specified by *string1* shall be  
 35773 deleted. The contents of *string2* shall be ignored, unless the `-s` option is also specified.
- 35774 • The same string cannot be used for both the `-d` and the `-s` option; when both options are  
 35775 specified, both *string1* (used for deletion) and *string2* (used for squeezing) shall be required.

35776 When the `-s` option is specified, after any deletions or translations have taken place, repeated  
 35777 sequences of the same character shall be replaced by one occurrence of the same character, if the  
 35778 character is found in the array specified by the last operand. If the last operand contains a  
 35779 character class, such as the following example:

```
35780 tr -s '[:space:]'
```

35781 the last operand's array shall contain all of the characters in that character class. However, in a  
 35782 case conversion, as described previously, such as:

```
35783 tr -s '[:upper:]' '[:lower:]'
```

35784 the last operand's array shall contain only those characters defined as the second characters in  
 35785 each of the **toupper** or **tolower** character pairs, as appropriate.

35786 An empty string used for *string1* or *string2* produces undefined results.

#### 35787 EXIT STATUS

35788 The following exit values shall be returned:

35789 0 All input was processed successfully.

35790 >0 An error occurred.

#### 35791 CONSEQUENCES OF ERRORS

35792 Default.

#### 35793 APPLICATION USAGE

35794 If necessary, *string1* and *string2* can be quoted to avoid pattern matching by the shell.

35795 If an ordinary digit (representing itself) is to follow an octal sequence, the octal sequence must  
 35796 use the full three digits to avoid ambiguity.

35797 When *string2* is shorter than *string1*, a difference results between historical System V and BSD  
 35798 systems. A BSD system pads *string2* with the last character found in *string2*. Thus, it is possible  
 35799 to do the following:

```
35800 tr 0123456789 d
```

35801 which would translate all digits to the letter 'd'. Since this area is specifically unspecified in  
 35802 this volume of IEEE Std. 1003.1-200x, both the BSD and System V behaviors are allowed, but a  
 35803 portable application cannot rely on the BSD behavior. It would have to code the example in the  
 35804 following way:

```
35805 tr 0123456789 '[d*]'
```

35806 It should be noted that, despite similarities in appearance, the string operands used by *tr* are not  
35807 regular expressions.

35808 Unlike some historical implementations, this definition of the *tr* utility correctly processes NUL  
35809 characters in its input stream. NUL characters can be stripped by using:

```
35810 tr -d '\000'
```

### 35811 EXAMPLES

35812 1. The following example creates a list of all words in **file1** one per line in **file2**, where a word  
35813 is taken to be a maximal string of letters.

```
35814 tr -cs "[:alpha:]" "[\n*]" <file1 >file2
```

35815 2. The next example translates all lowercase characters in **file1** to uppercase and writes the  
35816 results to standard output.

```
35817 tr "[:lower:]" "[:upper:]" <file1
```

35818 3. This example uses an equivalence class to identify accented variants of the base character  
35819 'e' in **file1**, which are stripped of diacritical marks and written to **file2**.

```
35820 tr "[=e=]" e <file1 >file2
```

### 35821 RATIONALE

35822 In some early proposals, an explicit option **-n** was added to disable the historical behavior of  
35823 stripping NUL characters from the input. It was considered that automatically stripping NUL  
35824 characters from the input was not correct functionality. However, the removal of **-n** in a later  
35825 proposal does not remove the requirement that *tr* correctly process NUL characters in its input  
35826 stream. NUL characters can be stripped by using *tr -d '\000'*.

35827 Historical implementations of *tr* differ widely in syntax and behavior. For example, the BSD  
35828 version has not needed the bracket characters for the repetition sequence. The POSIX Shell and  
35829 Utilities *tr* syntax is based more closely on the System V and XPG3 model while attempting to  
35830 accommodate historical BSD implementations. In the case of the short *string2* padding, the  
35831 decision was to unspecified the behavior and preserve System V and XPG3 scripts, which might  
35832 find difficulty with the BSD method. The assumption was made that BSD users of *tr* have to  
35833 make accommodations to meet the POSIX Shell and Utilities syntax. Since it is possible to use  
35834 the repetition sequence to duplicate the desired behavior, whereas there is no simple way to  
35835 achieve the System V method, this was the correct, if not desirable, approach.

35836 The use of octal values to specify control characters, while having historical precedents, is not  
35837 portable. The introduction of escape sequences for control characters should provide the  
35838 necessary portability. It is recognized that this may cause some historical scripts to break.

35839 An early proposal included support for multi-character collating elements. It was pointed out  
35840 that, while *tr* does employ some syntactical elements from REs, the aim of *tr* is quite different;  
35841 ranges, for example, do not have a similar meaning (“any of the chars in the range matches”,  
35842 *versus* “translate each character in the range to the output counterpart”). As a result, the  
35843 previously included support for multi-character collating elements has been removed. What  
35844 remains are ranges in current collation order (to support, for example, accented characters),  
35845 character classes, and equivalence classes.

35846 In XPG3 the *[:class:]* and *[=equiv=]* conventions are shown with double brackets, as in RE syntax.  
35847 However, *tr* does not implement RE principles; it just borrows part of the syntax. Consequently,  
35848 *[:class:]* and *[=equiv=]* should be regarded as syntactical elements on a par with *[x\*n]*, which is  
35849 not an RE bracket expression.



- 35850 The standard developers will consider changes to *tr* that allow it to translate characters between  
35851 different character encodings, or they will consider providing a new utility to accomplish this.
- 35852 On historical System V systems, a range expression requires enclosing square-brackets, such as:  
35853 `tr '[a-z]' '[A-Z]'`
- 35854 However, BSD-based systems did not require the brackets, and this convention is used by POSIX  
35855 Shell and Utilities to avoid breaking large numbers of BSD scripts:
- 35856 `tr a-z A-Z`
- 35857 The preceding System V script will continue to work because the brackets, treated as regular  
35858 characters, are translated to themselves. However, any System V script that relied on *a-z*  
35859 representing the three characters '-', ',' and 'z' have to be rewritten as *az-*.
- 35860 A prior version of IEEE Std. 1003.1-200x had a *-c* option that behaved similarly to the *-C* option,  
35861 but did not supply functionality equivalent to the *-c* option specified in IEEE Std. 1003.1-200x.  
35862 This meant that historical practice of being able to specify `tr -d\200-\377` (which would delete  
35863 all bytes with the top bit set) would have no effect because, in the C locale, bytes with the values  
35864 octal 200 to octal 377 are not characters.
- 35865 The earlier version also said that octal sequences referred to collating elements and could be  
35866 placed adjacent to each other to specify multi-byte characters. However, it was noted that this  
35867 caused ambiguities because *tr* would not be able to tell whether adjacent octal sequences were  
35868 intending to specify multi-byte characters or multiple single byte characters.  
35869 IEEE Std. 1003.1-200x specifies that octal sequences always refer to single byte binary values.
- 35870 **FUTURE DIRECTIONS**
- 35871 None.
- 35872 **SEE ALSO**
- 35873 *sed*
- 35874 **CHANGE HISTORY**
- 35875 First released in Issue 2.
- 35876 **Issue 4**
- 35877 Aligned with the ISO/IEC 9945-2:1993 standard.
- 35878 **Issue 6**
- 35879 The *-C* operand is added, and the description of the *-c* operand is changed to align with the  
35880 IEEE P1003.2b draft standard.
- 35881 The normative text is reworded to avoid use of the term “must” for application requirements.

35882 **NAME**

35883 true — return true value

35884 **SYNOPSIS**

35885 true

35886 **DESCRIPTION**35887 The *true* utility shall return with exit code zero.35888 **OPTIONS**

35889 None.

35890 **OPERANDS**

35891 None.

35892 **STDIN**

35893 Not used.

35894 **INPUT FILES**

35895 None.

35896 **ENVIRONMENT VARIABLES**

35897 None.

35898 **ASYNCHRONOUS EVENTS**

35899 Default.

35900 **STDOUT**

35901 Not used.

35902 **STDERR**

35903 None.

35904 **OUTPUT FILES**

35905 None.

35906 **EXTENDED DESCRIPTION**

35907 None.

35908 **EXIT STATUS**

35909 Default.

35910 **CONSEQUENCES OF ERRORS**

35911 None.

35912 **APPLICATION USAGE**35913 This utility is typically used in shell scripts, as shown in the **EXAMPLES** section. The special  
35914 built-in utility `:` is sometimes more efficient than *true*.35915 **EXAMPLES**

35916 This command is executed forever:

35917 while true  
35918 do  
35919     command  
35920 done

35921 **RATIONALE**

35922           The *true* utility has been retained in this volume of IEEE Std. 1003.1-200x, even though the shell  
35923           special built-in : provides similar functionality, because *true* is widely used in historical scripts  
35924           and is less cryptic to novice script readers.

35925 **FUTURE DIRECTIONS**

35926           None.

35927 **SEE ALSO**

35928           *false*, Section 2.9 (on page 2256)

35929 **CHANGE HISTORY**

35930           First released in Issue 2.

35931 **Issue 4**

35932           Aligned with the ISO/IEC 9945-2:1993 standard.

35933 **NAME**

35934           tsort — topological sort

35935 **SYNOPSIS**35936 XSI        tsort [*file*]

35937

35938 **DESCRIPTION**35939           The *tsort* utility shall write to standard output a totally ordered list of items consistent with a  
35940           partial ordering of items contained in the input.35941           The application shall ensure that the input consists of pairs of items (non-empty strings)  
35942           separated by <blank>s. Pairs of different items indicate ordering. Pairs of identical items  
35943           indicate presence, but not ordering.35944 **OPTIONS**

35945           None.

35946 **OPERANDS**

35947           The following operand shall be supported:

35948           *file*           A path name of a text file to order. If no *file* operand is given, the standard input is  
35949           used.35950 **STDIN**35951           The standard input shall be a text file that is used if no *file* operand is given.35952 **INPUT FILES**35953           The input file named by the *file* operand is a text file.35954 **ENVIRONMENT VARIABLES**35955           The following environment variables shall affect the execution of *tsort*:35956           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
35957           If *LANG* is unset or null, the corresponding value from the implementation-  
35958           defined default locale shall be used. If any of the internationalization variables  
35959           contains an invalid setting, the utility shall behave as if none of the variables had  
35960           been defined.35961           *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
35962           internationalization variables.35963           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
35964           characters (for example, single-byte as opposed to multi-byte characters in  
35965           arguments and input files).35966           *LC\_MESSAGES*35967           Determine the locale that should be used to affect the format and contents of  
35968           diagnostic messages written to standard error.35969           *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.35970 **ASYNCHRONOUS EVENTS**

35971           Default.

35972 **STDOUT**35973           The standard output shall be a text file consisting of the order list produced from the partially  
35974           ordered input.

35975 **STDERR**

35976 Used only for diagnostic messages.

35977 **OUTPUT FILES**

35978 None.

35979 **EXTENDED DESCRIPTION**

35980 None.

35981 **EXIT STATUS**

35982 The following exit values shall be returned:

35983 0 Successful completion.

35984 >0 An error occurred.

35985 **CONSEQUENCES OF ERRORS**

35986 Default.

35987 **APPLICATION USAGE**

35988 The *LC\_COLLATE* variable need not affect the actions of *tsort*. The output ordering is not  
35989 lexicographic, but depends on the pairs of items given as input.

35990 **EXAMPLES**

35991 The command:

```
35992 tsort <<EOF
35993 a b c c d e
35994 g g
35995 f g e f
35996 h h
35997 EOF
```

35998 produces the output:

```
35999 a
36000 b
36001 c
36002 d
36003 e
36004 f
36005 g
36006 h
```

36007 **RATIONALE**

36008 None.

36009 **FUTURE DIRECTIONS**

36010 None.

36011 **SEE ALSO**

36012 None.

36013 **CHANGE HISTORY**

36014 First released in Issue 2.

36015 **Issue 4**

36016 Format reorganized.

36017 Internationalized environment variable support mandated.

36018 **Issue 6**  
36019

The normative text is reworded to avoid use of the term “must” for application requirements.

36020 **NAME**

36021 tty — return user's terminal name

36022 **SYNOPSIS**

36023 tty

36024 **DESCRIPTION**

36025 The *tty* utility shall write to the standard output the name of the terminal that is open as  
 36026 standard input. The name that is used shall be equivalent to the string that would be returned by  
 36027 the *ttyname()* function defined in the System Interfaces volume of IEEE Std. 1003.1-200x.

36028 **OPTIONS**

36029 The *tty* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 36030 12.2, Utility Syntax Guidelines.

36031 **OPERANDS**

36032 None.

36033 **STDIN**

36034 While no input is read from standard input, standard input shall be examined to determine  
 36035 whether or not it is a terminal, and, if so, to determine the name of the terminal.

36036 **INPUT FILES**

36037 None.

36038 **ENVIRONMENT VARIABLES**36039 The following environment variables shall affect the execution of *tty*:

36040 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 36041 If *LANG* is unset or null, the corresponding value from the implementation-  
 36042 defined default locale shall be used. If any of the internationalization variables  
 36043 contains an invalid setting, the utility shall behave as if none of the variables had  
 36044 been defined.

36045 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 36046 internationalization variables.

36047 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 36048 characters (for example, single-byte as opposed to multi-byte characters in  
 36049 arguments).

36050 *LC\_MESSAGES*

36051 Determine the locale that should be used to affect the format and contents of  
 36052 diagnostic messages written to standard error and informative messages written to  
 36053 standard output.

36054 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36055 **ASYNCHRONOUS EVENTS**

36056 Default.

36057 **STDOUT**

36058 If standard input is a terminal device, a path name of the terminal as specified by the *ttyname()*  
 36059 function defined in the System Interfaces volume of IEEE Std. 1003.1-200x shall be written in the  
 36060 following format:

36061 "%s\n", &lt;terminal name&gt;

36062 Otherwise, a message shall be written indicating that standard input is not connected to a  
 36063 terminal. In the POSIX locale, the *tty* utility shall use the format:

36064 "not a tty\n"

36065 **STDERR**

36066 Used only for diagnostic messages.

36067 **OUTPUT FILES**

36068 None.

36069 **EXTENDED DESCRIPTION**

36070 None.

36071 **EXIT STATUS**

36072 The following exit values shall be returned:

36073 0 Standard input is a terminal.

36074 1 Standard input is not a terminal.

36075 >1 An error occurred.

36076 **CONSEQUENCES OF ERRORS**

36077 Default.

36078 **APPLICATION USAGE**

36079 This utility checks the status of the file open as standard input against that of a system-defined  
36080 set of files. It is possible that no match can be found, or that the match found need not be the  
36081 same file as that which was opened for standard input (although they are the same device).

36082 The `-s` option is useful only if the exit code is wanted. It does not rely on the ability to form a  
36083 valid path name. Portable applications should use `test -t 0`.

36084 **EXAMPLES**

36085 None.

36086 **RATIONALE**

36087 None.

36088 **FUTURE DIRECTIONS**

36089 None.

36090 **SEE ALSO**

36091 The System Interfaces volume of IEEE Std. 1003.1-200x, `isatty()`, `ttyname()`

36092 **CHANGE HISTORY**

36093 First released in Issue 2.

36094 **Issue 4**

36095 Aligned with the ISO/IEC 9945-2:1993 standard.

36096 **Issue 5**

36097 The SYNOPSIS is changed to indicate two forms of the command, with the second form marked  
36098 as obsolete. This is a clarification and does not change the functionality published in previous  
36099 issues.



36100 **NAME**

36101            type — write a description of command type

36102 **SYNOPSIS**

36103 xSI        type name...

36104

36105 **DESCRIPTION**36106            The *type* utility shall indicate how each argument would be interpreted if used as a command  
36107            name.36108 **OPTIONS**

36109            None.

36110 **OPERANDS**

36111            The following operand shall be supported:

36112            *name*            A name to be interpreted.36113 **STDIN**

36114            Not used.

36115 **INPUT FILES**

36116            None.

36117 **ENVIRONMENT VARIABLES**36118            The following environment variables shall affect the execution of *type*:36119            *LANG*            Provide a default value for the internationalization variables that are unset or null.  
36120                                If *LANG* is unset or null, the corresponding value from the implementation-  
36121                                defined default locale shall be used. If any of the internationalization variables  
36122                                contains an invalid setting, the utility shall behave as if none of the variables had  
36123                                been defined.36124            *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
36125                                internationalization variables.36126            *LC\_CTYPE*    Determine the locale for the interpretation of sequences of bytes of text data as  
36127                                characters (for example, single-byte as opposed to multi-byte characters in  
36128                                arguments).36129            *LC\_MESSAGES*36130                                Determine the locale that should be used to affect the format and contents of  
36131                                diagnostic messages written to standard error.36132            *NLSPATH*     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.36133            *PATH*            Determine the location of *name*, as described in the Base Definitions volume of  
36134                                IEEE Std. 1003.1-200x, Chapter 8, Environment Variables.36135 **ASYNCHRONOUS EVENTS**

36136            Default.

36137 **STDOUT**36138            The standard output of *type* contains information about each operand in an unspecified format.  
36139            The information provided typically identifies the operand as a shell built-in, function, alias, or  
36140            keyword, and where applicable, may display the operand's path name.

36141 **STDERR**

36142 Used only for diagnostic messages.

36143 **OUTPUT FILES**

36144 None.

36145 **EXTENDED DESCRIPTION**

36146 None.

36147 **EXIT STATUS**

36148 The following exit values shall be returned:

36149 0 Successful completion.

36150 &gt;0 An error occurred.

36151 **CONSEQUENCES OF ERRORS**

36152 Default.

36153 **APPLICATION USAGE**

36154 Since *type* must be aware of the contents of the current shell execution environment (such as the  
36155 lists of commands, functions, and built-ins processed by *hash*), it is always provided as a shell  
36156 regular built-in. If it is called in a separate utility execution environment, such as one of the  
36157 following:

36158 `nohup type writer`36159 `find . -type f | xargs type`

36160 it might not produce accurate results.

36161 **EXAMPLES**

36162 None.

36163 **RATIONALE**

36164 None.

36165 **FUTURE DIRECTIONS**

36166 None.

36167 **SEE ALSO**36168 *command*36169 **CHANGE HISTORY**

36170 First released in Issue 2.

36171 **Issue 4**36172 Relocated from the *sh* description to reflect its status as a regular built-in utility.

36173 **NAME**36174 `ulimit` — set or report file size limit36175 **SYNOPSIS**36176 XSI `ulimit [-f][blocks]`

36177

36178 **DESCRIPTION**

36179 The *ulimit* utility shall set or report the file-size writing limit imposed on files written by the  
 36180 shell and its child processes (files of any size may be read). Only a process with appropriate  
 36181 privileges can increase the limit.

36182 **OPTIONS**

36183 The *ulimit* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 36184 12.2, Utility Syntax Guidelines.

36185 The following option shall be supported:

36186 `-f` Set (or report, if no *blocks* operand is present), the file size limit in blocks. The `-f`  
 36187 option shall also be the default case.

36188 **OPERANDS**

36189 The following operand shall be supported:

36190 *blocks* The number of 512-byte blocks to use as the new file size limit.

36191 **STDIN**

36192 Not used.

36193 **INPUT FILES**

36194 None.

36195 **ENVIRONMENT VARIABLES**

36196 The following environment variables shall affect the execution of *ulimit*:

36197 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 36198 If *LANG* is unset or null, the corresponding value from the implementation-  
 36199 defined default locale shall be used. If any of the internationalization variables  
 36200 contains an invalid setting, the utility shall behave as if none of the variables had  
 36201 been defined.

36202 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 36203 internationalization variables.

36204 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 36205 characters (for example, single-byte as opposed to multi-byte characters in  
 36206 arguments).

36207 *LC\_MESSAGES*

36208 Determine the locale that should be used to affect the format and contents of  
 36209 diagnostic messages written to standard error.

36210 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36211 **ASYNCHRONOUS EVENTS**

36212 Default.

**36213 STDOUT**

36214 The standard output shall be used when no *blocks* operand is present. If the current number of  
36215 blocks is limited, the number of blocks in the current limit shall be written in the following  
36216 format:

36217 "%d\n", <number of 512-byte blocks>

36218 If there is no current limit on the number of blocks, in the POSIX locale the following format  
36219 shall be used:

36220 "unlimited\n"

**36221 STDERR**

36222 Used only for diagnostic messages.

**36223 OUTPUT FILES**

36224 None.

**36225 EXTENDED DESCRIPTION**

36226 None.

**36227 EXIT STATUS**

36228 The following exit values shall be returned:

36229 0 Successful completion.

36230 >0 A request for a higher limit was rejected or an error occurred.

**36231 CONSEQUENCES OF ERRORS**

36232 Default.

**36233 APPLICATION USAGE**

36234 Since *ulimit* affects the current shell execution environment, it is always provided as a shell  
36235 regular built-in. If it is called in separate utility execution environment, such as one of the  
36236 following:

36237 nohup ulimit -f 10000

36238 env ulimit 10000

36239 it does not affect the file size limit of the caller's environment.

36240 Once a limit has been decreased by a process, it cannot be increased (unless appropriate  
36241 privileges are involved), even back to the original system limit.

**36242 EXAMPLES**

36243 Set the file size limit to 51 200 bytes:

36244 ulimit -f 100

**36245 RATIONALE**

36246 None.

**36247 FUTURE DIRECTIONS**

36248 None.

**36249 SEE ALSO**

36250 The System Interfaces volume of IEEE Std. 1003.1-200x, *ulimit()*

**36251 CHANGE HISTORY**

36252 First released in Issue 2.

36253 **Issue 4**

36254 Relocated from the *sh* description to reflect its status as a regular built-in utility.

36255 **NAME**

36256           umask — get or set the file mode creation mask

36257 **SYNOPSIS**36258           umask [-S][*mask*]36259 **DESCRIPTION**

36260           The *umask* utility shall set the file mode creation mask of the current shell execution  
 36261           environment (see Section 2.13 (on page 2273)) to the value specified by the *mask* operand. This  
 36262           mask shall affect the initial value of the file permission bits of subsequently created files. If *umask*  
 36263           is called in a subshell or separate utility execution environment, such as one of the following:

36264            (*umask* 002)36265           nohup *umask* ...36266           find . -exec *umask* ... \;

36267           it shall not affect the file mode creation mask of the caller's environment.

36268           If the *mask* operand is not specified, the *umask* utility shall write to standard output the value of  
 36269           the invoking process's file mode creation mask.

36270 **OPTIONS**

36271           The *umask* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 36272           12.2, Utility Syntax Guidelines.

36273           The following option shall be supported:

36274           -S           Produce symbolic output.

36275           The default output style is unspecified, but shall be recognized on a subsequent invocation of  
 36276           *umask* on the same system as a *mask* operand to restore the previous file mode creation mask.

36277 **OPERANDS**

36278           The following operand shall be supported:

36279           *mask*           A string specifying the new file mode creation mask. The string is treated in the  
 36280           same way as the *mode* operand described in the the EXTENDED DESCRIPTION  
 36281           section for *chmod*.

36282           For a *symbolic\_mode* value, the new value of the file mode creation mask shall be  
 36283           the logical complement of the file permission bits portion of the file mode specified  
 36284           by the *symbolic\_mode* string.

36285           In a *symbolic\_mode* value, the permissions *op* characters '+' and '-' shall be  
 36286           interpreted relative to the current file mode creation mask; '+' shall cause the bits  
 36287           for the indicated permissions to be cleared in the mask; '-' shall cause the bits for  
 36288           the indicated permissions to be set in the mask.

36289           The interpretation of *mode* values that specify file mode bits other than the file  
 36290           permission bits is unspecified.

36291           In the octal integer form of *mode*, the specified bits are set in the file mode creation  
 36292           mask.

36293           The file mode creation mask shall be set to the resulting numeric value.

36294           The default output of a prior invocation of *umask* on the same system with no  
 36295           operand also shall be recognized as a *mask* operand.

36296 **STDIN**

36297 Not used.

36298 **INPUT FILES**

36299 None.

36300 **ENVIRONMENT VARIABLES**36301 The following environment variables shall affect the execution of *umask*:

36302 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 36303 If *LANG* is unset or null, the corresponding value from the implementation-  
 36304 defined default locale shall be used. If any of the internationalization variables  
 36305 contains an invalid setting, the utility shall behave as if none of the variables had  
 36306 been defined.

36307 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 36308 internationalization variables.

36309 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 36310 characters (for example, single-byte as opposed to multi-byte characters in  
 36311 arguments).

36312 *LC\_MESSAGES*

36313 Determine the locale that should be used to affect the format and contents of  
 36314 diagnostic messages written to standard error.

36315 *XS* *NLS\_PATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36316 **ASYNCHRONOUS EVENTS**

36317 Default.

36318 **STDOUT**

36319 When the *mask* operand is not specified, the *umask* utility shall write a message to standard  
 36320 output that can later be used as a *umask mask* operand.

36321 If *-S* is specified, the message shall be in the following format:

36322 "u=%s,g=%s,o=%s\n", <owner permissions>, <group permissions>,  
 36323 <other permissions>

36324 where the three values shall be combinations of letters from the set {r, w, x}; the presence of a  
 36325 letter shall indicate that the corresponding bit is clear in the file mode creation mask.

36326 If a *mask* operand is specified, there shall be no output written to standard output.

36327 **STDERR**

36328 Used only for diagnostic messages.

36329 **OUTPUT FILES**

36330 None.

36331 **EXTENDED DESCRIPTION**

36332 None.

36333 **EXIT STATUS**

36334 The following exit values shall be returned:

36335 0 The file mode creation mask was successfully changed, or no *mask* operand was supplied.

36336 >0 An error occurred.

## 36337 CONSEQUENCES OF ERRORS

36338 Default.

## 36339 APPLICATION USAGE

36340 Since *umask* affects the current shell execution environment, it is generally provided as a shell  
 36341 regular built-in.

36342 In contrast to the negative permission logic provided by the file mode creation mask and the  
 36343 octal number form of the *mask* argument, the symbolic form of the *mask* argument specifies those  
 36344 permissions that are left alone.

## 36345 EXAMPLES

36346 Either of the commands:

36347 `umask a=rx,ug+w`36348 `umask 002`36349 sets the mode mask so that subsequently created files have their `S_IWOTH` bit cleared.

36350 After setting the mode mask with either of the above commands, the *umask* command can be  
 36351 used to write out the current value of the mode mask:

36352 `$ umask`36353 `0002`

36354 (The output format is unspecified, but historical implementations use the octal integer mode  
 36355 format.)

36356 `$ umask -S`36357 `u=rwx,g=rwx,o=rx`

36358 Either of these outputs can be used as the mask operand to a subsequent invocation of the *umask*  
 36359 utility.

36360 Assuming the mode mask is set as above, the command:

36361 `umask g-w`

36362 sets the mode mask so that subsequently created files have their `S_IWGRP` and `S_IWOTH` bits  
 36363 cleared.

36364 The command:

36365 `umask -- -w`

36366 sets the mode mask so that subsequently created files have all their write bits cleared. Note that  
 36367 *mask* operands `-r`, `-w`, `-x` or anything beginning with a hyphen, must be preceded by `--` to  
 36368 keep it from being interpreted as an option.

## 36369 RATIONALE

36370 Since *umask* affects the current shell execution environment, it is generally provided as a shell  
 36371 regular built-in. If it is called in a subshell or separate utility execution environment, such as one  
 36372 of the following:

36373 `(umask 002)`36374 `nohup umask ...`36375 `find . -exec umask ... \;`

36376 it does not affect the file mode creation mask of the environment of the caller.

36377 The description of the historical utility was modified to allow it to use the symbolic modes of  
 36378 *chmod*. The `-s` option used in early proposals was changed to `-S` because `-s` could be confused



36379 with a *symbolic\_mode* form of mask referring to the S\_ISUID and S\_ISGID bits.

36380 **Notes to Reviewers**

36381 *This section with side shading will not appear in the final copy. - Ed.*

36382 D1, XCU, ERN 355 suggests we should specify the default output. Suggestions please.

36383 The default output style is implementation-defined to permit implementors to provide  
36384 migration to the new symbolic style at the time most appropriate to their users. An `-o` flag to  
36385 force octal mode output was omitted because the octal mode may not be sufficient to specify all  
36386 of the information that may be present in the file mode creation mask when more secure file  
36387 access permission checks are implemented.

36388 It has been suggested that trusted systems developers might appreciate ameliorating the  
36389 requirement that the mode mask “affects” the file access permissions, since it seems access  
36390 control lists might replace the mode mask to some degree. The wording has been changed to say  
36391 that it affects the file permission bits, and it leaves the details of the behavior of how they affect  
36392 the file access permissions to the description in the System Interfaces volume of  
36393 IEEE Std. 1003.1-200x.

36394 **FUTURE DIRECTIONS**

36395 None.

36396 **SEE ALSO**

36397 *chmod*, the System Interfaces volume of IEEE Std. 1003.1-200x, *umask()*

36398 **CHANGE HISTORY**

36399 First released in Issue 2.

36400 **Issue 4**

36401 Aligned with the ISO/IEC 9945-2:1993 standard.

36402 **Issue 6**

36403 The following new requirements on POSIX implementations derive from alignment with the  
36404 Single UNIX Specification:

- 36405
- The octal mode is supported.

36406 **NAME**

36407 unalias — remove alias definitions

36408 **SYNOPSIS**36409 UP unalias *alias-name*...

36410 unalias -a

36411

36412 **DESCRIPTION**

36413 The *unalias* utility shall remove the definition for each alias name specified. See Section 2.3.1 (on  
 36414 page 2239). The aliases shall be removed from the current shell execution environment; see  
 36415 Section 2.13 (on page 2273).

36416 **OPTIONS**

36417 The *unalias* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 36418 12.2, Utility Syntax Guidelines.

36419 The following option shall be supported:

36420 **-a** Remove all alias definitions from the current shell execution environment.

36421 **OPERANDS**

36422 The following operand shall be supported:

36423 *alias-name* The name of an alias to be removed.

36424 **STDIN**

36425 Not used.

36426 **INPUT FILES**

36427 None.

36428 **ENVIRONMENT VARIABLES**

36429 The following environment variables shall affect the execution of *unalias*:

36430 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 36431 If **LANG** is unset or null, the corresponding value from the implementation-  
 36432 defined default locale shall be used. If any of the internationalization variables  
 36433 contains an invalid setting, the utility shall behave as if none of the variables had  
 36434 been defined.

36435 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 36436 internationalization variables.

36437 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 36438 characters (for example, single-byte as opposed to multi-byte characters in  
 36439 arguments).

36440 **LC\_MESSAGES**

36441 Determine the locale that should be used to affect the format and contents of  
 36442 diagnostic messages written to standard error.

36443 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

36444 **ASYNCHRONOUS EVENTS**

36445 Default.

36446 **STDOUT**

36447 Not used.

36448 **STDERR**

36449 Used only for diagnostic messages.

36450 **OUTPUT FILES**

36451 None.

36452 **EXTENDED DESCRIPTION**

36453 None.

36454 **EXIT STATUS**

36455 The following exit values shall be returned:

36456 0 Successful completion.

36457 >0 One of the *alias-name* operands specified did not represent a valid alias definition, or an  
36458 error occurred.36459 **CONSEQUENCES OF ERRORS**

36460 Default.

36461 **APPLICATION USAGE**36462 Since *unalias* affects the current shell execution environment, it is generally provided as a shell  
36463 regular built-in.36464 **EXAMPLES**

36465 None.

36466 **RATIONALE**36467 The *unalias* description is based on that from historical KornShell implementations. Known  
36468 differences exist between that and the C shell. The KornShell version was adopted to be  
36469 consistent with all the other KornShell features in this volume of IEEE Std. 1003.1-200x, such as  
36470 command line editing.36471 The *-a* option is the equivalent of the *unalias \** form of the C shell and is provided to address  
36472 security concerns about unknown aliases entering the environment of a user (or application)  
36473 through the allowable implementation-defined predefined alias route or as a result of an *ENV*  
36474 file. (Although *unalias* could be used to simplify the “secure” shell script shown in the *command*  
36475 rationale, it does not obviate the need to quote all command names. An initial call to *unalias -a*  
36476 would have to be quoted in case there was an alias for *unalias*.)36477 **FUTURE DIRECTIONS**

36478 None.

36479 **SEE ALSO**36480 *alias*36481 **CHANGE HISTORY**

36482 First released in Issue 4.

36483 **Issue 6**

36484 This utility is now marked as part of the User Portability Utilities option.

36485 **NAME**36486            **uname** — return system name36487 **SYNOPSIS**36488            **uname** [**-snrvma**]36489 **DESCRIPTION**

36490            By default, the *uname* utility shall write the operating system name to standard output. When  
 36491            options are specified, symbols representing one or more system characteristics shall be written  
 36492            to the standard output. The format and contents of the symbols are implementation-defined. On  
 36493            systems conforming to the System Interfaces volume of IEEE Std. 1003.1-200x, the symbols  
 36494            written shall be those supported by the *uname()* function as defined in the System Interfaces  
 36495            volume of IEEE Std. 1003.1-200x.

36496 **OPTIONS**

36497            The *uname* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 36498            12.2, Utility Syntax Guidelines.

36499            The following options shall be supported:

- 36500            **-a**            Behave as though all of the options **-mnrsv** were specified.
- 36501            **-m**            Write the name of the hardware type on which the system is running to standard  
 36502            output.
- 36503            **-n**            Write the name of this node within an implementation-defined communications  
 36504            network.
- 36505            **-r**            Write the current release level of the operating system implementation.
- 36506            **-s**            Write the name of the implementation of the operating system.
- 36507            **-v**            Write the current version level of this release of the operating system  
 36508            implementation.

36509            If no options are specified, the *uname* utility shall write the operating system name, as if the **-s**  
 36510            option had been specified.

36511 **OPERANDS**

36512            None.

36513 **STDIN**

36514            Not used.

36515 **INPUT FILES**

36516            None.

36517 **ENVIRONMENT VARIABLES**

36518            The following environment variables shall affect the execution of *uname*:

- 36519            **LANG**        Provide a default value for the internationalization variables that are unset or null.  
 36520            If *LANG* is unset or null, the corresponding value from the implementation-  
 36521            defined default locale shall be used. If any of the internationalization variables  
 36522            contains an invalid setting, the utility shall behave as if none of the variables had  
 36523            been defined.
- 36524            **LC\_ALL**        If set to a non-empty string value, override the values of all the other  
 36525            internationalization variables.
- 36526            **LC\_CTYPE**    Determine the locale for the interpretation of sequences of bytes of text data as  
 36527            characters (for example, single-byte as opposed to multi-byte characters in

36528 arguments).

36529 **LC\_MESSAGES**

36530 Determine the locale that should be used to affect the format and contents of

36531 diagnostic messages written to standard error.

36532 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

36533 **ASYNCHRONOUS EVENTS**

36534 Default.

36535 **STDOUT**

36536 By default, the output shall be a single line of the following form:

36537 "%s\n", <sysname>

36538 If the **-a** option is specified, the output shall be a single line of the following form:

36539 "%s %s %s %s %s\n", <sysname>, <nodename>, <release>,  
36540 <version>, <machine>

36541 Additional implementation-defined symbols may be written; all such symbols shall be written at  
36542 the end of the line of output before the <newline> character.

36543 If options are specified to select different combinations of the symbols, only those symbols shall  
36544 be written, in the order shown above for the **-a** option. If a symbol is not selected for writing, its  
36545 corresponding trailing <blank> characters also shall not be written.

36546 **STDERR**

36547 Used only for diagnostic messages.

36548 **OUTPUT FILES**

36549 None.

36550 **EXTENDED DESCRIPTION**

36551 None.

36552 **EXIT STATUS**

36553 The following exit values shall be returned:

36554 0 The requested information was successfully written.

36555 >0 An error occurred.

36556 **CONSEQUENCES OF ERRORS**

36557 Default.

36558 **APPLICATION USAGE**

36559 Note that any of the symbols could include embedded <space> characters, which may affect  
36560 parsing algorithms if multiple options are selected for output.

36561 The node name is typically a name that the system uses to identify itself for intersystem  
36562 communication addressing.

36563 **EXAMPLES**

36564 The following command:

36565 `uname -sr`

36566 writes the operating system name and release level, separated by one or more <blank>  
36567 characters.

**36568 RATIONALE**

36569 It was suggested that this utility cannot be used portably since the format of the symbols is  
36570 implementation-defined. The POSIX.1 working group could not achieve consensus on defining  
36571 these formats in the underlying *uname()* function, and there was no expectation that this volume  
36572 of IEEE Std. 1003.1-200x would be any more successful. Some applications may still find this  
36573 historical utility of value. For example, the symbols could be used for system log entries or for  
36574 comparison with operator or user input.

**36575 FUTURE DIRECTIONS**

36576 None.

**36577 SEE ALSO**

36578 The System Interfaces volume of IEEE Std. 1003.1-200x, *uname()*

**36579 CHANGE HISTORY**

36580 First released in Issue 2.

**36581 Issue 4**

36582 Aligned with the ISO/IEC 9945-2:1993 standard.

**36583 Issue 4, Version 2**

36584 The SYNOPSIS section lists all the valid options.

36585 **NAME**

36586 uncompress — expand compressed data

36587 **SYNOPSIS**36588 xSI uncompress [-cfv][*file...*]

36589

36590 **DESCRIPTION**

36591 The *uncompress* utility shall restore files to their original state after they have been compressed  
 36592 using the *compress* utility. If no files are specified, the standard input shall be uncompressed to  
 36593 the standard output. If the invoking process has appropriate privileges, the ownership, modes,  
 36594 access time, and modification time of the original file shall be preserved.

36595 This utility shall support the uncompressing of any files produced by the *compress* utility on the  
 36596 same implementation. For files produced by *compress* on other systems, *uncompress* supports 9 to  
 36597 14-bit compression (see *compress* (on page 2477), **-b**); it is implementation-defined whether  
 36598 values of **-b** greater than 14 are supported.

36599 **OPTIONS**

36600 The *uncompress* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x,  
 36601 Section 12.2, Utility Syntax Guidelines.

36602 The following options shall be supported:

- 36603 **-c** Write to standard output; no files are changed.
- 36604 **-f** Do not prompt for overwriting files. Except when run in the background, if **-f** is  
 36605 not given the user shall be prompted as to whether an existing file should be  
 36606 overwritten. If the standard input is not a terminal and **-f** is not given, *uncompress*  
 36607 shall write a diagnostic message to standard error and exit with a status greater  
 36608 than zero.
- 36609 **-v** Write messages to standard error concerning the expansion of each file.

36610 **OPERANDS**

36611 The following operand shall be supported:

- 36612 *file* A path name of a file. If *file* already has the **.Z** suffix specified, it shall be used as  
 36613 the input file and the output file shall be named **file** with the **.Z** suffix removed.  
 36614 Otherwise, *file* shall be used as the name of the output file and **file** with the **.Z**  
 36615 suffix appended shall be used as the input file.

36616 **STDIN**

36617 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '- '.

36618 **INPUT FILES**

36619 Input files shall be in the format produced by the *compress* utility.

36620 **ENVIRONMENT VARIABLES**

36621 The following environment variables shall affect the execution of *uncompress*:

- 36622 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 36623 If **LANG** is unset or null, the corresponding value from the implementation-  
 36624 defined default locale shall be used. If any of the internationalization variables  
 36625 contains an invalid setting, the utility shall behave as if none of the variables had  
 36626 been defined.
- 36627 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 36628 internationalization variables.

- 36629            *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
36630 characters (for example, single-byte as opposed to multi-byte characters in  
36631 arguments).
- 36632            *LC\_MESSAGES*  
36633                    Determine the locale that should be used to affect the format and contents of  
36634 diagnostic messages written to standard error.
- 36635            *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 36636 **ASYNCHRONOUS EVENTS**
- 36637            Default.
- 36638 **STDOUT**
- 36639            When there are no *file* operands or the *-c* option is specified, the uncompressed output is written  
36640 to standard output.
- 36641 **STDERR**
- 36642            Prompts shall be written to the standard error output under the conditions specified in the  
36643 DESCRIPTION and OPTIONS sections. The prompts shall contain the *file* path name, but their  
36644 format is otherwise unspecified. Otherwise, the standard error output shall be used only for  
36645 diagnostic messages.
- 36646 **OUTPUT FILES**
- 36647            Output files are the same as the respective input files to *compress*.
- 36648 **EXTENDED DESCRIPTION**
- 36649            None.
- 36650 **EXIT STATUS**
- 36651            The following exit values shall be returned:
- 36652            0 Successful completion.
- 36653            >0 An error occurred.
- 36654 **CONSEQUENCES OF ERRORS**
- 36655            The input file remains unmodified.
- 36656 **APPLICATION USAGE**
- 36657            The limit of 14 on the *compress -b bits* argument is to achieve portability to all systems (within  
36658 the restrictions imposed by the lack of an explicit published file format). Some systems based on  
36659 16-bit architectures cannot support 15 or 16-bit uncompression.
- 36660 **EXAMPLES**
- 36661            None.
- 36662 **RATIONALE**
- 36663            None.
- 36664 **FUTURE DIRECTIONS**
- 36665            None.
- 36666 **SEE ALSO**
- 36667            *compress, zcat*
- 36668 **CHANGE HISTORY**
- 36669            First released in Issue 4.



36670 **Issue 4, Version 2**

36671 The DESCRIPTION is clarified to state that the ownership, modes, access time, and modification  
36672 time of the original file are preserved if the invoking process has appropriate privileges.

36673 **Issue 6**

36674 The normative text is reworded to avoid use of the term “must” for application requirements.

## 36675 NAME

36676 unexpand — convert spaces to tabs

## 36677 SYNOPSIS

36678 UP unexpand [ -a | -t *tablist* ][*file...*]

36679

## 36680 DESCRIPTION

36681 The *unexpand* utility shall copy files or standard input to standard output, converting <blank>  
 36682 characters at the beginning of each line into the maximum number of <tab> characters followed  
 36683 by the minimum number of <space> characters needed to fill the same column positions  
 36684 originally filled by the translated <blank> characters. By default, tabstops shall be set at every  
 36685 eighth column position. Each <backspace> character shall be copied to the output, and shall  
 36686 cause the column position count for tab calculations to be decremented; the count shall never be  
 36687 decremented to a value less than one.

## 36688 OPTIONS

36689 The *unexpand* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x,  
 36690 Section 12.2, Utility Syntax Guidelines.

36691 The following options shall be supported:

36692 **-a** In addition to translating <blank> characters at the beginning of each line, translate  
 36693 all sequences of two or more <blank> characters immediately preceding a tab stop  
 36694 to the maximum number of <tab> characters followed by the minimum number of  
 36695 <space> characters needed to fill the same column positions originally filled by the  
 36696 translated <blank> characters.

36697 **-t *tablist*** Specify the tab stops. The application shall ensure that the *tablist* option-argument  
 36698 is a single argument consisting of a single positive decimal integer or multiple  
 36699 positive decimal integers, separated by <blank> characters or commas, in  
 36700 ascending order. If a single number is given, tabs shall be set *tablist* column  
 36701 positions apart instead of the default 8. If multiple numbers are given, the tabs  
 36702 shall be set at those specific column positions.

36703 The application shall ensure that each tab-stop position *N* is an integer value  
 36704 greater than zero, and the list shall be in strictly ascending order. This is taken to  
 36705 mean that, from the start of a line of output, tabbing to position *N* shall cause the  
 36706 next character output to be in the (*N*+1)th column position on that line. When the  
 36707 **-t** option is not specified, the default shall be the equivalent of specifying **-t 8**  
 36708 (except for the interaction with **-a**, described below).

36709 No <space>-to-<tab> character conversions shall occur for characters at positions  
 36710 beyond the last of those specified in a multiple tab-stop list.

36711 When **-t** is specified, the presence or absence of the **-a** option shall be ignored;  
 36712 conversion shall not be limited to the processing of leading <blank> characters.

## 36713 OPERANDS

36714 The following operand shall be supported:

36715 *file* A path name of a text file to be used as input.

## 36716 STDIN

36717 See the INPUT FILES section.

36718 **INPUT FILES**

36719           The input files shall be text files.

36720 **ENVIRONMENT VARIABLES**

36721           The following environment variables shall affect the execution of *unexpand*:

36722           *LANG*       Provide a default value for the internationalization variables that are unset or null.  
36723                       If *LANG* is unset or null, the corresponding value from the implementation-  
36724                       defined default locale shall be used. If any of the internationalization variables  
36725                       contains an invalid setting, the utility shall behave as if none of the variables had  
36726                       been defined.

36727           *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
36728                       internationalization variables.

36729           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
36730                       characters (for example, single-byte as opposed to multi-byte characters in  
36731                       arguments and input files), the processing of <tab> and <space> characters and for  
36732                       the determination of the width in column positions each character would occupy  
36733                       on an output device.

36734           *LC\_MESSAGES*

36735                       Determine the locale that should be used to affect the format and contents of  
36736                       diagnostic messages written to standard error.

36737 XSI           *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36738 **ASYNCHRONOUS EVENTS**

36739           Default.

36740 **STDOUT**

36741           The standard output is equivalent to the input files with the specified <space>-to-<tab>  
36742           character conversions.

36743 **STDERR**

36744           Used only for diagnostic messages.

36745 **OUTPUT FILES**

36746           None.

36747 **EXTENDED DESCRIPTION**

36748           None.

36749 **EXIT STATUS**

36750           The following exit values shall be returned:

36751           0   Successful completion.

36752           >0  An error occurred.

36753 **CONSEQUENCES OF ERRORS**

36754           Default.

**36755 APPLICATION USAGE**

36756 One non-intuitive aspect of *unexpand* is its restriction to leading spaces when neither **-a** nor **-t** is  
36757 specified. Users who desire to always convert all spaces in a file can easily alias *unexpand* to use  
36758 the **-a** or **-t 8** option.

**36759 EXAMPLES**

36760 None.

**36761 RATIONALE**

36762 On several occasions, consideration was given to adding a **-t** option to the *unexpand* utility to  
36763 complement the **-t** in *expand* (see *expand* (on page 2636)). The historical intent of *unexpand* was  
36764 to translate multiple <blank>s into tab stops, where tab stops were a multiple of eight column  
36765 positions on most UNIX systems. An early proposal omitted **-t** because it seemed outside the  
36766 scope of the UPE; it was not described in any of the base documents. However, hard-coding tab  
36767 stops every eight columns was not suitable for the international community and broke historical  
36768 precedents for some vendors in the FORTRAN community, so **-t** was restored in conjunction  
36769 with the list of valid extension categories considered by the standard developers. Thus, *unexpand*  
36770 is now the logical converse of *expand*.

**36771 FUTURE DIRECTIONS**

36772 None.

**36773 SEE ALSO**

36774 *expand*, *tabs*

**36775 CHANGE HISTORY**

36776 First released in Issue 4.

**36777 Issue 6**

36778 This utility is now marked as part of the User Portability Utilities option.

36779 The definition of the *LC\_CTYPE* environment variable is changed to align with the  
36780 IEEE P1003.2b draft standard.

36781 The normative text is reworded to avoid use of the term “must” for application requirements.

36782 **NAME**36783 unget — undo a previous get of an SCCS file (**DEVELOPMENT**)36784 **SYNOPSIS**36785 xSI unget [-ns][-r *SID*] *file...*

36786

36787 **DESCRIPTION**36788 The *unget* utility shall reverse the effect of a *get -e* done prior to creating the intended new delta.36789 **OPTIONS**36790 The *unget* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
36791 12.2, Utility Syntax Guidelines.

36792 The following options shall be supported:

36793 **-r** *SID* Uniquely identify which delta is no longer intended. (This would have been  
36794 specified by *get* as the new delta.) The use of this option is necessary only if two or  
36795 more outstanding *get* commands for editing on the same SCCS file were done by  
36796 the same person (login name).36797 **-s** Suppress the writing to standard output of the intended delta's *SID*.36798 **-n** Retain the file that was obtained by *get*, which would normally be removed from  
36799 the current directory.36800 **OPERANDS**

36801 The following operands shall be supported:

36802 **file** A path name of an existing SCCS file or a directory. If *file* is a directory, the *unget*  
36803 utility shall behave as though each file in the directory were specified as a named  
36804 file, except that non-SCCS files (last component of the path name does not begin  
36805 with *s.*) and unreadable files shall be silently ignored.36806 If a single instance *file* is specified as *'-'*, the standard input shall be read; each  
36807 line of the standard input shall be taken to be the name of an SCCS file to be  
36808 processed. Non-SCCS files and unreadable files shall be silently ignored.36809 **STDIN**36810 The standard input shall be a text file used only when the *file* operand is specified as *'-'*. Each  
36811 line of the text file shall be interpreted as an SCCS path name.36812 **INPUT FILES**

36813 Any SCCS files processed are files of an unspecified format.

36814 **ENVIRONMENT VARIABLES**36815 The following environment variables shall affect the execution of *unget*:36816 **LANG** Provide a default value for the internationalization variables that are unset or null.  
36817 If *LANG* is unset or null, the corresponding value from the implementation-  
36818 defined default locale shall be used. If any of the internationalization variables  
36819 contains an invalid setting, the utility shall behave as if none of the variables had  
36820 been defined.36821 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
36822 internationalization variables.36823 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
36824 characters (for example, single-byte as opposed to multi-byte characters in  
36825 arguments and input files).

- 36826 **LC\_MESSAGES**  
36827 Determine the locale that should be used to affect the format and contents of  
36828 diagnostic messages written to standard error.
- 36829 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 36830 **ASYNCHRONOUS EVENTS**  
36831 Default.
- 36832 **STDOUT**  
36833 The standard output shall consist of a line for each file, in the following format:  
36834 "%s\n", <*SID removed from file*>  
36835 If there is more than one named file or if a directory or standard input is named, each path name  
36836 shall be written before each of the preceding lines:  
36837 "\n%s:\n", <*pathname*>
- 36838 **STDERR**  
36839 Used only for diagnostic messages.
- 36840 **OUTPUT FILES**  
36841 Any SCCS files updated are files of an unspecified format. During processing of a *file*, a locking  
36842 *z-file*, as described in *get*, and a *q-file* (a working copy of the *p-file*), may be created and deleted.  
36843 The *p-file* and *g-file*, as described in *get*, shall be deleted.
- 36844 **EXTENDED DESCRIPTION**  
36845 None.
- 36846 **EXIT STATUS**  
36847 The following exit values shall be returned:  
36848 0 Successful completion.  
36849 >0 An error occurred.
- 36850 **CONSEQUENCES OF ERRORS**  
36851 Default.
- 36852 **APPLICATION USAGE**  
36853 None.
- 36854 **EXAMPLES**  
36855 None.
- 36856 **RATIONALE**  
36857 None.
- 36858 **FUTURE DIRECTIONS**  
36859 None.
- 36860 **SEE ALSO**  
36861 *delta, get, sact*
- 36862 **CHANGE HISTORY**  
36863 First released in Issue 2.  
36864 **Issue 4**  
36865 Format reorganized.  
36866 Utility Syntax Guidelines support mandated.

36867 Internationalized environment variable support mandated.

36868 **Issue 6**

36869 The normative text is reworded to avoid use of the term “must” for application requirements. |

36870 The normative text is reworded to emphasise the term “shall” for implementation requirements. |

36871 **NAME**

36872            *uniq* — report or filter out repeated lines in a file

36873 **SYNOPSIS**

36874            *uniq* [-c|-d|-u][-f *fields*][-s *char*][*input\_file* [*output\_file*]]

36875 **DESCRIPTION**

36876            The *uniq* utility shall read an input file comparing adjacent lines, and writes one copy of each  
36877            input line on the output. The second and succeeding copies of repeated adjacent input lines shall  
36878            not be written.

36879            Repeated lines in the input shall not be detected if they are not adjacent.

36880 **OPTIONS**

36881            The *uniq* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
36882            12.2, Utility Syntax Guidelines.

36883            The following options shall be supported:

36884            -c            Precede each output line with a count of the number of times the line occurred in  
36885            the input.

36886            -d            Suppress the writing of lines that are not repeated in the input.

36887            -f *fields*    Ignore the first *fields* fields on each input line when doing comparisons, where  
36888            *fields* is a positive decimal integer. A field is the maximal string matched by the  
36889            basic regular expression:

36890            `[[:blank:]]*[^[:blank:]]*`

36891            If the *fields* option-argument specifies more fields than appear on an input line, a  
36892            null string shall be used for comparison.

36893            -s *chars*    Ignore the first *chars* characters when doing comparisons, where *chars* shall be a  
36894            positive decimal integer. If specified in conjunction with the -f option, the first  
36895            *chars* characters after the first *fields* fields shall be ignored. If the *chars* option-  
36896            argument specifies more characters than remain on an input line, a null string shall  
36897            be used for comparison.

36898            -u            Suppress the writing of lines that are repeated in the input.

36899 **OPERANDS**

36900            The following operands shall be supported:

36901            *input\_file*    A path name of the input file. If the *input\_file* operand is not specified, or if the  
36902            *input\_file* is '-', the standard input is used.

36903            *output\_file*    A path name of the output file. If the *output\_file* operand is not specified, the  
36904            standard output shall be used. The results are unspecified if the file named by  
36905            *output\_file* is the file named by *input\_file*.

36906 **STDIN**

36907            The standard input shall be used only if no *input\_file* operand is specified or if *input\_file* is '-'.  
36908            See the INPUT FILES section.

36909 **INPUT FILES**

36910            The input file shall be a text file.



36911 **ENVIRONMENT VARIABLES**

36912 The following environment variables shall affect the execution of *uniq*:

36913 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 36914 If *LANG* is unset or null, the corresponding value from the implementation-  
 36915 defined default locale shall be used. If any of the internationalization variables  
 36916 contains an invalid setting, the utility shall behave as if none of the variables had  
 36917 been defined.

36918 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 36919 internationalization variables.

36920 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 36921 characters (for example, single-byte as opposed to multi-byte characters in  
 36922 arguments and input files) which characters constitute a <blank> character in the  
 36923 current locale.

36924 *LC\_MESSAGES*  
 36925 Determine the locale that should be used to affect the format and contents of  
 36926 diagnostic messages written to standard error.

36927 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36928 **ASYNCHRONOUS EVENTS**

36929 Default.

36930 **STDOUT**

36931 The standard output shall be used only if no *output\_file* operand is specified. See the OUTPUT  
 36932 FILES section.

36933 **STDERR**

36934 Used only for diagnostic messages.

36935 **OUTPUT FILES**

36936 If the *-c* option is specified, the application shall ensure that the output file is empty or each line  
 36937 shall be of the form:

36938 "%d %s", <number of duplicates>, <line>

36939 otherwise, the application shall ensure that the output file is empty or each line shall be of the  
 36940 form:

36941 "%s", <line>

36942 **EXTENDED DESCRIPTION**

36943 None.

36944 **EXIT STATUS**

36945 The following exit values shall be returned:

36946 0 The utility executed successfully.

36947 >0 An error occurred.

36948 **CONSEQUENCES OF ERRORS**

36949 Default.

36950 **APPLICATION USAGE**

36951 The *sort* utility can be used to cause repeated lines to be adjacent in the input file.

36952 **EXAMPLES**

36953 The following input file data (but flushed left) was used for a test series on *uniqu*:

```
36954 #01 foo0 bar0 fool bar1
36955 #02 bar0 fool bar1 fool
36956 #03 foo0 bar0 fool bar1
36957 #04
36958 #05 foo0 bar0 fool bar1
36959 #06 foo0 bar0 fool bar1
36960 #07 bar0 fool bar1 foo0
```

36961 What follows is a series of test invocations of the *uniqu* utility that use a mixture of *uniqu* options  
 36962 against the input file data. These tests verify the meaning of *adjacent*. The *uniqu* utility views the  
 36963 input data as a sequence of strings delimited by '\n'. Accordingly, for the *fieldsth* member of  
 36964 the sequence, *uniqu* interprets unique or repeated adjacent lines strictly relative to the *fields+1*th  
 36965 member.

36966 1. This first example tests the line counting option, comparing each line of the input file data  
 36967 starting from the second field:

```
36968 uniqu -c -f 1 uniqu_0I.t
36969 1 #01 foo0 bar0 fool bar1
36970 1 #02 bar0 fool bar1 foo0
36971 1 #03 foo0 bar0 fool bar1
36972 1 #04
36973 2 #05 foo0 bar0 fool bar1
36974 1 #07 bar0 fool bar1 foo0
```

36975 The number '2', prefixing the fifth line of output, signifies that the *uniqu* utility detected a  
 36976 pair of repeated lines. Given the input data, this can only be true when *uniqu* is run using  
 36977 the *-f 1* option (which shall cause *uniqu* to ignore the first field on each input line).

36978 2. The second example tests the option to suppress unique lines, comparing each line of the  
 36979 input file data starting from the second field:

```
36980 uniqu -d -f 1 uniqu_0I.t
36981 #05 foo0 bar0 fool bar1
```

36982 3. This test suppresses repeated lines, comparing each line of the input file data starting from  
 36983 the second field:

```
36984 uniqu -u -f 1 uniqu_0I.t
36985 #01 foo0 bar0 fool bar1
36986 #02 bar0 fool bar1 fool
36987 #03 foo0 bar0 fool bar1
36988 #04
36989 #07 bar0 fool bar1 foo0
```

36990 4. This suppresses unique lines, comparing each line of the input file data starting from the  
 36991 third character:

```
36992 uniqu -d -s 2 uniqu_0I.t
```

36993 In the last example, the *uniqu* utility found no input matching the above criteria.

36994 **RATIONALE**

36995           Some historical implementations have limited lines to be 1 080 bytes in length, which does not  
36996           meet the implied {LINE\_MAX} limit.

36997 **FUTURE DIRECTIONS**

36998           None.

36999 **SEE ALSO**

37000           *comm, sort*

37001 **CHANGE HISTORY**

37002           First released in Issue 2.

37003 **Issue 4**

37004           Aligned with the ISO/IEC 9945-2: 1993 standard.

37005 **Issue 6**

37006           The obsolescent SYNOPSIS and associated text are removed.

37007           The normative text is reworded to avoid use of the term “must” for application requirements.

37008 **NAME**

37009 unlink — call the *unlink()* function

37010 **SYNOPSIS**

37011 xSI unlink *file*

37012

37013 **DESCRIPTION**

37014 The *unlink* utility shall perform the function call:

37015 unlink(*file*);

37016 A user may need appropriate privilege to invoke the *unlink* utility.

37017 **OPTIONS**

37018 None.

37019 **OPERANDS**

37020 The following operands shall be supported:

37021 *file* The path name of an existing file.

37022 **STDIN**

37023 Not used.

37024 **INPUT FILES**

37025 Not used.

37026 **ENVIRONMENT VARIABLES**

37027 The following environment variables shall affect the execution of *unlink*:

37028 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 37029 If *LANG* is unset or null, the corresponding value from the implementation-  
 37030 defined default locale shall be used. If any of the internationalization variables  
 37031 contain an invalid setting, the utility behaves as if none of the variables had been  
 37032 set.

37033 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 37034 internationalization variables.

37035 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 37036 characters (for example, single-byte as opposed to multi-byte characters in  
 37037 arguments).

37038 *LC\_MESSAGES*  
 37039 Determine the locale that should be used to affect the format and contents of  
 37040 diagnostic messages written to standard error.

37041 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

37042 **ASYNCHRONOUS EVENTS**

37043 Default.

37044 **STDOUT**

37045 None.

37046 **STDERR**

37047 Used only for diagnostic messages.

37048 **OUTPUT FILES**

37049           None.

37050 **EXTENDED DESCRIPTION**

37051           None.

37052 **EXIT STATUS**

37053           The following exit values shall be returned:

37054           0   Successful completion.

37055           &gt;0  An error occurred.

37056 **CONSEQUENCES OF ERRORS**

37057           Default.

37058 **APPLICATION USAGE**

37059           None.

37060 **EXAMPLES**

37061           None.

37062 **RATIONALE**

37063           None.

37064 **FUTURE DIRECTIONS**

37065           None.

37066 **SEE ALSO**37067           *link*, *rm*, the System Interfaces volume of IEEE Std. 1003.1-200x, *unlink()*37068 **CHANGE HISTORY**

37069           First released in Issue 5.

## 37070 NAME

37071 uucp — system-to-system copy

## 37072 SYNOPSIS

37073 UN XSI uucp [-cCdfjmr][-n user] source-file... destination-file

37074

## 37075 DESCRIPTION

37076 The *uucp* utility shall copy files named by the *source-file* arguments to the *destination-file*  
37077 argument. The files named can be on local or remote systems.37078 The *uucp* utility cannot guarantee support for all character encodings in all circumstances. For  
37079 example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and  
37080 file names need not be portable to non-internationalized systems, and so on. Under these  
37081 circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991  
37082 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used,  
37083 and that only characters defined in the Portable File Name Character Set be used for naming  
37084 files. The protocol for transfer of files is unspecified by IEEE Std. 1003.1-200x.

## 37085 OPTIONS

37086 The *uucp* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
37087 12.2, Utility Syntax Guidelines.

37088 The following options shall be supported:

37089 **-c** Do not copy local file to the spool directory for transfer to the remote machine  
37090 (default).37091 UN **-C** Force the copy of local files to the spool directory for transfer.37092 **-d** Make all necessary directories for the file copy (default).37093 UN **-f** Do not make intermediate directories for the file copy.37094 UN **-j** Write the job identification string to standard output. This job identification can be  
37095 used by *uustat* to obtain the status or terminate a job.37096 **-m** Send mail to the requester when the copy is completed.37097 UN **-n user** Notify *user* on the remote system that a file was sent.37098 UN **-r** Do not start the file transfer; just queue the job.

## 37099 OPERANDS

37100 The following operands shall be supported:

37101 *destination-file*, *source-file*37102 A path name of a file to be copied to, or from, respectively. Either name can be a  
37103 path name on the local machine, or can have the form:37104 *system-name!pathname*37105 where *system-name* is taken from a list of system names that *uucp* knows about.37106 The destination *system-name* can also be a list of names such as:37107 *system-name!system-name!...!system-name!pathname*37108 in which case, an attempt is made to send the file via the specified route to the  
37109 destination. Care should be taken to ensure that intermediate nodes in the route  
37110 are willing to forward information.

- 37111 The shell pattern matching notation characters '?', '\*', and "[...]" appearing  
37112 in *pathname* are expanded on the appropriate system.
- 37113 Path names can be one of:
- 37114 1. An absolute path name.
- 37115 2. A path name preceded by *~user* where *user* is a login name on the specified  
37116 system and is replaced by that user's login directory. Note that if an invalid  
37117 login is specified, the default is to the public directory (called *PUBDIR*; the  
37118 actual location of *PUBDIR* is implementation-defined).
- 37119 3. A path name preceded by *~/destination* where *destination* is appended to  
37120 *PUBDIR*.
- 37121 **Note:** This destination is treated as a file name unless more than one file  
37122 is being transferred by this request or the destination is already a  
37123 directory. To ensure that it is a directory, follow the destination  
37124 with a '/'. For example, *~/dan/* as the destination makes the  
37125 directory **PUBDIR/dan** if it does not exist and put the requested  
37126 files in that directory.
- 37127 4. Anything else is prefixed by the current directory.
- 37128 If the result is an erroneous path name for the remote system, the copy fails. If the  
37129 *destination-file* is a directory, the last part of the *source-file* name is used.
- 37130 The read, write, and execute permissions given by *uucp* are implementation-  
37131 defined.
- 37132 **STDIN**
- 37133 Not used.
- 37134 **INPUT FILES**
- 37135 The files to be copied are regular files.
- 37136 **ENVIRONMENT VARIABLES**
- 37137 The following environment variables shall affect the execution of *uucp*:
- 37138 *LANG* Provide a default value for the internationalization variables that are unset or null.  
37139 If *LANG* is unset or null, the corresponding value from the implementation-  
37140 defined default locale shall be used. If any of the internationalization variables  
37141 contains an invalid setting, the utility shall behave as if none of the variables had  
37142 been defined.
- 37143 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
37144 internationalization variables.
- 37145 *LC\_COLLATE*
- 37146 Determine the locale for the behavior of ranges, equivalence classes and multi-  
37147 character collating elements within bracketed file name patterns.
- 37148 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
37149 characters (for example, single-byte as opposed to multi-byte characters in  
37150 arguments and input files) and the behavior of character classes within bracketed  
37151 file name patterns (for example, "[[:lower:]]\*").
- 37152 *LC\_MESSAGES*
- 37153 Determine the locale that should be used to affect the format and contents of  
37154 diagnostic messages written to standard error, and informative messages written

- 37155 to standard output.
- 37156 *LC\_TIME* Determine the format of date and time strings output by *uucp*.
- 37157 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 37158 *TZ* Determine the timezone used with date and time strings.
- 37159 **ASYNCHRONOUS EVENTS**
- 37160 Default.
- 37161 **STDOUT**
- 37162 Not used.
- 37163 **STDERR**
- 37164 Used only for diagnostic messages.
- 37165 **OUTPUT FILES**
- 37166 The output files (which may be on other systems) are copies of the input files.
- 37167 If the *-m* is used, mail files are modified.
- 37168 **EXTENDED DESCRIPTION**
- 37169 None.
- 37170 **EXIT STATUS**
- 37171 The following exit values shall be returned:
- 37172 0 Successful completion.
- 37173 >0 An error occurred.
- 37174 **CONSEQUENCES OF ERRORS**
- 37175 Default.
- 37176 **APPLICATION USAGE**
- 37177 The domain of remotely accessible files can (and for obvious security reasons usually should) be severely restricted.
- 37178
- 37179 Note that the *'!*' character in addresses has to be escaped when using *cs**h* as a command interpreter because of its history substitution syntax. For *ksh* and *sh* the escape is not necessary, but may be used.
- 37180
- 37181
- 37182 Typical implementations of this utility require a communications line configured to use the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface, but other communications means may be used. On systems where there are no available communications means (either temporarily or permanently), this utility shall write an error message describing the problem and exit with a non-zero exit status.
- 37183
- 37184
- 37185
- 37186
- 37187 As noted above, shell metacharacters appearing in path names are expanded on the appropriate system. On an internationalized system, this is done under the control of local settings of *LC\_COLLATE* and *LC\_CTYPE*. Thus, care should be taken when using bracketed file name patterns, as collation and typing rules may vary from one system to another. Also be aware that certain types of expression (that is, equivalence classes, character classes, and collating symbols) need not be supported on non-internationalized systems.
- 37188
- 37189
- 37190
- 37191
- 37192
- 37193 **EXAMPLES**
- 37194 None.



37195 **RATIONALE**

37196           None.

37197 **FUTURE DIRECTIONS**

37198           None.

37199 **SEE ALSO**37200           *mailx, uuencode, uustat, uux*37201 **CHANGE HISTORY**

37202           First released in Issue 2.

37203 **Issue 4**

37204           Format reorganized.

37205           Split into a separate description.

37206           Utility Syntax Guidelines support mandated.

37207           Internationalized environment variable support mandated.

37208           Presence of the utility mandated, even on systems where no communications are available.

37209 **NAME**

37210 uudecode — decode a binary file

37211 **SYNOPSIS**37212 UP uudecode [-o *outfile*][*file*]

37213

37214 **DESCRIPTION**

37215 The *uudecode* utility shall read a file, or standard input if no file is specified, that includes data  
 37216 created by the *uuencode* utility. The *uudecode* utility shall scan the input file, searching for data  
 37217 compatible with one of the formats specified in *uuencode* and attempt to create or overwrite the  
 37218 file described by the data (or overridden by the **-o** option). The path name shall be contained in  
 37219 the data or specified by the **-o** option. The file access permission bits and contents for the file to  
 37220 be produced shall be contained in that data. The mode bits of the created file (other than  
 37221 standard output) shall be set from the file access permission bits contained in the data; that is,  
 37222 other attributes of the mode, including the file mode creation mask (see *umask*), shall not affect  
 37223 the file being produced.

37224 If the path name of the file to be produced exists, and the user does not have write permission on  
 37225 that file, *uudecode* shall terminate with an error. If the path name of the file to be produced exists,  
 37226 and the user has write permission on that file, the existing file shall be overwritten.

37227 If the input data was produced by *uuencode* on a system with a different number of bits per byte  
 37228 than on the target system, the results of *uudecode* are unspecified.

37229 **OPTIONS**

37230 The *uudecode* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x,  
 37231 Section 12.2, Utility Syntax Guidelines.

37232 The following option shall be supported by the implementation:

37233 **-o *outfile*** A path name of a file that shall be used instead of any path name contained in the  
 37234 input data. Specifying an *outfile* option-argument of **/dev/stdout** shall indicate  
 37235 standard output.

37236 **OPERANDS**

37237 The following operand shall be supported:

37238 *file* The path name of a file containing the output of *uuencode*.

37239 **STDIN**

37240 See the INPUT FILES section.

37241 **INPUT FILES**

37242 The input files shall be files containing the output of *uuencode*.

37243 **ENVIRONMENT VARIABLES**

37244 The following environment variables shall affect the execution of *uudecode*:

37245 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 37246 If **LANG** is unset or null, the corresponding value from the implementation-  
 37247 defined default locale shall be used. If any of the internationalization variables  
 37248 contains an invalid setting, the utility shall behave as if none of the variables had  
 37249 been defined.

37250 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 37251 internationalization variables.

37252 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 37253 characters (for example, single-byte as opposed to multi-byte characters in

- 37254 arguments and input files).
- 37255 **LC\_MESSAGES**
- 37256 Determine the locale that should be used to affect the format and contents of
- 37257 diagnostic messages written to standard error.
- 37258 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.
- 37259 **ASYNCHRONOUS EVENTS**
- 37260 Default.
- 37261 **STDOUT**
- 37262 If the file data header encoded by *uuencode* is `-` or `/dev/stdout`, or the `-o /dev/stdout` option
- 37263 overrides the file data, the standard output shall be in the same format as the file originally
- 37264 encoded by *uuencode*. Otherwise, the standard output shall not be used.
- 37265 **STDERR**
- 37266 Used only for diagnostic messages.
- 37267 **OUTPUT FILES**
- 37268 The output file shall be in the same format as the file originally encoded by *uuencode*.
- 37269 **EXTENDED DESCRIPTION**
- 37270 None.
- 37271 **EXIT STATUS**
- 37272 The following exit values shall be returned:
- 37273 0 Successful completion.
- 37274 >0 An error occurred.
- 37275 **CONSEQUENCES OF ERRORS**
- 37276 Default.
- 37277 **APPLICATION USAGE**
- 37278 The user who is invoking *uuencode* must have write permission on any file being created.
- 37279 The output of *uuencode* is essentially an encoded bit stream that is not cognizant of byte
- 37280 boundaries. It is possible that a 9-bit byte target machine can process input from an 8-bit source,
- 37281 if it is aware of the requirement, but the reverse is unlikely to be satisfying. Of course, the only
- 37282 data that is meaningful for such a transfer between architectures is generally character data.
- 37283 **EXAMPLES**
- 37284 None.
- 37285 **RATIONALE**
- 37286 Input files are not necessarily text files, as stated by an early proposal. Although the *uuencode*
- 37287 output is a text file, that output could have been wrapped within another file or mail message
- 37288 that is not a text file.
- 37289 The `-o` option is not historical practice, but was added at the request of WG15 so that the user
- 37290 could override the target path name without having to edit the input data itself.
- 37291 In early drafts, the `[-o outfile]` option-argument allowed the use of `-` to mean standard output.
- 37292 The symbol `-` has only been used previously in IEEE Std. 1003.1-200x as a standard input
- 37293 indicator. The developers of the standard did not wish to overload the meaning of `-` in this
- 37294 manner. The `/dev/stdout` concept exists on most modern systems. The `/dev/stdout` syntax does
- 37295 not refer to a new special file. It is just a magic cookie to specify standard output.

37296 **FUTURE DIRECTIONS**

37297           None.

37298 **SEE ALSO**37299           *uuencode*37300 **CHANGE HISTORY**

37301           First released in Issue 4.

37302 **Issue 6**

37303           This utility is now marked as part of the User Portability Utilities option.

37304           The **-o** *outfile* option is added, as specified in the IEEE P1003.2b draft standard.

37305           The normative text is reworded to avoid use of the term “must” for application requirements.

37306 **NAME**

37307 uuencode — encode a binary file

37308 **SYNOPSIS**37309 UP uuencode [-m][*file*] *decode\_pathname*

37310

37311 **DESCRIPTION**

37312 The *uuencode* utility shall write an encoded version of the named input file, or standard input if  
 37313 no *file* is specified, to standard output. The output shall be encoded using one of the algorithms  
 37314 described in the STDOUT section and shall include the file access permission bits (in *chmod* octal  
 37315 or symbolic notation) of the input file and the *decode\_pathname*, for re-creation of the file on  
 37316 another system that conforms to this volume of IEEE Std. 1003.1-200x.

37317 **OPTIONS**

37318 The *uuencode* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x,  
 37319 Section 12.2, Utility Syntax Guidelines.

37320 The following option shall be supported by the implementation:

37321 **-m** Encode the output using the MIME Base64 algorithm described below. If **-m** is not  
 37322 specified, the historical algorithm described in STDOUT shall be used.

37323 **OPERANDS**

37324 The following operands shall be supported:

37325 *decode\_pathname*

37326 The path name of the file into which the *uudecode* utility shall place the decoded  
 37327 file. Specifying a *decode\_pathname* operand of **/dev/stdout** shall indicate that  
 37328 *uudecode* is to use standard output. If there are characters in *decode\_pathname* that  
 37329 are not in the portable file name character set the results are unspecified.

37330 *file* A path name of the file to be encoded.

37331 **STDIN**

37332 See the INPUT FILES section.

37333 **INPUT FILES**

37334 Input files can be files of any type.

37335 **ENVIRONMENT VARIABLES**

37336 The following environment variables shall affect the execution of *uuencode*:

37337 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 37338 If **LANG** is unset or null, the corresponding value from the implementation-  
 37339 defined default locale shall be used. If any of the internationalization variables  
 37340 contains an invalid setting, the utility shall behave as if none of the variables had  
 37341 been defined.

37342 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 37343 internationalization variables.

37344 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 37345 characters (for example, single-byte as opposed to multi-byte characters in  
 37346 arguments and input files).

37347 **LC\_MESSAGES**

37348 Determine the locale that should be used to affect the format and contents of  
 37349 diagnostic messages written to standard error.

37350 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

37351 **ASYNCHRONOUS EVENTS**

37352 Default.

37353 **STDOUT**

37354 **uuencode Base64 Algorithm**

37355 The standard output shall be a text file (encoded in the character set of the current locale) that  
37356 begins with the line:

37357 "begin-base64 $\Delta$ %s $\Delta$ %s\n", <mode>, *decode\_pathname*

37358 and ends with the line:

37359 "====\n"

37360 In both cases, the lines shall have no preceding or trailing <blank>s.

37361 The encoding process represents 24-bit groups of input bits as output strings of four encoded  
37362 characters. Preceding from left to right, a 24-bit input group shall be formed by concatenating  
37363 three 8-bit input groups. These 24-bit then shall be treated as four concatenated 6-bit groups,  
37364 each of which shall be translated into a single digit in the base64 alphabet. When encoding a bit  
37365 stream via the base64 encoding, the bit stream shall be presumed to be ordered with the most-  
37366 significant bit first. That is, the first bit in the stream shall be the high-order bit in the first byte,  
37367 and the eighth bit shall be the low-order bit in the first byte, and so on. Each 6-bit group is used  
37368 as an index into an array of 64 printable characters, as shown in Table 4-21.

**Table 4-21 uuencode Base64 Values**

37369

37370

| Value | Encoding | Value | Encoding | Value | Encoding | Value | Encoding |
|-------|----------|-------|----------|-------|----------|-------|----------|
| 0     | A        | 17    | R        | 34    | i        | 51    | z        |
| 1     | B        | 18    | S        | 35    | j        | 52    | 0        |
| 2     | C        | 19    | T        | 36    | k        | 53    | 1        |
| 3     | D        | 20    | U        | 37    | l        | 54    | 2        |
| 4     | E        | 21    | V        | 38    | m        | 55    | 3        |
| 5     | F        | 22    | W        | 39    | n        | 56    | 4        |
| 6     | G        | 23    | X        | 40    | o        | 57    | 5        |
| 7     | H        | 24    | Y        | 41    | p        | 58    | 6        |
| 8     | I        | 25    | Z        | 42    | q        | 59    | 7        |
| 9     | J        | 26    | a        | 43    | r        | 60    | 8        |
| 10    | K        | 27    | b        | 44    | s        | 61    | 9        |
| 11    | L        | 28    | c        | 45    | t        | 62    | +        |
| 12    | M        | 29    | d        | 46    | u        | 63    | /        |
| 13    | N        | 30    | e        | 47    | v        | (pad) | =        |
| 14    | O        | 31    | f        | 48    | w        |       |          |
| 15    | P        | 32    | g        | 49    | x        |       |          |
| 16    | Q        | 33    | h        | 50    | y        |       |          |

37388 The character referenced by the index shall be placed in the output string.

37389 The output stream (encoded bytes) shall be represented in lines of no more than 76 characters  
37390 each. All line breaks or other characters not found in the table shall be ignored by decoding  
37391 software (see *uudecode*).

37392 Special processing shall be performed if fewer than 24 bits are available at the end of a message  
37393 or encapsulated part of a message. A full encoding quantum shall always be completed at the

37394 end of a message. When fewer than 24 input bits are available in an input group, zero bits shall  
 37395 be added (on the right) to form an integral number of 6-bit groups. Output character positions  
 37396 that are not required to represent actual input data shall be set to the character '='. Since all  
 37397 base64 input is an integral number of octets, only the following cases can arise:

- 37398 1. The final quantum of encoding input is an integral multiple of 24 bits; here, the final unit of  
 37399 encoded output shall be an integral multiple of 4 characters with no '=' padding.
- 37400 2. The final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output  
 37401 shall be two characters followed by two '=' padding characters.
- 37402 3. The final quantum of encoding input is exactly 16 bits; here, the final unit of encoded  
 37403 output shall be three characters followed by one '=' padding character.
- 37404 4. The terminating "====" evaluates to nothing and denotes the end of the encoded data.

#### 37405 **uuencode Historical Algorithm**

37406 The standard output shall be a text file (encoded in the character set of the current locale) that  
 37407 begins with the line:

```
37408 "beginΔ%sΔ%s\n" <mode>, <decode_pathname>
```

37409 and ends with the line:

```
37410 end\n
```

37411 In both cases, the lines shall have no preceding or trailing <blank> characters.

37412 The algorithm that shall be used for lines in between **begin** and **end** takes three octets as input  
 37413 and writes four characters of output by splitting the input at six-bit intervals into four octets,  
 37414 containing data in the lower six bits only. These octets shall be converted to characters by adding  
 37415 a value of 0x20 to each octet, so that each octet is in the range 0x20-0x5f, and then it shall be  
 37416 assumed to represent a printable character in the ISO/IEC 646: 1991 standard encoded character  
 37417 set. It then shall be translated into the corresponding character codes for the codeset in use in the  
 37418 current locale. (For example, the octet 0x41, representing 'A', would be translated to 'A' in the  
 37419 current codeset, such as 0xc1 if it were EBCDIC.)

37420 Where the bits of two octets are combined, the least significant bits of the first octet shall be  
 37421 shifted left and combined with the most significant bits of the second octet shifted right. Thus  
 37422 the three octets *A*, *B*, *C* shall be converted into the four octets:

```
37423 0x20 + ((A >> 2) & 0x3F)
37424 0x20 + (((A << 4) | ((B >> 4) & 0xF)) & 0x3F)
37425 0x20 + (((B << 2) | ((C >> 6) & 0x3)) & 0x3F)
37426 0x20 + ((C) & 0x3F)
```

37427 These octets then shall be translated into the local character set.

37428 Each encoded line contains a length character, equal to the number of characters to be decoded  
 37429 plus 0x20 translated to the local character set as described above, followed by the encoded  
 37430 characters. The maximum number of octets to be encoded on each line shall be 45.

#### 37431 **STDERR**

37432 Used only for diagnostic messages.

#### 37433 **OUTPUT FILES**

37434 None.

37435 **EXTENDED DESCRIPTION**

37436 None.

37437 **EXIT STATUS**

37438 The following exit values shall be returned:

37439 0 Successful completion.

37440 &gt;0 An error occurred.

37441 **CONSEQUENCES OF ERRORS**

37442 Default.

37443 **APPLICATION USAGE**

37444 The file is expanded by 35 percent (each three octets become four, plus control information)  
37445 causing it to take longer to transmit.

37446 Since this utility is intended to create files to be used for data interchange between systems with  
37447 possibly different codesets, and to represent binary data as a text file, the ISO/IEC 646:1991  
37448 standard was chosen for a midpoint in the algorithm as a known reference point. The output  
37449 from *uuencode* is a text file on the local system. If the output were in the ISO/IEC 646:1991  
37450 standard codeset, it might not be a text file (at least because the <newline> characters might not  
37451 match), and the goal of creating a text file would be defeated. If this text file was then carried to  
37452 another machine with the same codeset, it would be perfectly compatible with that system's  
37453 *uudecode*. If it was transmitted over a mail system or sent to a machine with a different codeset,  
37454 it is assumed that, as for every other text file, some translation mechanism would convert it (by  
37455 the time it reached a user on the other system) into an appropriate codeset. This translation only  
37456 makes sense from the local codeset, not if the file has been put into a ISO/IEC 646:1991 standard  
37457 representation first. Similarly, files processed by *uuencode* can be placed in *pax* archives,  
37458 intermixed with other text files in the same codeset.

37459 The algorithm is described in terms of 8-bit quantities, or octets. Since no byte alignment is  
37460 implied, it encodes data from machines with any number of bits per byte. However, unless that  
37461 encoded data is then decoded on a machine with the same number of bits per byte, the output  
37462 might not be useful.

37463 **EXAMPLES**

37464 None.

37465 **RATIONALE**

37466 A new algorithm was added at the request of the international community to parallel work in  
37467 RFC 2045 (MIME). As with the historical *uuencode* format, the Base64 Content-Transfer-Encoding  
37468 is designed to represent arbitrary sequences of octets in a form that is not humanly readable. A  
37469 65-character subset of the ISO/IEC 646:1991 standard is used, enabling 6 bits to be represented  
37470 per printable character. (The extra 65th character, '=', is used to signify a special processing  
37471 function.)

37472 This subset has the important property that it is represented identically in all versions of the  
37473 ISO/IEC 646:1991 standard, including US ASCII, and all characters in the subset are also  
37474 represented identically in all versions of EBCDIC. The historical *uuencode* algorithm does not  
37475 share this property, which is the reason that a second algorithm was added to the ISO POSIX-2  
37476 standard.

37477 The string "====" was used for the termination instead of the end used in the original format  
37478 because the latter is a string that could be valid encoded input.

37479 In an early draft, the `-m` option was named `-b` (for Base64), but it was renamed to reflect its  
37480 relationship to the RFC 2045. A `-u` was also present to invoke the default algorithm, but since



- 37481 this was not historical practice, it was omitted as being unnecessary.
- 37482 See the RATIONALE section in *uudecode* for the derivation of the `/dev/stdout` symbol.
- 37483 **FUTURE DIRECTIONS**
- 37484 None.
- 37485 **SEE ALSO**
- 37486 *mailx, uudecode*
- 37487 **CHANGE HISTORY**
- 37488 First released in Issue 4.
- 37489 **Issue 6**
- 37490 This utility is now marked as part of the User Portability Utilities option.
- 37491 The Base64 algorithm and the ability to output to `/dev/stdout` are added as specified in the
- 37492 IEEE P1003.2b draft standard.

## 37493 NAME

37494 uustat — uucp status inquiry and job control

## 37495 SYNOPSIS

37496 UN XSI uustat [ -q | -k *jobid* | -r *jobid* ]37497 XSI uustat [-s *system*][-u *user*]

## 37498 DESCRIPTION

37499 The *uustat* utility shall display the status of, or cancel, previously specified *uucp* requests, or  
37500 provide general status on *uucp* connections to other systems.37501 When no options are given, *uustat* shall write to standard output the status of all *uucp* requests  
37502 issued by the current user.37503 Typical implementations of this utility require a communications line configured to use the Base  
37504 Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface, but other  
37505 communications means may be used. On systems where there are no available communications  
37506 means (either temporarily or permanently), this utility shall write an error message describing  
37507 the problem and exits with a non-zero exit status.

## 37508 OPTIONS

37509 The *uustat* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
37510 12.2, Utility Syntax Guidelines.

37511 The following options shall be supported:

37512 UN -q Write the jobs queued for each machine.

37513 -k *jobid* Kill the *uucp* request whose job identification is *jobid*. The application shall ensure  
37514 that the killed *uucp* request belongs to the person invoking *uustat* unless that user  
37515 has appropriate privileges.37516 -r *jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their  
37517 modification time is set to the current time. This prevents the cleanup program  
37518 from deleting the job until the jobs modification time reaches the limit imposed by  
37519 the program.37520 -s *system* Write the status of all *uucp* requests for remote system *system*.37521 -u *user* Write the status of all *uucp* requests issued by *user*.

## 37522 OPERANDS

37523 None.

## 37524 STDIN

37525 Not used.

## 37526 INPUT FILES

37527 None.

## 37528 ENVIRONMENT VARIABLES

37529 The following environment variables shall affect the execution of *uustat*:37530 *LANG* Provide a default value for the internationalization variables that are unset or null.  
37531 If *LANG* is unset or null, the corresponding value from the implementation-  
37532 defined default locale shall be used. If any of the internationalization variables  
37533 contains an invalid setting, the utility shall behave as if none of the variables had  
37534 been defined.

- 37535        *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
37536                    internationalization variables.
- 37537        *LC\_CTYPE*       Determine the locale for the interpretation of sequences of bytes of text data as  
37538                    characters (for example, single-byte as opposed to multi-byte characters in  
37539                    arguments).
- 37540        *LC\_MESSAGES*  
37541                    Determine the locale that should be used to affect the format and contents of  
37542                    diagnostic messages written to standard error, and informative messages written  
37543                    to standard output.
- 37544        *LC\_TIME*        Determine the format of date and time strings output by *uustat*.
- 37545        *NLSPATH*       Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 37546        *TZ*             Determine the timezone used with date and time strings.
- 37547 **ASYNCHRONOUS EVENTS**  
37548        Default.
- 37549 **STDOUT**  
37550        The standard output shall consist of information about each job selected, in an unspecified  
37551        format. The information shall include at least the job ID, the user ID or name, and the remote  
37552        system name.
- 37553 **STDERR**  
37554        Used only for diagnostic messages.
- 37555 **OUTPUT FILES**  
37556        None.
- 37557 **EXTENDED DESCRIPTION**  
37558        None.
- 37559 **EXIT STATUS**  
37560        The following exit values shall be returned:  
37561            0    Successful completion.  
37562            >0   An error occurred.
- 37563 **CONSEQUENCES OF ERRORS**  
37564        Default.
- 37565 **APPLICATION USAGE**  
37566        None.
- 37567 **EXAMPLES**  
37568        None.
- 37569 **RATIONALE**  
37570        None.
- 37571 **FUTURE DIRECTIONS**  
37572        None.
- 37573 **SEE ALSO**  
37574        *uucp*

37575 **CHANGE HISTORY**

37576 First released in Issue 2.

37577 **Issue 4**

37578 Format reorganized.

37579 Utility Syntax Guidelines support mandated.

37580 Internationalized environment variable support mandated.

37581 Presence of the utility mandated, even on systems where no communications are available.

37582 **Issue 6**

37583 The normative text is reworded to avoid use of the term “must” for application requirements.

37584 **NAME**

37585 uux — remote command execution

37586 **SYNOPSIS**37587 XSI uux [-np] *command-string*37588 UN XSI uux [-jnp] *command-string*37589 **DESCRIPTION**

37590 The *uux* utility shall gather zero or more files from various systems, execute a shell pipeline (see  
 37591 Section 2.9 (on page 2256)) on a specified system, and then send the standard output of the  
 37592 command to a file on a specified system. Only the first command of a pipeline can have a  
 37593 *system-name!* prefix. All other commands in the pipeline shall be executed on the system of the  
 37594 first command.

37595 The following restrictions are applicable to the shell pipeline processed by *uux*:

- 37596 • In gathering files from different systems, path name expansion is not performed by *uux*.  
 37597 Thus, a request such as:

```
37598 uux "c99 remsys!~/*.c"
```

37599 would attempt to copy the file named literally \*.c to the local system.

- 37600 • The redirection operators ">>", "<<", ">|", and ">&" shall not be accepted. Any use of  
 37601 these redirection operators shall cause this utility to write an error message describing the  
 37602 problem and exit with a non-zero exit status.
- 37603 • The reserved word ! cannot be used at the head of the pipeline to modify the exit status.
- 37604 • Alias substitution is not performed.

37605 A file name can be specified as for *uucp*; it can be an absolute path name, a path name preceded  
 37606 by *~name* (which is replaced by the corresponding login directory), a path name specified as  
 37607 *~/dest* (*dest* is prefixed by the public directory called *PUBDIR*; the actual location of *PUBDIR* is  
 37608 implementation-defined), or a simple file name (which is prefixed by *uux* with the current  
 37609 directory). See *uucp* (on page 3178) for the details.

37610 The execution of commands on remote systems shall take place in an execution directory known  
 37611 to the *uucp* system. All files required for the execution shall be put into this directory unless they  
 37612 already reside on that machine. Therefore, the application shall ensure that non-local file names  
 37613 (without path or machine reference) are unique within the *uux* request.

37614 The *uux* utility shall attempt to get all files to the execution system. For files that are output files,  
 37615 the application shall ensure that the file name is escaped using parentheses.

37616 The remote system shall notify the user by mail if the requested command on the remote system  
 37617 was disallowed or the files were not accessible. This notification can be turned off by the *-n*  
 37618 option.

37619 Typical implementations of this utility require a communications line configured to use the Base  
 37620 Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General Terminal Interface, but other  
 37621 communications means may be used. On systems where there are no available communications  
 37622 means (either temporarily or permanently), this utility shall write an error message describing  
 37623 the problem and exits with a non-zero exit status.

37624 The *uux* utility cannot guarantee support for all character encodings in all circumstances. For  
 37625 example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and  
 37626 file names need not be portable to non-internationalized systems, and so on. Under these  
 37627 circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991

37628 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used  
 37629 and that only characters defined in the Portable File Name Character Set be used for naming  
 37630 files.

### 37631 OPTIONS

37632 The *uux* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 37633 12.2, Utility Syntax Guidelines.

37634 The following options shall be supported:

37635 **-p** Make the standard input to *uux* the standard input to the *command-string*.

37636 UN **-j** Write the job identification string to standard output. This job identification can be  
 37637 used by *uustat* to obtain the status or terminate a job.

37638 **-n** Do not notify the user if the command fails.

### 37639 OPERANDS

37640 The following operand shall be supported:

37641 *command-string*

37642 A string made up of one or more arguments that are similar to normal command  
 37643 arguments, except that the command and any file names can be prefixed by  
 37644 *system-name!*. A null *system-name* shall be interpreted as the local system.

### 37645 STDIN

37646 The standard input shall not be used unless the *'-'* or **-p** option is specified; in those cases, the  
 37647 standard input shall be made the standard input of the *command-string*.

### 37648 INPUT FILES

37649 Input files shall be selected according to the contents of *command-string*.

### 37650 ENVIRONMENT VARIABLES

37651 The following environment variables shall affect the execution of *uux*:

37652 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 37653 If *LANG* is unset or null, the corresponding value from the implementation-  
 37654 defined default locale shall be used. If any of the internationalization variables  
 37655 contains an invalid setting, the utility shall behave as if none of the variables had  
 37656 been defined.

37657 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 37658 internationalization variables.

37659 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 37660 characters (for example, single-byte as opposed to multi-byte characters in  
 37661 arguments).

37662 *LC\_MESSAGES*

37663 Determine the locale that should be used to affect the format and contents of  
 37664 diagnostic messages written to standard error.

37665 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

### 37666 ASYNCHRONOUS EVENTS

37667 Default.

37668 **STDOUT**

37669 The standard output shall be not used unless the **-j** option is specified; in that case, the job  
 37670 identification string shall be written to standard output in the following format:

37671 "%s\n", <jobid>

37672 **STDERR**

37673 Used only for diagnostic messages.

37674 **OUTPUT FILES**

37675 Output files shall be created or written, or both, according to the contents of *command-string*.

37676 If the **-n** is not used, mail files shall be modified following any command or file-access failures  
 37677 on the remote system.

37678 **EXTENDED DESCRIPTION**

37679 None.

37680 **EXIT STATUS**

37681 The following exit values shall be returned:

37682 0 Successful completion.

37683 >0 An error occurred.

37684 **CONSEQUENCES OF ERRORS**

37685 Default.

37686 **APPLICATION USAGE**

37687 Note that, for security reasons, many installations limit the list of commands executable on  
 37688 behalf of an incoming request from *uux*. Many sites permit little more than the receipt of mail  
 37689 via *uux*.

37690 Any characters special to the command interpreter should be quoted either by quoting the entire  
 37691 *command-string* or quoting the special characters as individual arguments.

37692 As noted in *uucp*, shell pattern matching notation characters appearing in path names are  
 37693 expanded on the appropriate local system. This is done under the control of local settings of  
 37694 *LC\_COLLATE* and *LC\_CTYPE*. Thus, care should be taken when using bracketed file name  
 37695 patterns, as collation and typing rules may vary from one system to another. Also be aware that  
 37696 certain types of expression (that is, equivalence classes, character classes, and collating symbols)  
 37697 need not be supported on non-internationalized systems.

37698 **EXAMPLES**

37699 1. The following command gets **file1** from system **a** and **file2** file from system **b**, executes *diff*  
 37700 on the local system, and puts the results in **file.diff** in the local *PUBDIR* directory.  
 37701 (*PUBDIR* is the *uucp* public directory on the local system.)

37702 uux "!diff a!/usr/file1 b!/a4/file2 >!~/file.diff"

37703 2. The following command fails because *uux* places all files copied to a system in the same  
 37704 working directory. Although the files **xyz** are from two different systems, their file names  
 37705 are the same and conflict.

37706 uux "!diff a!/usr1/xyz b!/usr2/xyz >!~/xyz.diff"

37707 3. The following command succeeds (assuming *diff* is permitted on system **a**) because the file  
 37708 local to system **a** is not copied to the working directory, and hence does not conflict the file  
 37709 from system **c**.

37710 uux "a!diff a!/usr/xyz c!/usr/xyz >!~/xyz.diff"

37711 **RATIONALE**

37712 None.

37713 **FUTURE DIRECTIONS**

37714 A version of *uux* that fully supports the Base Definitions volume of IEEE Std. 1003.1-200x,  
37715 Section 12.2, Utility Syntax Guidelines may be introduced in a future issue.

37716 **SEE ALSO**

37717 *uucp, uuencode, uustat*

37718 **CHANGE HISTORY**

37719 First released in Issue 2.

37720 **Issue 4**

37721 Format reorganized.

37722 Exceptions to Utility Syntax Guidelines conformance noted.

37723 Internationalized environment variable support mandated.

37724 Presence of the utility mandated, even on systems where no communications are available.

37725 **Issue 6**

37726 The obsolescent SYNOPSIS is removed.

37727 The normative text is reworded to avoid use of the term “must” for application requirements.



37728 **NAME**37729 val — validate SCCS files (**DEVELOPMENT**)37730 **SYNOPSIS**

37731 xSI val -

37732 val [-s][*-m name*][*-r SID*][*-y type*] *file...*

37733

37734 **DESCRIPTION**37735 The *val* utility shall determine whether the specified *file* is an SCCS file meeting the  
37736 characteristics specified by the options.37737 **OPTIONS**37738 The *val* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
37739 12.2, Utility Syntax Guidelines, except that the usage of the '-' operand is not strictly as  
37740 intended by the guidelines (that is, reading options and operands from standard input).

37741 The following options shall be supported:

37742 *-m name* Specify a *name*, which is compared with the SCCS %M% keyword in *file*; see *get*  
37743 (on page 2685).37744 *-r SID* Specify a *SID* (SCCS Identification String), an SCCS delta number. A check shall be  
37745 made to determine whether the *SID* is ambiguous (for example, *-r 1* is ambiguous  
37746 because it physically does not exist but implies 1.1, 1.2, and so on, which may  
37747 exist) or invalid (for example, *-r 1.0* or *-r 1.1.0* are invalid because neither case can  
37748 exist as a valid delta number). If the *SID* is valid and not ambiguous, a check shall  
37749 be made to determine whether it actually exists.37750 *-s* Silence the diagnostic message normally written to standard output for any error  
37751 that is detected while processing each named file on a given command line.37752 *-y type* Specify a *type*, which shall be compared with the SCCS %Y% keyword in *file*; see  
37753 *get* (on page 2685).37754 **OPERANDS**

37755 The following operands shall be supported:

37756 *file* A path name of an existing SCCS file. If exactly one *file* operand appears, and it is  
37757 '-', the standard input shall be read: each line is independently processed as if it  
37758 were a command line argument list. (However, the line is not subjected to any of  
37759 the shell word expansions, such as parameter expansion or quote removal.)37760 **STDIN**37761 The standard input shall be a text file used only when the *file* operand is specified as '-'.37762 **INPUT FILES**

37763 Any SCCS files processed are files of an unspecified format.

37764 **ENVIRONMENT VARIABLES**37765 The following environment variables shall affect the execution of *val*:37766 *LANG* Provide a default value for the internationalization variables that are unset or null.  
37767 If *LANG* is unset or null, the corresponding value from the implementation-  
37768 defined default locale shall be used. If any of the internationalization variables  
37769 contains an invalid setting, the utility shall behave as if none of the variables had  
37770 been defined.

37771 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
37772 internationalization variables.

37773 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
37774 characters (for example, single-byte as opposed to multi-byte characters in  
37775 arguments and input files).

37776 *LC\_MESSAGES*  
37777 Determine the locale that should be used to affect the format and contents of  
37778 diagnostic messages written to standard error, and informative messages written  
37779 to standard output.

37780 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

37781 **ASYNCHRONOUS EVENTS**  
37782 Default.

37783 **STDOUT**  
37784 The standard output shall consist of informative messages about either:  
37785 1. Each file processed  
37786 2. Each command line read from standard input

37787 If the standard input is not used, for each *file* operand yielding a discrepancy, the output line  
37788 shall have the following format:  
37789 "%s: %s\n", <pathname>, <unspecified string>

37790 If standard input is used, a line of input shall be written before each of the preceding lines for  
37791 files containing discrepancies:  
37792 "%s:\n", <input line>

37793 **STDERR**  
37794 Not used.

37795 **OUTPUT FILES**  
37796 None.

37797 **EXTENDED DESCRIPTION**  
37798 None.

37799 **EXIT STATUS**  
37800 The 8-bit code returned by *val* is a disjunction of the possible errors, that is, it can be interpreted  
37801 as a bit string where set bits are interpreted as follows:

37802 0x80 = Missing file argument.  
37803 0x40 = Unknown or duplicate option.  
37804 0x20 = Corrupted SCCS file.  
37805 0x10 = Cannot open file or file not SCCS.  
37806 0x08 = *SID* is invalid or ambiguous.  
37807 0x04 = *SID* does not exist.  
37808 0x02 = %Y%, -y mismatch.  
37809 0x01 = %M%, -m mismatch.

37810 Note that *val* can process two or more files on a given command line and can process multiple  
37811 command lines (when reading the standard input). In these cases an aggregate code shall be  
37812 returned: a logical OR of the codes generated for each command line and file processed.

37813 **CONSEQUENCES OF ERRORS**

37814 Default.

37815 **APPLICATION USAGE**

37816 Since the *val* exit status sets the 0x80 bit, shell applications checking "\$?" cannot tell if it  
37817 terminated due to a missing file argument or receipt of a signal.

37818 **EXAMPLES**

37819 In a directory with three SCCS files, *s.x* (of *t* type "text"), *s.y*, and *s.z* (a corrupted file), the  
37820 following command could produce the output shown:

37821 `val - <<EOF`37822 `-y source s.x`37823 `-m y s.y`37824 `s.z`37825 `EOF`37826 `-y source s.x`37827 `s.x: %Y%, -y mismatch`37828 `s.z`37829 `s.z: corrupted SCCS file`37830 **RATIONALE**

37831 None.

37832 **FUTURE DIRECTIONS**

37833 None.

37834 **SEE ALSO**37835 *admin, delta, get, prs*37836 **CHANGE HISTORY**

37837 First released in Issue 2.

37838 **Issue 4**

37839 Format reorganized.

37840 Exceptions to Utility Syntax Guidelines conformance noted.

37841 Internationalized environment variable support mandated.

37842 **Issue 6**37843 The Open Group corrigenda item U025/4 has been applied, correcting a typographical error in  
37844 the EXIT STATUS.

37845 The normative text is reworded to emphasise the term "shall" for implementation requirements.

37846 **NAME**

37847 vi — screen-oriented (visual) display editor

37848 **SYNOPSIS**37849 UP `vi [-rR][-l][-c command][-t tagstring][-w size][file ...]`

37850

37851 **DESCRIPTION**37852 This utility shall be provided on systems that both support the User Portability Utilities option  
37853 and define the POSIX2\_CHAR\_TERM symbol. On other systems it is optional.37854 The *vi* (visual) utility is a screen-oriented text editor. Only the open and visual modes of the  
37855 editor are described in IEEE Std. 1003.1-200x; see the line editor *ex* for additional editing  
37856 capabilities used in *vi*. The user can switch back and forth between *vi* and *ex* and execute *ex*  
37857 commands from within *vi*.37858 This reference page uses the term *edit buffer* to describe the current working text. No specific  
37859 implementation is implied by this term. All editing changes are performed on the edit buffer,  
37860 and no changes to it shall affect any file until an editor command writes the file.37861 When using *vi*, the terminal screen acts as a window into the editing buffer. Changes made to  
37862 the editing buffer shall be reflected in the screen display; the position of the cursor on the screen  
37863 shall indicate the position within the editing buffer.37864 Certain terminals do not have all the capabilities necessary to support the complete *vi* definition.  
37865 When these commands cannot be supported on such terminals, this condition shall not produce  
37866 an error message such as “not an editor command” or report a syntax error. The implementation  
37867 may either accept the commands and produce results on the screen that are the result of an  
37868 unsuccessful attempt to meet the requirements of this volume of IEEE Std. 1003.1-200x or report  
37869 an error describing the terminal-related deficiency.37870 **OPTIONS**37871 The *vi* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2,  
37872 Utility Syntax Guidelines.

37873 The following options shall be supported:

37874 `-c command` See the *ex* command description of the `-c` option.37875 `-l` (The letter ell.) Set lisp mode; see **Edit Options in *ex*** (on page 2602).37876 `-r` See the *ex* command description of the `-r` option.37877 `-R` See the *ex* command description of the `-R` option.37878 `-t tagstring` See the *ex* command description of the `-t` option.37879 `-w size` See the *ex* command description of the `-w` option.37880 **OPERANDS**37881 See the OPERANDS section of the *ex* command for a description of the operands supported by  
37882 the *vi* command.37883 **STDIN**37884 If standard input is not a terminal device, the results are undefined. The standard input consists  
37885 of a series of commands and input text, as described in the EXTENDED DESCRIPTION section.37886 If a read from the standard input returns an error, or if the editor detects an end-of-file condition  
37887 from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

37888 **INPUT FILES**

37889 See the INPUT FILES section of the *ex* command for a description of the input files supported by  
37890 the *vi* command.

37891 **ENVIRONMENT VARIABLES**

37892 See the ENVIRONMENT VARIABLES section of the *ex* command for the environment variables  
37893 that affect the execution of the *vi* command.

37894 **ASYNCHRONOUS EVENTS**

37895 See the ASYNCHRONOUS EVENTS section of the *ex* for the asynchronous events that affect the  
37896 execution of the *vi* command.

37897 **STDOUT**

37898 If standard output is not a terminal device, undefined results occur.

37899 Standard output may be used for writing prompts to the user, for informational messages, and  
37900 for writing lines from the file.

37901 **STDERR**

37902 If standard output is not a terminal device, undefined results occur.

37903 Used only for diagnostic messages.

37904 **OUTPUT FILES**

37905 See the OUTPUT FILES section of the *ex* command for a description of the output files  
37906 supported by the *vi* command.

37907 **EXTENDED DESCRIPTION**

37908 If the terminal does not have the capabilities necessary to support an unspecified portion of the  
37909 *vi* definition, implementations shall start initially in *ex* mode or open mode. Otherwise, after  
37910 initialization, *vi* shall be in command mode; text input mode can be entered by one of several  
37911 commands used to insert or change text. In text input mode, <ESC> can be used to return to  
37912 command mode; other uses of <ESC> are described later in this section; see **Terminate**  
37913 **Command or Input Mode** (on page 3209).

37914 **Initialization in *ex* and *vi***

37915 See **Initialization in *ex* and *vi*** (on page 2569) for a description of *ex* and *vi* initialization for the *vi*  
37916 utility.

37917 **Command Descriptions in *vi***

37918 The following symbols are used in this reference page to represent arguments to commands.

37919 *buffer* See the description of *buffer* in the EXTENDED DESCRIPTION section of the *ex* utility;  
37920 see **Command Descriptions in *ex*** (on page 2578).

37921 In open and visual mode, when a command synopsis shows both [*buffer*] and [*count*]  
37922 preceding the command name, they can be specified in either order.

37923 *count* A positive integer used as an optional argument to most commands, either to give a  
37924 repeat count or as a size. This argument is optional and shall default to 1 unless  
37925 otherwise specified.

37926 The Synopsis lines for the *vi* commands <control>-G, <control>-L, <control>-R,  
37927 <control>-], %, &, ^, D, m, M, Q, u, U, and ZZ do not have *count* as an optional  
37928 argument. Regardless, it shall not be an error to specify a *count* to these commands, and  
37929 any specified *count* shall be ignored.

37930 *motion* An optional trailing argument used by the **!**, **<**, **>**, **c**, **d**, and **y** commands, which is used  
 37931 to indicate the region of text that shall be affected by the command. The motion can be  
 37932 either one of the command characters repeated or one of several other *vi* commands  
 37933 (listed in the following table). Each of the applicable commands specifies the region of  
 37934 text matched by repeating the command; each command that can be used as a motion  
 37935 command specifies the region of text it affects.

37936 Commands that take *motion* arguments operate on either lines or characters, depending  
 37937 on the circumstances. When operating on lines, all lines that fall partially or wholly  
 37938 within the text region specified for the command shall be affected. When operating on  
 37939 characters, only the exact characters in the specified text region shall be affected. Each  
 37940 motion command specifies this individually.

37941 When commands that may be motion commands are not used as motion commands,  
 37942 they shall set the current position to the current line and column as specified.

37943 The following commands shall be valid cursor motion commands:

|                         |           |            |
|-------------------------|-----------|------------|
| 37944 <control>-H       | ;         | 'character |
| 37945 <newline>         | ?         | <b>b</b>   |
| 37946 <carriage-return> | <b>B</b>  | <b>e</b>   |
| 37947 <control>-N       | <b>E</b>  | <b>f</b>   |
| 37948 <control>-P       | <b>F</b>  | <b>h</b>   |
| 37949 <space>           | <b>G</b>  | <b>j</b>   |
| 37950 \$                | <b>H</b>  | <b>k</b>   |
| 37951 %                 | <b>L</b>  | <b>l</b>   |
| 37952 'character        | <b>M</b>  | <b>n</b>   |
| 37953 (                 | <b>N</b>  | <b>t</b>   |
| 37954 )                 | <b>T</b>  | <b>w</b>   |
| 37955 +                 | <b>W</b>  | <b>{</b>   |
| 37956 ,                 | <b>[[</b> | <b> </b>   |
| 37957 -                 | <b>]]</b> | <b>}</b>   |
| 37958 /                 | <b>^</b>  | <b>0</b>   |
| 37959 _                 |           |            |

37960 Any *count* that is specified to a command that has an associated motion command shall  
 37961 be applied to the motion command. If a *count* is applied to both the command and its  
 37962 associated motion command, the effect shall be multiplicative.

37963 The following symbol is used in this section to specify locations in the edit buffer:

37964 *current character*

37965 The character that is currently displayed by the cursor.

37966 The following symbols are used in this section to specify command actions:

37967 *bigword* In the POSIX locale, *vi* shall recognize four kinds of *bigwords*:

- 37968 1. A maximal sequence of non-<blank> characters preceded and followed by  
 37969 <blank> characters or the beginning or end of a line or the edit buffer
- 37970 2. One or more sequential empty or <blank> character-filled lines
- 37971 3. The first character in the edit buffer
- 37972 4. The last character in the edit buffer

37973 *word* In the POSIX locale, *vi* shall recognize five kinds of words:

- 37974 1. A maximal sequence of letters, digits, and underscores, delimited at both ends by:
- 37975 — Characters other than letters, digits, or underscores
- 37976 — The beginning or end of a line
- 37977 — The beginning or end of the edit buffer
- 37978 2. A maximal sequence of characters other than letters, digits, underscores, or
- 37979 <blank> characters, delimited at both ends by:
- 37980 — A letter, digit, underscore
- 37981 — <blank> characters
- 37982 — The beginning or end of a line
- 37983 — The beginning or end of the edit buffer
- 37984 3. One or more sequential empty or <blank> character-filled lines
- 37985 4. The first character in the edit buffer
- 37986 5. The last character in the edit buffer

37987 *section boundary*

37988 A *section boundary* is one of the following:

- 37989 1. A line whose first character is a <form-feed> character
- 37990 2. A line whose first character is an open curly brace ( ' { ' )
- 37991 3. A line whose first character is a period and whose second and third characters
- 37992 match a two-character pair in the **sections** edit option (see *ed*)
- 37993 4. A line whose first character is a period and whose only other character matches
- 37994 the first character of a two-character pair in the **sections** edit option, where the
- 37995 second character of the two-character pair is a <space> character
- 37996 5. The first line of the edit buffer
- 37997 6. The last line of the edit buffer if the last line of the edit buffer is empty or if it is a
- 37998 **]]** or **}** command; otherwise, the last character of the last line of the edit buffer

37999 *paragraph boundary*

38000 A *paragraph boundary* is one of the following:

- 38001 1. A section boundary
- 38002 2. A line whose first character is a period and whose second and third characters
- 38003 match a two-character pair in the **paragraphs** edit option (see *ed*)
- 38004 3. A line whose first character is a period and whose only other character matches
- 38005 the first character of a two-character pair in the *paragraphs* edit option, where the
- 38006 second character of the two-character pair is a <space> character
- 38007 4. One or more sequential empty or <blank> character-filled lines

38008 *remembered search direction*

38009 See the description of remembered search direction in *ed*.

38010 *sentence boundary*

38011 A *sentence boundary* is one of the following:

- 38012 1. A paragraph boundary
- 38013 2. The first non-<blank> character that occurs after a paragraph boundary
- 38014 3. The first non-<blank> character that occurs after a period ( '.' ), exclamation  
38015 mark ( ' ! ' ), or question mark ( ' ? ' ), followed by two <space> characters or the  
38016 end of a line; any number of closing parenthesis ( ' ) ' ), closing brackets ( ' ] ' ),  
38017 double quote ( ' " ' ), or single quote ( ' \ ' ' ) characters can appear between the  
38018 punctuation mark and the two <space> characters or end-of-line

38019 Any lines displayed on the screen that logically represent lines after the last line in the edit buffer  
38020 shall consist of a single tilde ( '~ ' ) character.

38021 The last line of the screen shall be used to report errors or display informational messages. It  
38022 shall also be used to display the input for “line-oriented commands” ( / , ? , : , and ! ). When a line-  
38023 oriented command is executed, the editor shall enter text input mode on the last line on the  
38024 screen, using the respective command characters as prompt characters. (In the case of the !  
38025 command, the associated motion shall be entered by the user before the editor enters text input  
38026 mode.) The line entered by the user shall be terminated by a character, a non-<control>-V-  
38027 escaped <carriage-return> character, or unescaped <ESC>. It is unspecified if more characters  
38028 than require a display width minus one column number of screen columns can be entered.

38029 If any command is executed that overwrites a portion of the screen other than the last line of the  
38030 screen (for example, the *ex suspend*, or ! commands), other than the *ex shell* command, the user  
38031 shall be prompted for a character before the screen is refreshed and the edit session continued.

38032 <tab> characters shall take up the number of columns on the screen set by the **tabstop** edit  
38033 option (see *ed*), unless there are less than that number of columns before the display margin that  
38034 will cause the displayed line to be folded; in this case, they shall only take up the number of  
38035 columns up to that boundary.

38036 The cursor shall be placed on the current line and relative to the current column as specified by  
38037 each command described in the following sections.

38038 In open mode, if the current line is not already displayed, then it shall be displayed.

38039 In the remainder of the description of the *vi* utility, the term “physical line” refers to a line in the  
38040 edit buffer and the term “logical line” refers to the line or lines on the display screen used to  
38041 display a physical line.

38042 In visual mode, if the current line is not displayed, then the lines that are displayed shall be  
38043 expanded, scrolled, or redrawn to cause an unspecified portion of the current line to be  
38044 displayed. If the screen is redrawn, no more than the number of logical lines specified by the  
38045 value of the **window** edit option shall be displayed (unless the current line cannot be completely  
38046 displayed in the number of logical lines specified by the **window** edit option) and the current  
38047 line shall be positioned as close to the center of the displayed lines as possible (within the  
38048 constraints imposed by the distance of the line from the beginning or end of the edit buffer). If  
38049 the current line is before the first line in the display and the screen is scrolled, an unspecified  
38050 portion of the current line shall be placed on the first line of the display. If the current line is after  
38051 the last line in the display and the screen is scrolled, an unspecified portion of the current line  
38052 shall be placed on the last line of the display.

38053 In visual mode, if a line from the edit buffer (other than the current line) does not entirely fit into  
38054 the lines at the bottom of the display that are available for its presentation, the editor may  
38055 choose not to display any portion of the line. The lines of the display that do not contain text  
38056 from the edit buffer for this reason shall each consist of a single '@' character.



38057 In visual mode, the editor may choose for unspecified reasons to not update lines in the display  
 38058 to correspond to the underlying edit buffer text. The lines of the display that do not correctly  
 38059 correspond to text from the edit buffer for this reason shall consist of a single '@' character, and  
 38060 the <control>-R command shall cause the editor to update the screen to correctly represent the  
 38061 edit buffer.

38062 Open and visual mode commands that set the current column set it to a column position in the  
 38063 display, and not a character position in the line. In this case, however, the column position in the  
 38064 display shall be calculated for a infinite width display; for example, the column related to a  
 38065 character that is part of a line that has been folded onto additional screen lines will be offset from  
 38066 the screen column where the physical line begins, not from the beginning of a particular screen  
 38067 line.

38068 The physical cursor column in the display is based on the value of the current column, as  
 38069 follows, with each rule applied in turn:

- 38070 1. If the current column is after the last screen column used by the displayed line, the  
 38071 physical cursor column shall be set to the last screen column occupied by the last character  
 38072 in the current line; otherwise, the physical cursor column shall be set to the current  
 38073 column.
- 38074 2. If the character of which some portion is displayed in the screen column specified by the  
 38075 physical cursor column requires more than a single screen column:
  - 38076 a. If in text input mode, the physical cursor column shall be adjusted to the first screen  
 38077 column in which any portion of that character is displayed.
  - 38078 b. Otherwise, the physical cursor column shall be adjusted to the last screen column in  
 38079 which any portion of that character is displayed.

38080 The current column shall not be changed by these adjustments to the physical cursor column.

38081 If an error occurs during the parsing or execution of a *vi* command:

- 38082 • The terminal shall be alerted. Execution of the *vi* command shall stop, and the cursor (for  
 38083 example, the current line and column) shall not be further modified.
- 38084 • Unless otherwise specified by the following command sections, it is unspecified whether an  
 38085 informational message shall be displayed.
- 38086 • Any partially entered *vi* command shall be discarded.
- 38087 • If the *vi* command resulted from a **map** expansion, all characters from that **map** expansion  
 38088 shall be discarded, except as otherwise specified by the **map** command (see *ed*).
- 38089 • If the *vi* command resulted from the execution of a buffer, no further commands caused by  
 38090 the execution of the buffer shall be executed.

### 38091 **Page Backwards**

38092 *Synopsis:* [count] <control>-B

38093 If in open mode, the <control>-B command shall behave identically to the **z** command.  
 38094 Otherwise, if the current line is the first line of the edit buffer, it shall be an error.

38095 If the **window** edit option is less than 3, display a screen where the last line of the display shall  
 38096 be some portion of:

38097 (*current first line*) -1

38098 otherwise, display a screen where the first line of the display shall be some portion of:  
 38099  $(\textit{current first line}) - \textit{count} \times ((\textit{window edit option}) - 2)$   
 38100 If this calculation would result in a line that is before the first line of the edit buffer, the first line  
 38101 of the display shall display some portion of the first line of the edit buffer.  
 38102 *Current line*: If no lines from the previous display remain on the screen, set to the last line of the  
 38103 display; otherwise, set to  $(\textit{line} - \text{the number of new lines displayed on this screen})$ .  
 38104 *Current column*: Set to non-`<blank>`.

38105 **Scroll Forward**  
 38106 *Synopsis*:  $[\textit{count}] \text{ <control>-D}$   
 38107 If the current line is the last line of the edit buffer, it shall be an error.  
 38108 If no *count* is specified, *count* shall default to the *count* associated with the previous `<control>-D`  
 38109 or `<control>-U` command. If there was no previous `<control>-D` or `<control>-U` command, *count*  
 38110 shall default to the value of the **scroll** edit option.  
 38111 If in open mode, write lines starting with the line after the current line, until *count* lines or the  
 38112 last line of the file have been written.  
 38113 *Current line*: If the current line + *count* is past the last line of the edit buffer, set to the last line of  
 38114 the edit buffer; otherwise, set to the current line + *count*.  
 38115 *Current column*: Set to non-`<blank>`.

38116 **Scroll Forward by Line**  
 38117 *Synopsis*:  $[\textit{count}] \text{ <control>-E}$   
 38118 Display the line *count* lines after the last line currently displayed.  
 38119 If the last line of the edit buffer is displayed, it shall be an error. If there is no line *count* lines  
 38120 after the last line currently displayed, the last line of the display shall display some portion of  
 38121 the last line of the edit buffer.  
 38122 *Current line*: Unchanged if the previous current character is displayed; otherwise, set to the first  
 38123 line displayed.  
 38124 *Current column*: Unchanged.

38125 **Page Forward**  
 38126 *Synopsis*:  $[\textit{count}] \text{ <control>-F}$   
 38127 If in open mode, the `<control>-F` command shall behave identically to the **z** command.  
 38128 Otherwise, if the current line is the last line of the edit buffer, it shall be an error.  
 38129 If the **window** edit option is less than 3, display a screen where the first line of the display shall  
 38130 be some portion of:  
 38131  $(\textit{current last line}) + 1$   
 38132 otherwise, display a screen where the first line of the display shall be some portion of:  
 38133  $(\textit{current first line}) + \textit{count} \times ((\textit{window edit option}) - 2)$   
 38134 If this calculation would result in a line that is after the last line of the edit buffer, the last line of  
 38135 the display shall display some portion of the last line of the edit buffer.

38136 *Current line*: If no lines from the previous display remain on the screen, set to the first line of the  
 38137 display; otherwise, set to (*line* + the number of new lines displayed on this screen).

38138 *Current column*: Set to non-<blank>.

### 38139 **Display Information**

38140 *Synopsis*: <control>-G

38141 This command shall be equivalent to the *ex file* command .

### 38142 **Move Cursor Backwards**

38143 *Synopsis*: [*count*] <control>-H

38144 [*count*] h

38145 the current *erase* character (see *stty*)

38146 If there are no characters before the current character on the current line, it shall be an error. If  
 38147 there are less than *count* previous characters on the current line, *count* shall be adjusted to the  
 38148 number of previous characters on the line.

38149 If used as a motion command:

38150 1. The text region shall be from the character before the starting cursor up to and including  
 38151 the *count*th character before the starting cursor.

38152 2. Any text copied to a buffer shall be in character mode.

38153 If not used as a motion command:

38154 *Current line*: Unchanged.

38155 *Current column*: Set to (*column* – the number of columns occupied by *count* characters ending  
 38156 with the previous current column).

### 38157 **Move Down**

38158 *Synopsis*: [*count*] <newline>

38159 [*count*] <control>-J

38160 [*count*] <control>-M

38161 [*count*] <control>-N

38162 [*count*] j

38163 [*count*] <carriage-return>

38164 [*count*] +

38165 If there are less than *count* lines after the current line in the edit buffer, it shall be an error.

38166 If used as a motion command:

38167 1. The text region shall include the starting line and the next *count* – 1 lines.

38168 2. Any text copied to a buffer shall be in line mode.

38169 If not used as a motion command:

38170 *Current line*: Set to *current line*+ *count*.

38171 *Current column*: Set to non-<blank> for the <carriage-return> character, <control>-M, and +  
 38172 commands; otherwise, unchanged.

38173 **Clear and Redisplay**38174 *Synopsis:* <control>-L38175 If in open mode, clear the screen and redisplay the current line. Otherwise, clear and redisplay  
38176 the screen.38177 *Current line:* Unchanged.38178 *Current column:* Unchanged.38179 **Move Up**38180 *Synopsis:* [count] <control>-P

38181 [count] k

38182 [count] -

38183 If there are less than *count* lines before the current line in the edit buffer, it shall be an error.

38184 If used as a motion command:

- 38185 1. The text region shall include the starting line and the previous *count* lines.
- 38186 2. Any text copied to a buffer shall be in line mode.

38187 If not used as a motion command:

38188 *Current line:* Set to *current line* - *count*.38189 *Current column:* Set to non-<blank> for the - command; otherwise, unchanged.38190 **Redraw Screen**38191 *Synopsis:* <control>-R38192 If any lines have been deleted from the logical screen and flagged as deleted on the terminal  
38193 using the @ convention (see the beginning of the EXTENDED DESCRIPTION section), they shall  
38194 be redisplayed to match the contents of the edit buffer.38195 It is unspecified whether lines flagged with @ because they do not fit on the terminal display  
38196 shall be affected.38197 *Current line:* Unchanged.38198 *Current column:* Unchanged.38199 **Scroll Backward**38200 *Synopsis:* [count] <control>-U

38201 If the current line is the first line of the edit buffer, it shall be an error.

38202 If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D  
38203 or <control>-U command. If there was no previous <control>-D or <control>-U command, *count*  
38204 shall default to the value of the **scroll** edit option.38205 *Current line:* If *count* is greater than the current line, set to 1; otherwise, set to the current line -  
38206 *count*.38207 *Current column:* Set to non-<blank>.

**38208 Scroll Backward by Line**

38209 *Synopsis:* [count] <control>-Y

38210 Display the line *count* lines before the first line currently displayed.

38211 If the current line is the first line of the edit buffer, it shall be an error. If this calculation would  
38212 result in a line that is before the first line of the edit buffer, the first line of the display shall  
38213 display some portion of the first line of the edit buffer.

38214 *Current line:* Unchanged if the previous current character is displayed; otherwise, set to the first  
38215 line displayed.

38216 *Current column:* Unchanged.

**38217 Edit the Alternate File**

38218 *Synopsis:* <control>-^

38219 This command shall be equivalent to the *ex edit* command, with the alternate path name as its  
38220 argument.

**38221 Terminate Command or Input Mode**

38222 *Synopsis:* <ESC>

38223 If a partial *vi* command (as defined by at least one, non-*count* character) has been entered,  
38224 discard the *count* and the command character(s).

38225 Otherwise, if no command characters have been entered, and the <ESC> was the result of a map  
38226 expansion, the terminal shall be alerted and the <ESC> character shall be discarded, but it shall  
38227 not be an error.

38228 Otherwise, it shall be an error.

38229 *Current line:* Unchanged.

38230 *Current column:* Unchanged.

**38231 Search for tagstring**

38232 *Synopsis:* <control>-]

38233 If the current character is not a word or <blank> character, it shall be an error.

38234 This command shall be equivalent to the *ex tag* command, with the argument to that command  
38235 defined as follows.

38236 If the current character is a <blank> character:

- 38237 1. Skip all <blank> characters after the cursor up to the end of the line.  
38238 2. If the end of the line is reached, it shall be an error.

38239 Then, the argument to the *ex tag* command shall be the current character and all subsequent  
38240 characters, up to the first non-word character or the end of the line.

38241 **Move Cursor Forward**

38242 *Synopsis:* [ *count* ] <space>  
 38243 [ *count* ] 1 (ell)

38244 If there are less than *count* characters after the cursor on the current line, *count* shall be adjusted  
 38245 to the number of characters after the cursor on the line.

38246 If used as a motion command:

- 38247 1. If the current or *count*th character after the cursor is the last character in the line, the text  
 38248 region shall be comprised of the current character up to and including the last character in  
 38249 the line. Otherwise, the text region shall be from the current character up to, but not  
 38250 including, the *count*th character after the cursor.
- 38251 2. Any text copied to a buffer shall be in character mode.

38252 If not used as a motion command:

38253 If there are no characters after the current character on the current line, it shall be an error.

38254 *Current line:* Unchanged.

38255 *Current column:* Set to the last column that displays any portion of the *count*th character after the  
 38256 current character.

38257 **Replace Text with Results from Shell Command**

38258 *Synopsis:* [ *count* ] ! *motion shell-commands* <newline>

38259 If the motion command is the ! command repeated:

- 38260 1. If the edit buffer is empty and no *count* was supplied, the command shall be the equivalent  
 38261 of the *ex:read!* command, with the text input, and no text shall be copied to any buffer.
- 38262 2. Otherwise:
  - 38263 a. If there are less than *count* - 1 lines after the current line in the edit buffer, it shall be  
 38264 an error.
  - 38265 b. The text region shall be from the current line up to and including the next *count* - 1  
 38266 lines.

38267 Otherwise, the text region shall be the lines in which any character of the text region specified by  
 38268 the motion command appear.

38269 Any text copied to a buffer shall be in line mode.

38270 This command shall be equivalent to the *ex!* command for the specified lines.

38271 **Move Cursor to End-of-line**

38272 *Synopsis:* [ *count* ] \$

38273 It shall be an error if there are less than (*count* - 1) lines after the current line in the edit buffer.

38274 If used as a motion command:

- 38275 1. If *count* is 1:
  - 38276 a. It shall be an error if the line is empty.
  - 38277 b. Otherwise, the text region shall consist of all characters from the starting cursor to  
 38278 the last character in the line, inclusive, and any text copied to a buffer shall be in

- 38279 character mode.
- 38280 2. Otherwise, if the starting cursor position is at or before the first non-<blank> character in  
38281 the line, the text region shall consist of the current and the next *count* - 1 lines, and any text  
38282 saved to a buffer shall be in line mode.
- 38283 3. Otherwise, the text region shall consist of all characters from the starting cursor to the last  
38284 character in the line that is *count* - 1 lines forward from the current line, and any text copied  
38285 to a buffer shall be in character mode.
- 38286 If not used as a motion command:
- 38287 *Current line*: Set to the *current line* + *count* - 1.
- 38288 *Current column*: The current column is set to the last screen column of the last character in the  
38289 line, or column position 1 if the line is empty.
- 38290 The current column shall be adjusted to be on the last screen column of the last character of the  
38291 current line as subsequent commands change the current line, until a command changes the  
38292 current column.
- 38293 **Move to Matching Character**
- 38294 *Synopsis*: %
- 38295 If the character at the current position is not a parenthesis, bracket, or curly brace, search  
38296 forward in the line to the first one of those characters. If no such character is found, it shall be an  
38297 error.
- 38298 The matching character shall be the parenthesis, bracket, or curly brace matching the  
38299 parenthesis, bracket, or curly brace, respectively, that was at the current position or that was  
38300 found on the current line.
- 38301 Matching shall be determined as follows, for an open parenthesis:
- 38302 1. Set a counter to 1.
- 38303 2. Search forwards until a parenthesis is found or the end of the edit buffer is reached.
- 38304 3. If the end of the edit buffer is reached, it shall be an error.
- 38305 4. If an open parenthesis is found, increment the counter by 1.
- 38306 5. If a close parenthesis is found, decrement the counter by 1.
- 38307 6. If the counter is zero, the current character is the matching character.
- 38308 Matching for a close parenthesis shall be equivalent, except that the search shall be backwards,  
38309 from the starting character to the beginning of the buffer, a close parenthesis shall increment the  
38310 counter by 1, and an open parenthesis shall decrement the counter by 1.
- 38311 Matching for brackets and curly braces shall be equivalent, except that searching shall be done  
38312 for open and close brackets or open and close curly braces. It is implementation-defined whether  
38313 other characters are searched for and matched as well.
- 38314 If used as a motion command:
- 38315 1. If the matching cursor was after the starting cursor in the edit buffer, and the starting  
38316 cursor position was at or before the first non-<blank> character in the starting line, and the  
38317 matching cursor position was at or after the last non-<blank> character in the matching  
38318 line, the text region shall consist of the current line to the matching line, inclusive, and any  
38319 text copied to a buffer shall be in line mode.

38320 2. If the matching cursor was before the starting cursor in the edit buffer, and the starting  
 38321 cursor position was at or after the last non-<blank> character in the starting line, and the  
 38322 matching cursor position was at or before the first non-<blank> character in the matching  
 38323 line, the text region shall consist of the current line to the matching line, inclusive, and any  
 38324 text copied to a buffer shall be in line mode.

38325 3. Otherwise, the text region shall consist of the starting character to the matching character,  
 38326 inclusive, and any text copied to a buffer shall be in character mode.

38327 If not used as a motion command:

38328 *Current line*: Set to the line where the matching character is located.

38329 *Current column*: Set to the last column where any portion of the matching character is displayed.

### 38330 **Repeat Substitution**

38331 *Synopsis*: &

38332 Repeat the previous substitution command. This command shall be equivalent to the *ex &*  
 38333 command with the current line as its addresses, and without *options*, *count*, or *flags*.

### 38334 **Return to Previous Context at Beginning of Line**

38335 *Synopsis*: ' *character*

38336 It shall be an error if there is no line in the edit buffer marked by *character*.

38337 If used as a motion command:

38338 1. If the starting cursor is after the marked cursor, then the locations of the starting cursor  
 38339 and the marked cursor in the edit buffer shall be logically swapped.

38340 2. The text region shall consist of the starting line up to and including the marked line, and  
 38341 any text copied to a buffer shall be in line mode.

38342 If not used as a motion command:

38343 *Current line*: Set to the line referenced by the mark.

38344 *Current column*: Set to non-<blank>.

### 38345 **Return to Previous Context**

38346 *Synopsis*: ` *character*

38347 It shall be an error if the marked line is no longer in the edit buffer. If the marked line no longer  
 38348 contains a character in the saved numbered character position, it shall be as if the marked  
 38349 position is the first non-<blank> character.

38350 If used as a motion command:

38351 1. It shall be an error if the marked cursor references the same character in the edit buffer as  
 38352 the starting cursor.

38353 2. If the starting cursor is after the marked cursor, then the locations of the starting cursor  
 38354 and the marked cursor in the edit buffer shall be logically swapped.

38355 3. If the starting line is empty or the starting cursor is at or before the first non-<blank>  
 38356 character of the starting line, and the marked cursor line is empty or the marked cursor  
 38357 references the first character of the marked cursor line, the text region shall consist of all  
 38358 lines containing characters from the starting cursor to the line before the marked cursor



- 38359 line, inclusive, and any text copied to a buffer shall be in line mode.
- 38360 4. Otherwise, if the marked cursor line is empty or the marked cursor references a character  
38361 at or before the first non-<blank> character of the marked cursor line, the region of text  
38362 shall be from the starting cursor to the last character of the line before the marked cursor  
38363 line, inclusive, and any text copied to a buffer shall be in character mode.
- 38364 5. Otherwise, the region of text shall be from the starting cursor (inclusive), to the marked  
38365 cursor (exclusive), and any text copied to a buffer shall be in character mode.

38366 If not used as a motion command:

38367 *Current line*: Set to the line referenced by the mark.

38368 *Current column*: Set to the last column in which any portion of the character referenced by the  
38369 mark is displayed.

### 38370 **Return to Previous Section**

38371 *Synopsis*: [ [

38372 Move the cursor backward through the edit buffer to the first character of the previous section  
38373 boundary, *count* times.

38374 If used as a motion command:

- 38375 1. If the starting cursor was at the first character of the starting line or the starting line was  
38376 empty, and the first character of the boundary was the first character of the boundary line,  
38377 the text region shall consist of the current line up to and including the line where the  
38378 *countth* next boundary starts, and any text copied to a buffer shall be in line mode.
- 38379 2. If the boundary was the last line of the edit buffer or the last character of the last line of the  
38380 edit buffer, the text region shall consist of the last character in the edit buffer up to and  
38381 including the starting character, and any text saved to a buffer shall be in character mode.
- 38382 3. Otherwise, the text region shall consist of the starting character up to but not including the  
38383 first character in the *countth* next boundary, and any text copied to a buffer shall be in  
38384 character mode.

38385 If the *lisp* option is set, a section boundary is also identified by a line with a leading ' ( ' .

38386 If not used as a motion command:

38387 *Current line*: Set to the line where the *countth* next boundary in the edit buffer starts.

38388 *Current column*: Set to the last column in which any portion of the first character of the *countth*  
38389 next boundary is displayed, or column position 1 if the line is empty.

### 38390 **Move to Next Section**

38391 *Synopsis*: ] ]

38392 Move the cursor forward through the edit buffer to the first character of the next section  
38393 boundary, *count* times.

38394 If used as a motion command:

- 38395 1. If the starting cursor was at the first character of the starting line or the starting line was  
38396 empty, and the first character of the boundary was the first character of the boundary line,  
38397 the text region shall consist of the current line up to and including the line where the  
38398 *countth* previous boundary starts, and any text copied to a buffer shall be in line mode.

38399 2. If the boundary was the first line of the edit buffer, the text region shall consist of the first  
 38400 character in the edit buffer up to but not including the starting character, and any text  
 38401 copied to a buffer shall be in character mode.

38402 3. Otherwise, the text region shall consist of the first character in the *count*th previous section  
 38403 boundary up to but not including the starting character, and any text copied to a buffer  
 38404 shall be in character mode.

38405 If the *lisp* option is set, a section boundary is also identified by a line with a leading ' ( ' .

38406 If not used as a motion command:

38407 *Current line*: Set to the line where the *count*th previous boundary in the edit buffer starts.

38408 *Current column*: Set to the last column in which any portion of the first character of the *count*th  
 38409 previous boundary is displayed, or column position 1 if the line is empty.

### 38410 **Move to First Non-<blank> Position on Current Line**

38411 *Synopsis*:     ^

38412 If used as a motion command:

38413 1. If the line has no non-<blank> characters, or if the cursor is at the first non-<blank>  
 38414 character of the line, it shall be an error.

38415 2. If the cursor is before the first non-<blank> character of the line, the text region shall be  
 38416 comprised of the current character, up to, but not including, the first non-<blank>  
 38417 character of the line.

38418 3. If the cursor is after the first non-<blank> character of the line, the text region shall be from  
 38419 the character before the starting cursor up to and including the first non-<blank> character  
 38420 of the line.

38421 4. Any text copied to a buffer shall be in character mode.

38422 If not used as a motion command:

38423 *Current line*: Unchanged.

38424 *Current column*: Set to non-<blank>.

### 38425 **Current and line above**

38426 *Synopsis*:     [ *count* ] \_

38427 If there are less than *count* - 1 lines after the current line in the edit buffer, it shall be an error.

38428 If used as a motion command:

38429 1. If *count* is less than 2, the text region shall be the current line.

38430 2. Otherwise, the text region shall include the starting line and the next *count* - 1 lines.

38431 3. Any text copied to a buffer shall be in line mode.

38432 If not used as a motion command:

38433 *Current line*: Set to current line + *count* - 1.

38434 *Current column*: Set to non-<blank>.

38435 **Move Back to Beginning of Sentence**38436 *Synopsis:* [ *count* ] (38437 Move backward to the beginning of a sentence. This command shall be equivalent to the `[[`  
38438 command, with the exception that sentence boundaries shall be used instead of section  
38439 boundaries.38440 If the *lisp* option is set, a LISP s-expression is considered a sentence for this command.38441 **Move Forward to Beginning of Sentence**38442 *Synopsis:* [ *count* ] )38443 Move forward to the beginning of a sentence. This command shall be equivalent to the `]]`  
38444 command, with the exception that sentence boundaries shall be used instead of section  
38445 boundaries.38446 If the *lisp* option is set, a LISP s-expression is considered a sentence for this command.38447 **Move Back to Preceding Paragraph**38448 *Synopsis:* [ *count* ] {38449 Move back to the beginning of the preceding paragraph. This command shall be equivalent to  
38450 the `[[` command, with the exception that paragraph boundaries shall be used instead of section  
38451 boundaries.38452 **Move Forward to Next Paragraph**38453 *Synopsis:* [ *count* ] }38454 Move forward to the beginning of the next paragraph. This command shall be equivalent to the  
38455 `]]` command, with the exception that paragraph boundaries shall be used instead of section  
38456 boundaries.38457 **Move to Specific Column Position**38458 *Synopsis:* [ *count* ] |38459 For the purposes of this command, lines that are too long for the current display and that have  
38460 been folded shall be treated as having a single, 1-based, number of columns.38461 If there are less than *count* columns in which characters from the current line are displayed on  
38462 the screen, *count* shall be adjusted to be the last column in which any portion of the line is  
38463 displayed on the screen.

38464 If used as a motion command:

- 38465 1. If the line is empty, or the cursor character is the same as the character on the *count*th  
38466 column of the line, it shall be an error.
- 38467 2. If the cursor is before the *count*th column of the line, the text region shall be comprised of  
38468 the current character, up to but not including the character on the *count*th column of the  
38469 line.
- 38470 3. If the cursor is after the *count*th column of the line, the text region shall be from the  
38471 character before the starting cursor up to and including the character on the *count*th  
38472 column of the line.

38473 4. Any text copied to a buffer shall be in character mode.

38474 If not used as a motion command:

38475 *Current line*: Unchanged.

38476 *Current column*: Set to the last column in which any portion of the character that is displayed in  
38477 the *count* column of the line is displayed.

### 38478 **Reverse Find Character**

38479 *Synopsis*: [ *count* ] ,

38480 If the last **F**, **f**, **T**, or **t** command was **F**, **f**, **T**, or **t**, this command shall be equivalent to an **f**, **F**, **t**, or  
38481 **T** command, respectively, with the specified *count* and the same search character.

38482 If there was no previous **F**, **f**, **T**, or **t** command, it shall be an error.

### 38483 **Repeat**

38484 *Synopsis*: [ *count* ] .

38485 Repeat the last **!**, **<**, **>**, **A**, **C**, **D**, **I**, **J**, **O**, **P**, **R**, **S**, **X**, **Y**, **a**, **c**, **d**, **i**, **o**, **p**, **r**, **s**, **x**, **y**, or **~** command. It shall  
38486 be an error if none of these commands have been executed. Commands (other than commands  
38487 that enter text input mode) executed as a result of map expansions, shall not change the value of  
38488 the last repeatable command.

38489 Repeated commands with associated motion commands shall repeat the motion command as  
38490 well; however, any specified *count* shall replace the *count*(s) that were originally specified to the  
38491 repeated command or its associated motion command.

38492 If the motion component of the repeated command is **f**, **F**, **t**, or **T**, the repeated command shall  
38493 not set the remembered search character for the **;** and **,** commands.

38494 If the repeated command is **p** or **P**, and the buffer associated with that command was a numeric  
38495 buffer named with a number less than 9, the buffer associated with the repeated command shall  
38496 be set to be the buffer named by the name of the previous buffer logically incremented by 1.

38497 If the repeated character is a text input command, the input text associated with that command  
38498 is repeated literally:

- 38499 • Input characters are neither macro or abbreviation-expanded.
- 38500 • Input characters are not interpreted in any special way with the exception that the <newline>  
38501 character and the <carriage-return> character, and <control>-T behave as described in **Input**  
38502 **Mode Commands in vi** (on page 3235).

38503 *Current line*: Set as described for the repeated command.

38504 *Current column*: Set as described for the repeated command.

### 38505 **Find Regular Expression**

38506 *Synopsis*: /

38507 If the input line contains no characters, it shall be equivalent to a line containing only the last  
38508 regular expression encountered. The enhanced regular expressions supported by *vi* are  
38509 described in **Regular Expressions in ex** (on page 2601).

38510 Otherwise, the line shall be interpreted as one or more regular expressions, optionally followed  
38511 by an address offset or a *vi z* command.

38512 If the regular expression is not the last regular expression on the line, or if a line offset or **z**  
 38513 command is specified, the regular expression shall be terminated by an unescaped `'/'`  
 38514 character, which shall not be used as part of the regular expression. If the regular expression is  
 38515 not the first regular expression on the line, it shall be preceded by zero or more `<blank>`  
 38516 characters, a semicolon, zero or more `<blank>` characters, and a leading `'/'` character, which  
 38517 shall not be interpreted as part of the regular expression. It shall be an error to precede any  
 38518 regular expression with any characters other than these.

38519 Each search shall begin from the character after the first character of the last match (or, if it is the  
 38520 first search, after the cursor). If the **wrapsan** edit option is set, the search shall continue to the  
 38521 character before the starting cursor character; otherwise, to the end of the edit buffer. It shall be  
 38522 an error if any search fails to find a match, and an informational message to this effect shall be  
 38523 displayed.

38524 An optional address offset (see **Addressing in ex** (on page 2571)) can be specified after the last  
 38525 regular expression by including a trailing `'/'` character after the regular expression and  
 38526 specifying the address offset. This offset will be from the line containing the match for the last  
 38527 regular expression specified. It shall be an error if the line offset would indicate a line address  
 38528 less than 1 or greater than the last line in the edit buffer. An address offset of zero shall be  
 38529 supported. It shall be an error to follow the address offset with any other characters than  
 38530 `<blank>` characters.

38531 If not used as a motion command, an optional **z** command (see **Redraw Window** (on page 3234))  
 38532 can be specified after the last regular expression by including a trailing `'/'` character after the  
 38533 regular expression, zero or more `<blank>` characters, a `'z'`, zero or more `<blank>` characters, an  
 38534 optional new **window** edit option value, zero or more `<blank>` characters, and a location  
 38535 character. The effect shall be as if the **z** command was executed after the `/` command. It shall be  
 38536 an error to follow the **z** command with any other characters than `<blank>` characters.

38537 The remembered search direction shall be set to forward.

38538 If used as a motion command:

- 38539 1. It shall be an error if the last match references the same character in the edit buffer as the  
 38540 starting cursor.
- 38541 2. If any address offset is specified, the last match shall be adjusted by the specified offset as  
 38542 described previously.
- 38543 3. If the starting cursor is after the last match, then the locations of the starting cursor and the  
 38544 last match in the edit buffer shall be logically swapped.
- 38545 4. If any address offset is specified, the text region shall consist of all lines containing  
 38546 characters from the starting cursor to the last match line, inclusive, and any text copied to a  
 38547 buffer shall be in line mode.
- 38548 5. Otherwise, if the starting line is empty or the starting cursor is at or before the first non-  
 38549 `<blank>` character of the starting line, and the last match line is empty or the last match  
 38550 starts at the first character of the last match line, the text region shall consist of all lines  
 38551 containing characters from the starting cursor to the line before the last match line,  
 38552 inclusive, and any text copied to a buffer shall be in line mode.
- 38553 6. Otherwise, if the last match line is empty or the last match begins at a character at or  
 38554 before the first non-`<blank>` of the last match line, the region of text shall be from the  
 38555 current cursor to the last character of the line before the last match line, inclusive, and any  
 38556 text copied to a buffer shall be in character mode.

38557 7. Otherwise, the region of text shall be from the current cursor (inclusive), to the first  
 38558 character of the last match (exclusive), and any text copied to a buffer shall be in  
 38559 character mode.

38560 If not used as a motion command:

38561 *Current line*: If a match is found, set to the last matched line plus the address offset, if any;  
 38562 otherwise, unchanged.

38563 *Current column*: Set to the last column on which any portion of the first character in the last  
 38564 matched string is displayed, if a match is found; otherwise, unchanged.

### 38565 **Move to First Character in Line**

38566 *Synopsis*: 0 (zero)

38567 Move to the first character on the current line. The character '0' shall not be interpreted as a  
 38568 command if it is immediately preceded by a digit.

38569 If used as a motion command:

- 38570 1. If the cursor character is the first character in the line, it shall be an error.
- 38571 2. The text region shall be from the character before the cursor character up to and including  
 38572 the first character in the line.
- 38573 3. Any text copied to a buffer shall be in character mode.

38574 If not used as a motion command:

38575 *Current line*: Unchanged.

38576 *Current column*: The last column in which any portion of the first character in the line is  
 38577 displayed, or if the line is empty, unchanged.

### 38578 **Execute an ex Command**

38579 *Synopsis*: :

38580 Execute one or more *ex* commands.

38581 If any portion of the screen other than the last line of the screen was overwritten by any *ex*  
 38582 command (except **shell**), *vi* shall display a message indicating that it is waiting for an input from  
 38583 the user, and shall then read a character. This action may also be taken for other, unspecified  
 38584 reasons.

38585 If the next character entered is a ':', another *ex* command shall be accepted and executed. Any  
 38586 other character shall cause the screen to be refreshed and *vi* shall return to command mode.

38587 *Current line*: As specified for the *ex* command.

38588 *Current column*: As specified for the *ex* command.

38589       **Repeat Find**38590       *Synopsis:*     [ *count* ] ;38591       This command shall be equivalent to the last **F**, **f**, **T**, or **t** command, with the specified *count*, and  
38592       with the same search character used for the last **F**, **f**, **T**, or **t** command. If there was no previous **F**,  
38593       **f**, **T**, or **t** command, it shall be an error.38594       **Shift Left**38595       *Synopsis:*     [ *count* ] < *motion*

38596       If the motion command is the &lt; command repeated:

38597           1. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an  
38598           error.38599           2. The text region shall be from the current line, up to and including the next *count* -1 lines.38600       Shift any line in the text region specified by the *count* and motion command one shiftwidth (see  
38601       the *ex* **shiftwidth** option) toward the start of the line, as described by the *ex* < command. The  
38602       unshifted lines shall be copied to the unnamed buffer in line mode.38603       *Current line:* If the motion was from the current cursor position toward the end of the edit  
38604       buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region  
38605       specified by the motion command.38606       *Current column:* Set to non-<blank>.38607       **Shift Right**38608       *Synopsis:*     [ *count* ] > *motion*

38609       If the motion command is the &gt; command repeated:

38610           1. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an  
38611           error.38612           2. The text region shall be from the current line, up to and including the next *count* -1 lines.38613       Shift any line with characters in the text region specified by the *count* and motion command one  
38614       shiftwidth (see the *ex* **shiftwidth** option) away from the start of the line, as described by the *ex* >  
38615       command. The unshifted lines shall be copied into the unnamed buffer in line mode.38616       *Current line:* If the motion was from the current cursor position toward the end of the edit  
38617       buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region  
38618       specified by the motion command.38619       *Current column:* Set to non-<blank>.38620       **Scan Backwards for Regular Expression**38621       *Synopsis:*     ?38622       Scan backwards; The ? command shall be equivalent to the / command (see **Find Regular**  
38623       **Expression** (on page 3216)) with the following exceptions:

38624           1. The input prompt shall be a ' ? '.

38625           2. Each search shall begin from the character before the first character of the last match (or, if  
38626           it is the first search, the character before the cursor character).

- 38627           3. The search direction shall be from the cursor toward the beginning of the edit buffer, and  
 38628           the **wrapsan** edit option shall affect whether the search wraps to the end of the edit buffer  
 38629           and continues.
- 38630           4. The remembered search direction shall be set to backward.

### 38631       **Execute**

38632       *Synopsis:*     @*buffer*

38633       If the *buffer* is specified as @, the last buffer executed shall be used. If no previous buffer has been  
 38634       executed, it shall be an error.

38635       Behave as if the contents of the named buffer were entered as standard input. After each line of a  
 38636       line-mode buffer, and all but the last line of a character mode buffer, behave as if a <newline>  
 38637       character were entered as standard input.

38638       If an error occurs during this process, an error message shall be written, and no more characters  
 38639       resulting from the execution of this command shall be processed.

38640       If a *count* is specified, behave as if that count were entered as user input before the characters  
 38641       from the @ buffer were entered.

38642       *Current line:* As specified for the individual commands.

38643       *Current column:* As specified for the individual commands.

### 38644       **Reverse Case**

38645       *Synopsis:*     [*count*] ~

38646       Reverse the case of the current character and the next *count* - 1 characters, such that lowercase  
 38647       characters that have uppercase counterparts shall be changed to uppercase characters, and  
 38648       uppercase characters that have lowercase counterparts shall be changed to lowercase characters,  
 38649       as prescribed by the current locale. No other characters shall be affected by this command.

38650       If there are less than *count* - 1 characters after the cursor in the edit buffer, *count* shall be adjusted  
 38651       to the number of characters after the cursor in the edit buffer minus 1.

38652       For the purposes of this command, the next character after the last character on the line shall be  
 38653       the next character in the edit buffer.

38654       *Current line:* Set to the line including the (*count*-1)th character after the cursor.

38655       *Current column:* Set to the last column in which any portion of the (*count*-1)th character after the  
 38656       cursor is displayed.

### 38657       **Reindent**

38658       *Synopsis:*     [*count*]=[*motion*]

38659       If the *lisp* option is set, reindents the specified lines, as though they were typed in with **lisp** and  
 38660       **autoindent** set.

38661       *Current line:* Unchanged.

38662       *Current column:* Move to the first non-<blank> character of the line or the last character if the  
 38663       line is a blank line.



38664 **Append**38665 *Synopsis:* [count] a

38666 Enter text input mode after the current cursor position. No characters already in the edit buffer  
 38667 shall be affected by this command. A *count* shall cause the input text to be appended *count* -1  
 38668 more times to the end of the input.

38669 *Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi**  
 38670 (on page 3235)).

38671 **Append at End-of-Line**38672 *Synopsis:* [count] A38673 This command shall be equivalent to the *vi* command:

38674 \$ [ count ] a

38675 (see **Append**).38676 **Move Backward to Preceding Word**38677 *Synopsis:* [count] b

38678 With the exception that words are used as the delimiter instead of bigwords, this command shall  
 38679 be equivalent to the **B** command.

38680 **Move Backward to Preceding Bigword**38681 *Synopsis:* [count] B

38682 If the edit buffer is empty or the cursor is on the first character of the edit buffer, it shall be an  
 38683 error. If less than *count* bigwords begin between the cursor and the start of the edit buffer, *count*  
 38684 shall be adjusted to the number of bigword beginnings between the cursor and the start of the  
 38685 edit buffer.

38686 If used as a motion command:

38687 1. The text region shall be from the first character of the *count*th previous bigword beginning  
 38688 up to but not including the cursor character.

38689 2. Any text copied to a buffer shall be in character mode.

38690 If not used as a motion command:

38691 *Current line:* Set to the line containing the *current column*.

38692 *Current column:* Set to the last column upon which any part of the first character of the *count*th  
 38693 previous bigword is displayed.

38694 **Change**38695 *Synopsis:* [buffer][count] c motion38696 If the motion command is the *c* command repeated:

38697 1. The buffer text shall be in line mode.

38698 2. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an  
 38699 error.

- 38700           3. The text region shall be from the current line up to and including the next *count* -1 lines.  
 38701           Otherwise, the buffer text mode and text region shall be as specified by the motion command.  
 38702           The replaced text shall be copied into *buffer*, if specified, and into the unnamed buffer. If the text  
 38703           to be replaced contains characters from more than a single line, or the buffer text is in line mode,  
 38704           the replaced text shall be copied into the numeric buffers as well.  
 38705           If the buffer text is in line mode:  
 38706           1. Any lines that contain characters in the region shall be deleted, and the editor shall enter  
 38707           text input mode at the beginning of a new line which shall replace the first line deleted.  
 38708           2. If the **autoindent** edit option is set, **autoindent** characters equal to the **autoindent**  
 38709           characters on the first line deleted shall be inserted as if entered by the user.  
 38710           Otherwise, if characters from more than one line are in the region of text:  
 38711           1. The text shall be deleted.  
 38712           2. Any text remaining in the last line in the text region shall be appended to the first line in  
 38713           the region, and the last line in the region shall be deleted.  
 38714           3. The editor shall enter text input mode after the last character not deleted from the first line  
 38715           in the text region, if any; otherwise, on the first column of the first line in the region.  
 38716           Otherwise:  
 38717           1. If the glyph for ' \$ ' is smaller than the region, the end of the region shall be marked with a  
 38718           ' \$ ' .  
 38719           2. The editor shall enter text input mode, overwriting the region of text.  
 38720           *Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi**  
 38721           (on page 3235)).  
 38722           **Change to End-of-Line**  
 38723           *Synopsis:*     [*buffer*][*count*] C  
 38724           This command shall be equivalent to the *vi* command:  
 38725           [*buffer*][*count*] c\$  
 38726           See the **c** command.  
 38727           **Delete**  
 38728           *Synopsis:*     [*buffer*][*count*] d *motion*  
 38729           If the motion command is the **d** command repeated:  
 38730           1. The buffer text shall be in line mode.  
 38731           2. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an  
 38732           error.  
 38733           3. The text region shall be from the current line up to and including the next *count* -1 lines.  
 38734           Otherwise, the buffer text mode and text region shall be as specified by the motion command.  
 38735           If in open mode, and the current line is deleted, and the line remains on the display, an '@'  
 38736           character shall be displayed as the first glyph of that line.

38737 Delete the region of text into *buffer*, if specified, and into the unnamed buffer. If the text to be  
 38738 deleted contains characters from more than a single line, or the buffer text is in line mode, the  
 38739 deleted text shall be copied into the numeric buffers, as well.

38740 *Current line*: Set to the first text region line that appears in the edit buffer, unless that line has  
 38741 been deleted, in which case it shall be set to the last line in the edit buffer, or line 1 if the edit  
 38742 buffer is empty.

38743 *Current column*:

- 38744 1. If the line is empty, set to column position 1.
- 38745 2. Otherwise, if the buffer text is in line mode or the motion was from the cursor toward the  
 38746 end of the edit buffer:
  - 38747 a. If a character from the current line is displayed in the current column, set to the last  
 38748 column that displays any portion of that character.
  - 38749 b. Otherwise, set to the last column in which any portion of any character in the line is  
 38750 displayed.
- 38751 3. Otherwise, if a character is displayed in the column that began the text region, set to the  
 38752 last column that displays any portion of that character.
- 38753 4. Otherwise, set to the last column in which any portion of any character in the line is  
 38754 displayed.

### 38755 **Delete to End-of-Line**

38756 *Synopsis*: [*buffer*] D

38757 Delete the text from the current position to the end of the current line; equivalent to the *vi*  
 38758 command:

38759 [*buffer*] d\$

### 38760 **Move to End-of-Word**

38761 *Synopsis*: [*count*] e

38762 With the exception that words are used instead of bigwords as the delimiter, this command shall  
 38763 be equivalent to the **E** command.

### 38764 **Move to End-of-Bigword**

38765 *Synopsis*: [*count*] E

38766 If the edit buffer is empty it shall be an error. If less than *count* bigwords end between the cursor  
 38767 and the end of the edit buffer, *count* shall be adjusted to the number of bigword endings between  
 38768 the cursor and the end of the edit buffer.

38769 If used as a motion command:

- 38770 1. The text region shall be from the last character of the *count*th next bigword up to and  
 38771 including the cursor character.
- 38772 2. Any text copied to a buffer shall be in character mode.

38773 If not used as a motion command:

38774 *Current line*: Set to the line containing the current column.

38775 *Current column*: Set to the last column upon which any part of the last character of the *countth*  
38776 next bigword is displayed.

### 38777 **Find Character in Current Line (Forward)**

38778 *Synopsis*: [ *count* ] f *character*

38779 It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

38780 If used as a motion command:

38781 1. The text range shall be from the cursor character up to and including the *countth*  
38782 occurrence of the specified character after the cursor.

38783 2. Any text copied to a buffer shall be in character mode.

38784 If not used as a motion command:

38785 *Current line*: Unchanged.

38786 *Current column*: Set to the last column in which any portion of the *countth* occurrence of the  
38787 specified character after the cursor appears in the line.

### 38788 **Find Character in Current Line (Reverse)**

38789 *Synopsis*: [ *count* ] F *character*

38790 It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

38791 If used as a motion command:

38792 1. The text region shall be from the *countth* occurrence of the specified character before the  
38793 cursor, up to, but not including the cursor character.

38794 2. Any text copied to a buffer shall be in character mode.

38795 If not used as a motion command:

38796 *Current line*: Unchanged.

38797 *Current column*: Set to the last column in which any portion of the *countth* occurrence of the  
38798 specified character before the cursor appears in the line.

### 38799 **Move to Line**

38800 *Synopsis*: [ *count* ] G

38801 If *count* is not specified, it shall default to the last line of the edit buffer. If *count* is greater than  
38802 the last line of the edit buffer, it shall be an error.

38803 If used as a motion command:

38804 1. The text region shall be from the cursor line up to and including the specified line.

38805 2. Any text copied to a buffer shall be in line mode.

38806 If not used as a motion command:

38807 *Current line*: Set to *count* if *count* is specified; otherwise, the last line.

38808 *Current column*: Set to non-<blank>.

38809        **Move to Top of Screen**38810        *Synopsis:*     [ *count* ] H38811        If the beginning of the line *count* greater than the first line of which any portion appears on the  
38812        display does not exist, it shall be an error.

38813        If used as a motion command:

- 38814        1. If in open mode, the text region shall be the current line.
- 38815        2. Otherwise, the text region shall be from the starting line up to and including (the first line  
38816        of the display + *count* -1).
- 38817        3. Any text copied to a buffer shall be in line mode.

38818        If not used as a motion command:

38819        If in open mode, this command shall set the current column to non-&lt;blank&gt; and do nothing else.

38820        Otherwise, it shall set the current line and current column as follows.

38821        *Current line:* Set to (the first line of the display + *count* -1).38822        *Current column:* Set to non-<blank>.38823        **Insert Before Cursor**38824        *Synopsis:*     [ *count* ] i

38825        Enter text input mode before the current cursor position. No characters already in the edit buffer  
38826        shall be affected by this command. A *count* shall cause the input text to be appended *count* -1  
38827        more times to the end of the input.

38828        *Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi**  
38829        (on page 3235)).

38830        **Insert at Beginning of Line**38831        *Synopsis:*     [ *count* ] I38832        This command shall be equivalent to the *vi* command `^[count]i` command.38833        **Join**38834        *Synopsis:*     [ *count* ] J

38835        If the current line is the last line in the edit buffer, it shall be an error.

38836        This command shall be equivalent to the *ex* **join** command with no addresses, and an *ex*  
38837        command *count* value of 1 if *count* was not specified or if a *count* of 1 was specified, and an *ex*  
38838        command *count* value of *count* -1 for any other value of *count*, except that the current line and  
38839        column shall be set as follows.

38840        *Current line:* Unchanged.

38841        *Current column:* The last column in which any portion of the character following the last  
38842        character in the initial line is displayed, or the last character in the line if no characters were  
38843        appended.

38844 **Move to Bottom of Screen**38845 *Synopsis:* [count] L38846 If the beginning of the line count less than the last line of which any portion appears on the  
38847 display does not exist, it shall be an error.

38848 If used as a motion command:

- 38849 1. If in open mode, the text region shall be the current line.
- 38850 2. Otherwise, the text region shall include all lines from the starting cursor line to (the last  
38851 line of the display  $-(count - 1)$ ).
- 38852 3. Any text copied to a buffer shall be in line mode.

38853 If not used as a motion command:

- 38854 1. If in open mode, this command shall set the current column to non-<blank> and do  
38855 nothing else.
- 38856 2. Otherwise, it shall set the current line and current column as follows.

38857 *Current line:* Set to (the last line of the display  $-(count - 1)$ ).38858 *Current column:* Set to non-<blank>.38859 **Mark Position**38860 *Synopsis:* m letter38861 This command shall be equivalent to the *ex mark* command with the specified character as an  
38862 argument.38863 **Move to Middle of Screen**38864 *Synopsis:* M

38865 The middle line of the display shall be calculated as follows:

38866  $(\text{the top line of the display}) + (((\text{number of lines displayed}) + 1) / 2) - 1$ 

38867 If used as a motion command:

- 38868 1. If in open mode, the text region shall be the current line.
- 38869 2. Otherwise, the text region shall include all lines from the starting cursor line up to and  
38870 including the middle line of the display.
- 38871 3. Any text copied to a buffer shall be in line mode.

38872 If not used as a motion command:

38873 If in open mode, this command shall set the current column to non-&lt;blank&gt; and do nothing else.

38874 Otherwise, it shall set the current line and current column as follows.

38875 *Current line:* Set to the middle line of the display.38876 *Current column:* Set to non-<blank>.

38877 **Repeat Regular Expression Find (Forward)**38878 *Synopsis:* n38879 If the remembered search direction was forward, the **n** command shall be equivalent to the *vi /*  
38880 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi ?*  
38881 command with no characters entered by the user.38882 If the **n** command is used as a motion command for the **!** command, the editor shall not enter  
38883 text input mode on the last line on the screen, and shall behave as if the user entered a single **' ! '**  
38884 character as the text input.38885 **Repeat Regular Expression Find (Reverse)**38886 *Synopsis:* N38887 Scan for the next match of the last pattern given to */* or *?*, but in the reverse direction; this is the  
38888 reverse of **n**.38889 If the remembered search direction was forward, the **N** command shall be equivalent to the *vi ?*  
38890 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi /*  
38891 command with no characters entered by the user. If the **N** command is used as a motion  
38892 command for the **!** command, the editor shall not enter text input mode on the last line on the  
38893 screen, and shall behave as if the user entered a single **!** character as the text input.38894 **Insert Empty Line Below**38895 *Synopsis:* o38896 Enter text input mode in a new line appended after the current line. A *count* shall cause the input  
38897 text to be appended *count* - 1 more times to the end of the already added text, each time starting  
38898 on a new, appended line.38899 *Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi**  
38900 (on page 3235)).38901 **Insert Empty Line Above**38902 *Synopsis:* O38903 Enter text input mode in a new line inserted before the current line. A *count* shall cause the input  
38904 text to be appended *count* - 1 more times to the end of the already added text, each time starting  
38905 on a new, appended line.38906 *Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi**  
38907 (on page 3235)).38908 **Put from Buffer Following**38909 *Synopsis:* [*buffer*] p38910 If no *buffer* is specified, the unnamed buffer shall be used.38911 If the buffer text is in line mode, the text shall be appended below the current line, and each line  
38912 of the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be  
38913 appended *count* - 1 more times to the end of the already added text, each time starting on a new,  
38914 appended line.38915 If the buffer text is in character mode, the text shall be appended into the current line after the  
38916 cursor, and each line of the buffer other than the first and last shall become a new line in the edit

38917 buffer. A *count* shall cause the buffer text to be appended *count* –1 more times to the end of the  
38918 already added text, each time starting after the last added character.

38919 *Current line*: If the buffer text is in line mode, set the line to line +1; otherwise, unchanged.

38920 *Current column*: If the buffer text is in line mode:

- 38921 1. If there is a non-<blank> character in the first line of the buffer, set to the last column on  
38922 which any portion of the first non-<blank> character in the line is displayed.
- 38923 2. If there is no non-<blank> character in the first line of the buffer, set to the last column on  
38924 which any portion of the last character in the first line of the buffer is displayed.

38925 If the buffer text is in character mode:

- 38926 1. If the text in the buffer is from more than a single line, then set to the last column on which  
38927 any portion of the first character from the buffer is displayed.
- 38928 2. Otherwise, if the buffer is the unnamed buffer, set to the last column on which any portion  
38929 of the last character from the buffer is displayed.
- 38930 3. Otherwise, set to the first column on which any portion of the first character from the  
38931 buffer is displayed.

### 38932 **Put from Buffer Before**

38933 *Synopsis*: [ *buffer* ] P

38934 If no *buffer* is specified, the unnamed buffer shall be used.

38935 If the buffer text is in line mode, the text shall be inserted above the current line, and each line of  
38936 the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be  
38937 appended *count* –1 more times to the end of the already added text, each time starting on a new,  
38938 appended line.

38939 If the buffer text is in character mode, the text shall be inserted into the current line before the  
38940 cursor, and each line of the buffer other than the first and last shall become a new line in the edit  
38941 buffer. A *count* shall cause the buffer text to be appended *count* –1 more times to the end of the  
38942 already added text, each time starting after the last added character.

38943 *Current line*: Unchanged.

38944 *Current column*: If the buffer text is in line mode:

- 38945 1. If there is a non-<blank> character in the first line of the buffer, set to the last column on  
38946 which any portion of that character is displayed.
- 38947 2. If there is no non-<blank> character in the first line of the buffer, set to the last column on  
38948 which any portion of the last character in the first line of the buffer is displayed.

38949 If the buffer text is in character mode:

- 38950 1. If the buffer is the unnamed buffer, set to the last column on which any portion of the last  
38951 character from the buffer is displayed.
- 38952 2. Otherwise, set to the first column on which any portion of the first character from the  
38953 buffer is displayed.



38954 **Enter ex Mode**38955 *Synopsis:* Q38956 Leave visual or open mode and enter *ex* command mode.38957 *Current line:* Unchanged.38958 *Current column:* Unchanged.38959 **Replace Character**38960 *Synopsis:* [*count*] r *character*38961 **Notes to Reviewers**38962 *This section with side shading will not appear in the final copy. - Ed.*38963 D3, XCU, ERN 270: The description of R is not correct. R is not the same as the i command,  
38964 which is what the text describes. Something should be done here when .2b is approved.38965 Replace the *count* characters at and after the cursor with the specified character. If there are less  
38966 than *count* characters at and after the cursor on the line, it shall be an error.38967 If character is <control>-V, any next character other than the <newline> shall be stripped of any  
38968 special meaning and used as a literal character.38969 If character is <ESC>, no replacement shall be made and the current line and current column  
38970 shall be unchanged.38971 If character is <carriage-return> or <newline>, *count* new lines shall be appended to the current  
38972 line. All but the last of these lines shall be empty. *count* characters at and after the cursor shall be  
38973 discarded, and any remaining characters after the cursor in the current line shall be moved to the  
38974 last of the new lines. If the **autoindent** edit option is set, they shall be preceded by the same  
38975 number of **autoindent** characters found on the line from which the command was executed.38976 *Current line:* Unchanged unless the replacement character is a <carriage-return> or <newline>  
38977 character, in which case it shall be set to line + *count*.38978 *Current column:* Set to the last column position on which a portion of the last replaced character  
38979 is displayed, or if the replacement character caused new lines to be created, set to non-<blank>.38980 **Replace Characters**38981 *Synopsis:* R38982 Enter text input mode at the current cursor position. A *count* shall cause the input text to be  
38983 appended *count* - 1 more times to the end of the input.38984 *Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi**  
38985 (on page 3235)).

**38986      Substitute Character**

38987      *Synopsis:*    [*buffer*][*count*] s

38988      This command shall be equivalent to the *vi* command:

38989      [*buffer*][*count*] c<space>

**38990      Substitute Lines**

38991      *Synopsis:*    [*buffer*][*count*] S

38992      This command shall be equivalent to the *vi* command:

38993      [*buffer*][*count*] c\_

**38994      Move Cursor to Before Character (Forward)**

38995      *Synopsis:*    [*count*] t *character*

38996      It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

38997      If used as a motion command:

- 38998      1. The text region shall be from the cursor up to but not including the *count*th occurrence of  
38999      the specified character after the cursor.
- 39000      2. Any text copied to a buffer shall be in character mode.

39001      If not used as a motion command:

39002      *Current line:* Unchanged.

39003      *Current column:* Set to the last column in which any portion of the character before the *count*th  
39004      occurrence of the specified character after the cursor appears in the line.

**39005      Move Cursor to After Character (Reverse)**

39006      *Synopsis:*    [*count*] T *character*

39007      It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

39008      If used as a motion command:

- 39009      1. If the character before the cursor is the specified character, it shall be an error.
- 39010      2. The text region shall be from the character before the cursor up to but not including the  
39011      *count*th occurrence of the specified character before the cursor.
- 39012      3. Any text copied to a buffer shall be in character mode.

39013      If not used as a motion command:

39014      *Current line:* Unchanged.

39015      *Current column:* Set to the last column in which any portion of the character after the *count*th  
39016      occurrence of the specified character before the cursor appears in the line.

- 39017           **Undo**
- 39018           *Synopsis:*     u
- 39019           This command shall be equivalent to the *ex* **undo** command except that the current line and  
39020           current column shall be set as follows:
- 39021           *Current line:* Set to the first line added or changed if any; otherwise, move to the line preceding  
39022           any deleted text if one exists; otherwise, move to line 1.
- 39023           *Current column:* If undoing an *ex* command, set to the first non-<blank> character.  
39024           Otherwise, if undoing a text input command:
- 39025           1. If the command was an **C**, **c**, **O**, **o**, **R**, **S**, or **s** command, the current column shall be set to  
39026           the value it held when the text input command was entered.
- 39027           2. Otherwise, set to the last column in which any portion of the first character after the  
39028           deleted text is displayed, or, if no characters follow the text deleted from this line, set to the  
39029           last column in which any portion of the last character in the line is displayed, or 1 if the line  
39030           is empty.
- 39031           Otherwise, if a single line was modified (that is, not added or deleted) by the **u** command:
- 39032           1. If text was added or changed, set to the last column in which any portion of the first  
39033           character added or changed is displayed.
- 39034           2. If text was deleted, set to the last column in which any portion of the first character after  
39035           the deleted text is displayed, or, if no characters follow the deleted text, set to the last  
39036           column in which any portion of the last character in the line is displayed, or 1 if the line is  
39037           empty.
- 39038           Otherwise, set to non-<blank>.
- 39039           **Undo Current Line**
- 39040           *Synopsis:*     U
- 39041           Restore the current line to its state immediately before the most recent time that it became the  
39042           current line.
- 39043           *Current line:* Unchanged.
- 39044           *Current column:* Set to the first column in the line in which any portion of the first character in  
39045           the line is displayed.
- 39046           **Move to Beginning of Word**
- 39047           *Synopsis:*     [*count*] w
- 39048           With the exception that words are used as the delimiter instead of bigwords, this command shall  
39049           be equivalent to the **W** command.

39050 **Move to Beginning of Bigword**39051 *Synopsis:* [ *count* ] W39052 If the edit buffer is empty, it shall be an error. If there are less than *count* bigwords between the  
39053 cursor and the end of the edit buffer, *count* shall be adjusted to move the cursor to the last  
39054 bigword in the edit buffer.

39055 If used as a motion command:

- 39056 1. If the associated command is *c*, *count* is 1, and the cursor is on a <blank> character, the  
39057 region of text shall be the current character and no further action shall be taken.
- 39058 2. If there are less than *count* bigwords between the cursor and the end of the edit buffer, then  
39059 the command shall succeed, and the region of text shall include the last character of the  
39060 edit buffer.
- 39061 3. If there are <blank> characters or an end-of-line that precede the *count*th bigword, and the  
39062 associated command is *c*, the region of text shall be up to and including the last character  
39063 before the preceding <blank> characters or end-of-line.
- 39064 4. If there are <blank> characters or an end-of-line that precede the bigword, and the  
39065 associated command is *d* or *y*, the region of text shall be up to and including the last  
39066 <blank> character before the start of the bigword or end-of-line.
- 39067 5. Any text copied to a buffer shall be in character mode.

39068 If not used as a motion command:

- 39069 1. If the cursor is on the last character of the edit buffer, it shall be an error.

39070 *Current line:* Set to the line containing the current column.39071 *Current column:* Set to the last column in which any part of the first character of the *count*th next  
39072 bigword is displayed.39073 **Delete Character at Cursor**39074 *Synopsis:* [ *buffer* ] [ *count* ] x39075 Delete the *count* characters at and after the current character into *buffer*, if specified, and into the  
39076 unnamed buffer.39077 If the line is empty, it shall be an error. If there are less than *count* characters at and after the  
39078 cursor on the current line, *count* shall be adjusted to the number of characters at and after the  
39079 cursor.39080 *Current line:* Unchanged.39081 *Current column:* If the line is empty, set to column position 1. Otherwise, if there were *count* or  
39082 less characters at and after the cursor on the current line, set to the last column that displays any  
39083 part of the last character of the line. Otherwise, unchanged.

39084 **Delete Character Before Cursor**39085 *Synopsis:* `[buffer][count] X`39086 Delete the *count* characters before the current character into *buffer*, if specified, and into the  
39087 unnamed buffer.39088 If there are no characters before the current character on the current line, it shall be an error. If  
39089 there are less than *count* previous characters on the current line, *count* shall be adjusted to the  
39090 number of previous characters on the line.39091 *Current line:* Unchanged.39092 *Current column:* Set to (*current column* – *the width of the deleted characters*).39093 **Yank**39094 *Synopsis:* `[buffer][count] y motion`39095 Copy (yank) the region of text into *buffer*, if specified, and into the unnamed buffer.

39096 If the motion command is the y command repeated:

- 39097 1. The buffer shall be in line mode.
- 39098 2. If there are less than *count* – 1 lines after the current line in the edit buffer, it shall be an  
39099 error.
- 39100 3. The text region shall be from the current line up to and including the next *count* – 1 lines.

39101 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

39102 *Current line:* If the motion was from the current cursor position toward the end of the edit  
39103 buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region  
39104 specified by the motion command.39105 *Current column:*

- 39106 1. If the motion was from the current cursor position toward the end of the edit buffer,  
39107 unchanged.
- 39108 2. Otherwise, if the current line is empty, set to column position 1.
- 39109 3. Otherwise, set to the last column that displays any part of the first character in the file that  
39110 is part of the text region specified by the motion command.

39111 **Yank Current Line**39112 *Synopsis:* `[buffer][count] Y`39113 This command shall be equivalent to the *vi* command:39114 `[buffer][count] y_`

39115 **Redraw Window**

39116 If in open mode, the **z** command shall have the Synopsis:

39117 *Synopsis:*     [ *count* ] **z**

39118 If *count* is not specified, it shall default to the **window** edit option  $-1$ . The **z** command shall be  
 39119 equivalent to the *ex z* command, with a type character of = and a *count* of *count*  $-2$ , except that  
 39120 the current line and current column shall be set as follows, and the **window** edit option shall not  
 39121 be affected. If the calculation for the *count* argument would result in a negative number, the  
 39122 *count* argument to the *ex z* command shall be zero. A blank line shall be written after the last line  
 39123 is written.

39124 *Current line:* Unchanged.

39125 *Current column:* Unchanged.

39126 If not in open mode, the **z** command shall have the following Synopsis:

39127 *Synopsis:*     [ *line* ] **z** [ *count* ] *character*

39128 If *line* is not specified, it shall default to the current line. If *line* is specified, but is greater than the  
 39129 number of lines in the edit buffer, it shall default to the number of lines in the edit buffer.

39130 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in the  
 39131 *ex window* command), and the screen shall be redrawn.

39132 *line* shall be placed as specified by the following characters:

39133 <newline>, <carriage-return>

39134     Place the beginning of the line on the first line of the display.

39135     .   Place the beginning of the line in the center of the display. The middle line of the display  
 39136 shall be calculated as described for the **M** command.

39137     –   Place an unspecified portion of the line on the last line of the display.

39138     +   If *line* was specified, equivalent to the <newline> case. If *line* was not specified, display a  
 39139 screen where the first line of the display shall be (current last line)  $+1$ . If there are no lines  
 39140 after the last line in the display, it shall be an error.

39141     ^   If *line* was specified, display a screen where the last line of the display shall contain an  
 39142 unspecified portion of the first line of a display that had an unspecified portion of the  
 39143 specified line on the last line of the display. If this calculation results in a line before the  
 39144 beginning of the edit buffer, display the first screen of the edit buffer.

39145     Otherwise, display a screen where the last line of the display shall contain an unspecified  
 39146 portion of (current first line  $-1$ ). If this calculation results in a line before the beginning of  
 39147 the edit buffer, it shall be an error.

39148 *Current line:* If *line* and the ' ^ ' character were specified:

39149     1. If the first screen was displayed as a result of the command attempting to display lines  
 39150 before the beginning of the edit buffer: if the first screen was already displayed,  
 39151 unchanged; otherwise, set to (current first line  $-1$ ).

39152     2. Otherwise, set to the last line of the display.

39153 If *line* and the ' + ' character were specified, set to the first line of the display.

39154 Otherwise, if *line* was specified, set to *line*.

39155 Otherwise, unchanged.

39156 *Current column*: Set to non-`<blank>`.

## 39157 **Exit**

39158 *Synopsis*:     ZZ

39159 This command shall be equivalent to the `ex xit` command with no addresses, trailing `!`, or file  
39160 name (see the `ex xit` command).

## 39161 **Input Mode Commands in vi**

39162 In text input mode, the current line shall consist of zero or more of the following categories:

39163     1. Characters preceding the text input entry point

39164         Characters in this category shall not be modified during text input mode.

39165     2. **autoindent** characters

39166         **autoindent** characters shall be automatically inserted into each line that is created in text  
39167 input mode, either as a result of entering a `<newline>` character or `<carriage-return>`  
39168 character while in text input mode, or as an effect of the command itself; for example, `O` or  
39169 `o` (see the `ex autoindent` command), as if entered by the user.

39170         It shall be possible to erase **autoindent** characters with the `<control>-D` command; it is  
39171 unspecified whether they can be erased by `<control>-H`, `<control>-U`, and `<control>-W`  
39172 characters. Erasing any **autoindent** character turns the glyph into erase-columns and  
39173 deletes the character from the edit buffer, but does not change its representation on the  
39174 screen.

39175     3. Text input characters

39176         Text input characters are the characters entered by the user. Erasing any text input  
39177 character turns the glyph into erase-columns and deletes the character from the edit buffer,  
39178 but does not change its representation on the screen.

39179         Each text input character entered by the user (that does not have a special meaning) shall  
39180 be treated as follows:

39181         a. The text input character shall be appended to the last character in the edit buffer  
39182 from the first, second, or third categories.

39183         b. If there are no erase-columns on the screen, the text input command was the **R**  
39184 command, and characters in the fifth category from the original line follow the  
39185 cursor, the next such character shall be deleted from the edit buffer. If the **slowopen**  
39186 edit option is not set, the corresponding glyph on the screen shall become erase-  
39187 columns.

39188         c. If there are erase-columns on the screen, as many columns as they occupy, or as are  
39189 necessary, shall be overwritten to display the text input character. (If only part of a  
39190 multi-column glyph is overwritten, the remainder shall be left on the screen, and  
39191 continue to be treated as erase-columns; it is unspecified whether the remainder of  
39192 the glyph is modified in any way.)

39193         d. If additional screen columns are needed to display the text input character:

39194             1. If the **slowopen** edit option is set, the text input characters shall be displayed  
39195 on subsequent screen columns, overwriting any characters displayed in those  
39196 columns.

39197                   2. Otherwise, any characters currently displayed on or after the column on the  
39198 screen where the text input character is to be displayed shall be pushed ahead  
39199 the number of screen columns necessary to display the rest of the text input  
39200 character.

39201                   4. Erase-columns

39202 Erase-columns are not logically part of the edit buffer, appearing only on the screen, and  
39203 may be overwritten on the screen by subsequent text input characters. When text input  
39204 mode ends, all erase-columns shall no longer appear on the screen.

39205 Erase-columns are initially the region of text specified by the **c** command ( see **Change** (on  
39206 page 3221)) however, erasing **autoindent** or text input characters causes the glyphs of the  
39207 erased characters to be treated as erase-columns.

39208                   5. Characters following the text region for the **c** command, or the text input entry point for all  
39209 other commands

39210 Characters in this category shall not be modified during text input mode, except as  
39211 specified in category 3.b. for the **R** text input command, or as <blank> characters deleted  
39212 when a <newline> character or <carriage-return> character is entered.

39213 It is unspecified whether it is an error to attempt to erase past the beginning of a line that was  
39214 created by the entry of a <newline> or <carriage-return> character during text input mode. If it  
39215 is not an error, the editor shall behave as if the erasing character was entered immediately after  
39216 the last text input character entered on the previous line, and all of the characters on the current  
39217 line shall be treated as erase-columns.

39218 When text input mode is entered, or after a text input mode character is entered (except as  
39219 specified for the special characters below), the cursor shall be positioned as follows:

- 39220                   1. On the first column that displays any part of the first erase-column, if one exists
- 39221                   2. Otherwise, if the **slowopen** edit option is set, on the first screen column after the last  
39222 character in the first, second, or third categories, if one exists
- 39223                   3. Otherwise, the first column that displays any part of the first character in the fifth category,  
39224 if one exists
- 39225                   4. Otherwise, the screen column after the last character in the first, second, or third  
39226 categories, if one exists
- 39227                   5. Otherwise, on column position 1

39228 The characters that are updated on the screen during text input mode are unspecified, other than  
39229 that the last text input character shall always be updated, and, if the **slowopen** edit option is not  
39230 set, the current cursor character shall always be updated.

39231 The following specifications are for command characters entered during text input mode.

39232                   **NUL**

39233                   *Synopsis:*       NUL

39234 If the first character of the text input is a NUL, the most recently input text shall be input as if  
39235 entered by the user, and then text input mode shall be exited. The text shall be input literally;  
39236 that is, characters are neither macro or abbreviation expanded, nor are any characters interpreted  
39237 in any special manner. It is unspecified whether implementations shall support more than 256  
39238 bytes of remembered input text.



39239 **<control>-D**39240 *Synopsis:* `<control>-D`39241 The `<control>-D` character shall have no special meaning when in text input mode for a line-  
39242 oriented command (see **Command Descriptions in vi** (on page 3201)).

39243 This command need not be supported on block-mode terminals.

39244 If the cursor does not follow an **autoindent** character, or an **autoindent** character and a `'0'` or  
39245 `'^'` character:39246 1. If the cursor is in column position 1, the `<control>-D` character shall be discarded and no  
39247 further action taken.39248 2. Otherwise, the `<control>-D` character shall have no special meaning.39249 If the last input character was a `'0'`, the cursor shall be moved to column position 1.39250 Otherwise, if the last input character was a `'^'`, the cursor shall be moved to column position 1.  
39251 In addition, the **autoindent** level for the next input line shall be derived from the same line from  
39252 which the **autoindent** level for the current input line was derived.39253 Otherwise, the cursor shall be moved back to the column after the previous shiftwidth (see the  
39254 *ex* **shiftwidth** command) boundary.39255 All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
39256 cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on  
39257 page 3235).39258 *Current line:* Unchanged.39259 *Current column:* Set to 1 if the `<control>-D` was preceded by a `'^'` or `'0'`; otherwise, set to  
39260  $(\text{column} - 1) - ((\text{column} - 2) \% \text{shiftwidth})$ .39261 **<control>-H**39262 *Synopsis:* `<control>-H`39263 If in text input mode for a line-oriented command, and there are no characters to erase, text  
39264 input mode shall be terminated, no further action shall be done for this command, and the  
39265 current line and column shall be unchanged.39266 If there are characters other than **autoindent** characters that have been input on the current line  
39267 before the cursor, the cursor shall move back one character.39268 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is  
39269 implementation-defined whether the `<control>-H` command is an error or if the cursor moves  
39270 back one **autoindent** character.39271 Otherwise, if the cursor is in column position 1 and there are previous lines that have been input,  
39272 it is implementation-defined whether the `<control>-H` command is an error or if it is equivalent  
39273 to entering `<control>-H` after the last input character on the previous input line.

39274 Otherwise, it shall be an error.

39275 All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
39276 cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on  
39277 page 3235).39278 The current erase character (see *stty*) shall cause an equivalent action to the `<control>-H`  
39279 command, unless the previously inserted character was a backslash, in which case it shall be as

39280 if the literal current erase character had been inserted instead of the backslash.

39281 *Current line:* Unchanged, unless previously input lines are erased, in which case it shall be set to  
39282 line -1.

39283 *Current column:* Set to the first column that displays any portion of the character backed up  
39284 over.

39285 **<newline>**

39286 *Synopsis:* <newline>  
39287 <carriage-return>  
39288 <control>-J  
39289 <control>-M

39290 If input was part of a line-oriented command, text input mode shall be terminated and the  
39291 command shall continue execution with the input provided.

39292 Otherwise, terminate the current line. If there are no characters other than **autoindent** characters  
39293 on the line, all characters on the line shall be discarded. Otherwise, it is unspecified whether the  
39294 **autoindent** characters in the line are modified by entering these characters.

39295 Continue text input mode on a new line appended after the current line. If the **slowopen** edit  
39296 option is set, the lines on the screen below the current line shall not be pushed down, but the  
39297 first of them shall be cleared and shall appear to be overwritten. Otherwise, the lines of the  
39298 screen below the current line shall be pushed down.

39299 If the **autoindent** edit option is set, an appropriate number of **autoindent** characters shall be  
39300 added as a prefix to the line as described by the *ex autoindent* edit option.

39301 All columns after the cursor that are erase-columns (as described in **Input Mode Commands in**  
39302 **vi** (on page 3235)) shall be discarded.

39303 If the **autoindent** edit option is set, all <blank> characters immediately following the cursor shall  
39304 be discarded.

39305 All remaining characters after the cursor shall be transferred to the new line, positioned after any  
39306 **autoindent** characters.

39307 *Current line:* Set to current line +1.

39308 *Current column:* Set to the first column that displays any portion of the first character after the  
39309 **autoindent** characters on the new line, if any, or the first column position after the last  
39310 **autoindent** character, if any, or column position 1.

39311 **<control>-T**

39312 *Synopsis:* <control>-T

39313 The <control>-T character shall have no special meaning when in text input mode for a line-  
39314 oriented command (see **Command Descriptions in vi** (on page 3201)).

39315 This command need not be supported on block-mode terminals.

39316 Behave as if the user entered the minimum number of <blank> characters necessary to move the  
39317 cursor forward to the column position after the next **shiftwidth** (see the *ex shiftwidth*  
39318 command) boundary.

39319 *Current line:* Unchanged.

39320 *Current column*: Set to  $column + \mathbf{shiftwidth} - ((column - 1) \% \mathbf{shiftwidth})$ .

39321 **<control>-U**

39322 *Synopsis*: `<control>-U`

39323 If there are characters other than **autoindent** characters that have been input on the current line  
39324 before the cursor, the cursor shall move to the first character input after the **autoindent**  
39325 characters.

39326 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is  
39327 implementation-defined whether the `<control>-U` command is an error or if the cursor moves to  
39328 the first column position on the line.

39329 Otherwise, if the cursor is in column position 1 and there are previous lines that have been input,  
39330 it is implementation-defined whether the `<control>-U` command is an error or if it is equivalent  
39331 to entering `<control>-U` after the last input character on the previous input line.

39332 Otherwise, it shall be an error.

39333 All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
39334 cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on  
39335 page 3235).

39336 The current *kill* character (see *stty*) shall cause an equivalent action to the `<control>-U`  
39337 command, unless the previously inserted character was a backslash, in which case it shall be as  
39338 if the literal current *kill* character had been inserted instead of the backslash.

39339 *Current line*: Unchanged, unless previously input lines are erased, in which case it shall be set to  
39340 line  $-1$ .

39341 *Current column*: Set to the first column that displays any portion of the last character backed up  
39342 over.

39343 **<control>-V**

39344 *Synopsis*: `<control>-V`

39345 `<control>-Q`

39346 Allow the entry of any subsequent character, other than `<control>-J` or the `<newline>` character,  
39347 as a literal character, removing any special meaning that it may have to the editor in text input  
39348 mode. If a `<control>-V` or `<control>-Q` is entered before a `<control>-J` or `<newline>` character,  
39349 the `<control>-V` or `<control>-Q` character shall be discarded, and the `<control>-J` or `<newline>`  
39350 shall behave as described in the `<newline>` command character during input mode.

39351 For purposes of the display only, the editor shall behave as if a `'^'` character was entered, and  
39352 the cursor shall be positioned as if overwriting the `'^'` character. When a subsequent character  
39353 is entered, the editor shall behave as if that character was entered instead of the original  
39354 `<control>-V` or `<control>-Q` character.

39355 *Current line*: Unchanged.

39356 *Current column*: Unchanged.

- 39357 **<control>-W**
- 39358 *Synopsis:* `<control>-W`
- 39359 If there are characters other than **autoindent** characters that have been input on the current line  
39360 before the cursor, the cursor shall move back over the last word preceding the cursor (including  
39361 any `<blank>` characters between the end of the last word and the current cursor); the cursor shall  
39362 not move to before the first character after the end of any **autoindent** characters.
- 39363 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is  
39364 implementation-defined whether the `<control>-W` command is an error or if the cursor moves to  
39365 the first column position on the line.
- 39366 Otherwise, if the cursor is in column position 1 and there are previous lines that have been input,  
39367 it is implementation-defined whether the `<control>-W` command is an error or if it is equivalent  
39368 to entering `<control>-W` after the last input character on the previous input line.
- 39369 Otherwise, it shall be an error.
- 39370 All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
39371 cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on  
39372 page 3235).
- 39373 *Current line:* Unchanged, unless previously input lines are erased, in which case it shall be set to  
39374 line -1.
- 39375 *Current column:* Set to the first column that displays any portion of the last character backed up  
39376 over.
- 39377 **<ESC>**
- 39378 *Synopsis:* `<ESC>`
- 39379 If input was part of a line-oriented command:
- 39380 1. If *interrupt* was entered, text input mode shall be terminated and the editor shall return to  
39381 command mode. The terminal shall be alerted.
- 39382 **Notes to Reviewers**
- 39383 *This section with side shading will not appear in the final copy. - Ed.*
- 39384 D3, XCU, ERN 274 says the character ESC is not an interrupt character, so why is point 1  
39385 here? This will need to be revisited when .2b is approved. I believe this is covered in  
39386 Rationale; see later.
- 39387 2. If `<ESC>` was entered, text input mode shall be terminated and the command shall  
39388 continue execution with the input provided.
- 39389 Otherwise, terminate text input mode and return to command mode.
- 39390 Any **autoindent** characters entered on newly created lines that have no other characters shall be  
39391 deleted.
- 39392 Any leading **autoindent** and `<blank>` characters on newly created lines shall be rewritten to be  
39393 the minimum number of `<blank>` characters possible.
- 39394 The screen shall be redisplayed as necessary to match the contents of the edit buffer.
- 39395 *Current line:* Unchanged.

39396 *Current column:*

- 39397 1. If there are text input characters on the current line, the column shall be set to the last  
39398 column where any portion of the last text input character is displayed.
- 39399 2. Otherwise, if a character is displayed in the current column, unchanged.
- 39400 3. Otherwise, set to column position 1.

#### 39401 **EXIT STATUS**

39402 The following exit values shall be returned:

- 39403 0 Successful completion.
- 39404 >0 An error occurred.

#### 39405 **CONSEQUENCES OF ERRORS**

39406 When any error is encountered and the standard input is not a terminal device file, *vi* shall not  
39407 write the file or return to command or text input mode, and shall terminate with a non-zero exit  
39408 status.

39409 Otherwise, when an unrecoverable error is encountered it shall be equivalent to a SIGHUP  
39410 asynchronous event.

39411 Otherwise, when an error is encountered, the editor shall behave as specified in **Command**  
39412 **Descriptions in vi** (on page 3201).

#### 39413 **APPLICATION USAGE**

39414 None.

#### 39415 **EXAMPLES**

39416 None.

#### 39417 **RATIONALE**

39418 See the RATIONALE for *ex* for more information on *vi*. Major portions of the *vi* utility  
39419 specification point to *ex* to avoid inadvertent divergence. While *ex* and *vi* have historically been  
39420 implemented as a single utility, this is not required by IEEE Std. 1003.1-200x.

39421 It is recognized that portions of *vi* would be difficult, if not impossible, to implement  
39422 satisfactorily on a block-mode terminal, or a terminal without any form of cursor addressing,  
39423 thus it is not a mandatory requirement that such features should work on all terminals. It is the  
39424 intention, however, that a *vi* implementation should provide the full set of capabilities on all  
39425 terminals capable of supporting them.

39426 Historically, *vi* exited immediately if the standard input was not a terminal.  
39427 IEEE Std. 1003.1-200x permits, but does not require, this behavior. An end-of-file condition is not  
39428 equivalent to an end-of-file character. A common end-of-file character, <control>-D, is  
39429 historically a *vi* command.

39430 The text in the STANDARD OUTPUT section reflects the usage of the verb *display* in this section;  
39431 some implementations of *vi* use standard output to write to the terminal, but  
39432 IEEE Std. 1003.1-200x does not require that to be the case.

39433 Historically, implementations reverted to open mode if the terminal was incapable of  
39434 supporting full visual mode. IEEE Std. 1003.1-200x requires this behavior. Historically, the open  
39435 mode of *vi* behaved roughly equivalently to the visual mode, with the exception that only a  
39436 single physical line from the edit buffer was kept current at any time. This line was normally  
39437 displayed on the next-to-last line of a terminal with cursor addressing (and the last line  
39438 performed its normal visual functions for line-oriented commands and messages). In addition,  
39439 some few commands behaved differently in open mode than in visual mode.

39440 IEEE Std. 1003.1-200x requires conformance to historical practice.

39441 Historically, *ex* and *vi* implementations have expected text to proceed in the usual  
39442 European/Latin order of left to right, top to bottom. There is no requirement in  
39443 IEEE Std. 1003.1-200x that this be the case. The specification was deliberately written using  
39444 words like “before”, “after”, “first”, and “last” in order to permit implementations to support  
39445 the natural text order of the language.

39446 Historically, lines past the end of the edit buffer were marked with single tilde (‘~’) characters;  
39447 that is, if the one-based display was 20 lines in length, and the last line of the file was on line one,  
39448 then lines 2-20 would contain only a single ‘~’ character.

39449 Historically, the *vi* editor attempted to display only complete lines at the bottom of the screen (it  
39450 did display partial lines at the top of the screen). If a line was too long to fit in its entirety at the  
39451 bottom of the screen, the screen lines where the line would have been displayed were displayed  
39452 as single ‘@’ characters, instead of displaying part of the line. IEEE Std. 1003.1-200x permits, but  
39453 does not require, this behavior. Implementations are encouraged to attempt always to display a  
39454 complete line at the bottom of the screen when doing scrolling or screen positioning by physical  
39455 lines.

39456 Historically, lines marked with ‘@’ were also used to minimize output to dumb terminals over  
39457 slow lines; that is, changes local to the cursor were updated, but changes to lines on the screen  
39458 that were not close to the cursor were simply marked with an ‘@’ sign instead of being updated  
39459 to match the current text. IEEE Std. 1003.1-200x permits, but does not require this feature  
39460 because it is used ever less frequently as terminals become smarter and connections are faster.

#### 39461 Initialization in *ex* and *vi*

39462 Historically, *vi* always had a line in the edit buffer, even if the edit buffer was “empty”. For  
39463 example:

- 39464 1. The *ex* command = executed from visual mode wrote “1” when the buffer was empty.
- 39465 2. Writes from visual mode of an empty edit buffer wrote files of a single character (a  
39466 <newline> character), while writes from *ex* mode of an empty edit buffer wrote empty  
39467 files.
- 39468 3. Put and read commands into an empty edit buffer left an empty line at the top of the edit  
39469 buffer.

39470 For consistency, IEEE Std. 1003.1-200x does not permit any of these behaviors.

39471 Historically, *vi* did not always return the terminal to its original modes; for example, ICRNL was  
39472 modified if it was not originally set. IEEE Std. 1003.1-200x does not permit this behavior.

#### 39473 Command Descriptions in *vi*

39474 Motion commands are among the most complicated aspects of *vi* to describe. With some  
39475 exceptions, the text region and buffer type effect of a motion command on a *vi* command are  
39476 described on a case-by-case basis. The descriptions of text regions in IEEE Std. 1003.1-200x are  
39477 not intended to imply direction; that is, an inclusive region from line *n* to line *n*+5 is identical to  
39478 a region from line *n*+5 to line *n*. This is of more than academic interest—movements to marks  
39479 can be in either direction, and, if the **wrapsan** option is set, so can movements to search points.  
39480 Historically, lines are always stored into buffers in text order; that is, from the start of the edit  
39481 buffer to the end. IEEE Std. 1003.1-200x requires conformance to historical practice.

39482 Historically, command counts were applied to any associated motion, and were multiplicative  
39483 to any supplied motion count. For example, **2cw** is the same as **c2w**, and **2c3w** is the same as

39484 **c6w**. IEEE Std. 1003.1-200x requires this behavior. Historically, *vi* commands that used  
39485 bigwords, words, paragraphs, and sentences as objects treated groups of empty lines, or lines  
39486 that contained only <blank> characters, inconsistently. Some commands treated them as a single  
39487 entity, while others treated each line separately. For example, the **w**, **W**, and **B** commands treated  
39488 groups of empty lines as individual words; that is, the command would move the cursor to each  
39489 new empty line. The **e** and **E** commands treated groups of empty lines as a single word; that is,  
39490 the first use would move past the group of lines. The **b** command would just beep at the user, or  
39491 if done from the start of the line as a motion command, fail in unexpected ways. If the lines  
39492 contained only (or ended with) <blank> characters, the **w** and **W** commands would just beep at  
39493 the user, the **E** and **e** commands would treat the group as a single word, and the **B** and **b**  
39494 commands would treat the lines as individual words. For consistency and simplicity of  
39495 specification, IEEE Std. 1003.1-200x requires that all *vi* commands treat groups of empty or  
39496 <blank> character-filled lines as a single entity, and that movement through lines ending with  
39497 <blank> characters be consistent with other movements.

39498 Historically, *vi* documentation indicated that any number of double quotes were skipped after  
39499 punctuation marks at sentence boundaries; however, implementations only skipped single  
39500 quotes. IEEE Std. 1003.1-200x requires both to be skipped.

39501 Historically, the first and last characters in the edit buffer were word boundaries. This historical  
39502 practice is required by IEEE Std. 1003.1-200x.

39503 Historically, *vi* attempted to update the minimum number of columns on the screen possible,  
39504 which could lead to misleading information being displayed. IEEE Std. 1003.1-200x makes no  
39505 requirements other than that the current character being entered is displayed correctly, leaving  
39506 all other decisions in this area up to the implementation.

39507 Historically, lines were arbitrarily folded between columns of any characters that required  
39508 multiple column positions on the screen, with the exception of tabs, which terminated at the  
39509 right-hand margin. IEEE Std. 1003.1-200x permits the former and requires the latter.  
39510 Implementations that do not arbitrarily break lines between columns of characters that occupy  
39511 multiple column positions should not permit the cursor to rest on a column that does not  
39512 contain any part of a character.

39513 The historical *vi* had a problem in that all movements were by physical lines, not by logical, or  
39514 screen, lines. This is often the right thing to do; for example, single line movements, such as **j** or  
39515 **k**, should work on physical lines. Commands like **dj**, or **j.**, where **.** is a change command, only  
39516 make sense for physical lines. It is not, however, the right thing to do for screen motion or  
39517 scrolling commands like <control>-D, <control>-F, and **H**. If the window is fairly small, using  
39518 physical lines in these cases can result in completely random motion; for example, **1<control>-D**  
39519 can result in a completely changed screen, without any overlap. This is clearly not what the user  
39520 wanted. The problem is even worse in the case of the **H**, **L**, and **M** commands—as they position  
39521 the cursor at the first non-<blank> character of the line, they may all refer to the same location in  
39522 large lines, and will result in no movement at all.

39523 In addition, if the line is larger than the screen, using physical lines can make it impossible to  
39524 display parts of the line—there are not any commands that do not display the beginning of the  
39525 line in historical *vi*, and if both the beginning and end of the line cannot be on the screen at  
39526 the same time, the user suffers. Finally, the page and half-page scrolling commands historically  
39527 moved to the first non-<blank> character in the new line. If the line is approximately the same  
39528 size as the screen, this is inadequate because the cursor before and after a <control>-D command  
39529 will refer to the same location on the screen.

39530 Implementations of *ex* and *vi* exist that do not have these problems because the relevant  
39531 commands (<control>-B, <control>-D, <control>-F, <control>-U, <control>-Y, <control>-E, **H**, **L**,  
39532 and **M**) operate on logical screen lines, not physical edit buffer lines.

39533 IEEE Std. 1003.1-200x does not permit this behavior by default because the standard developers  
 39534 believed that users would find it too confusing. However, historical practice has been relaxed.  
 39535 For example, *ex* and *vi* historically attempted, albeit sometimes unsuccessfully, to never put part  
 39536 of a line on the last lines of a screen; for example, if a line would not fit in its entirety, no part of  
 39537 the line was displayed, and the screen lines corresponding to the line contained single '@'  
 39538 characters. This behavior is permitted, but not required by IEEE Std. 1003.1-200x, so that it is  
 39539 possible for implementations to support long lines in small screens more reasonably without  
 39540 changing the commands to be logically (instead of physically) oriented. IEEE Std. 1003.1-200x  
 39541 also permits implementations to refuse to edit any edit buffer containing a line that will not fit  
 39542 on the screen in its entirety.

39543 The display area (for example, the value of the **window** edit option) has historically been  
 39544 “grown”, or expanded, to display new text when local movements are done in displays where  
 39545 the number of lines displayed is less than the maximum possible. Expansion has historically  
 39546 been the first choice, when the target line is less than the maximum possible expansion value  
 39547 away. Scrolling has historically been the next choice, done when the target line is less than half a  
 39548 display away, and otherwise, the screen was redrawn. There were exceptions, however, in that  
 39549 *ex* commands generally always caused the screen to be redrawn. IEEE Std. 1003.1-200x does not  
 39550 specify a standard behavior because there may be external issues, such as connection speed, the  
 39551 number of characters necessary to redraw as opposed to scroll, or terminal capabilities that  
 39552 implementations will have to accommodate.

39553 The current line in IEEE Std. 1003.1-200x maps one-to-one to a physical line in the file. The  
 39554 current column does not. There are two different column values that are described by  
 39555 IEEE Std. 1003.1-200x. The first is the current column value as set by many of the *vi* commands.  
 39556 This value is remembered for the lifetime of the editor. The second column value is the actual  
 39557 position on the screen where the cursor rests. The two are not always the same. For example,  
 39558 when the cursor is backed by a multi-column character, the actual cursor position on the screen  
 39559 has historically been the last column of the character in command mode, and the first column of  
 39560 the character in input mode.

39561 Commands that set the current line, but that do not set the current cursor value (for example, **j**  
 39562 and **k**) attempt to get as close as possible to the remembered column position, so that the cursor  
 39563 tends to restrict itself to a vertical column as the user moves around in the edit buffer.  
 39564 IEEE Std. 1003.1-200x requires conformance to historical practice, requiring that the physical  
 39565 location of the cursor on the screen be adjusted from the current column value as necessary to  
 39566 support this historical behavior.

39567 Historically, only a single line (and for some terminals, a single line minus 1 column) of  
 39568 characters could be entered by the user for the line oriented commands; that is, **:**, **!**, **/**, or **?**.  
 39569 IEEE Std. 1003.1-200x permits, but does not require, this limitation.

39570 Historically, “soft” errors in *vi* caused the terminal to be alerted, but no error message was  
 39571 displayed. As a general rule, no error message was displayed for errors in command execution  
 39572 in *vi*, when the error resulted from the user attempting an invalid or impossible action, or when  
 39573 a searched-for object was not found. Examples of soft errors included **h** at the left margin,  
 39574 <control>-B or **[[** at the beginning of the file, **2G** at the end of the file, and so on. In addition,  
 39575 errors such as **%**, **[[**, **}]**, **)**, **N**, **n**, **f**, **F**, **t**, and **T** failing to find the searched-for object were soft as well.  
 39576 Less consistently, **/** and **?** displayed an error message if the pattern was not found, **/**, **?**, **N**, and **n**  
 39577 displayed an error message if no previous regular expression had been specified, and **;** did not  
 39578 display an error message if no previous **f**, **F**, **t**, or **T** command had occurred. Also, behavior in  
 39579 this area might reasonably be based on a runtime evaluation of the speed of a network  
 39580 connection. Finally, some implementations have provided error messages for soft errors in  
 39581 order to assist naive users, based on the value of a verbose edit option. IEEE Std. 1003.1-200x  
 39582 does not list specific errors for which an error message shall be displayed. Implementations



39583 should conform to historical practice in the absence of any strong reason to diverge.

### 39584 **Page Backwards**

39585 The <control>-B and <control>-F commands historically considered it an error to attempt to  
 39586 page past the beginning or end of the file, whereas the <control>-D and <control>-U commands  
 39587 simply moved to the beginning or end of the file. For consistency, IEEE Std. 1003.1-200x requires  
 39588 the latter behavior for all four commands. All four commands still consider it an error if the  
 39589 current line is at the beginning (<control>-B, <control>-U) or end (<control>-F, <control>-D) of  
 39590 the file. Historically, the <control>-B and <control>-F commands skip two lines in order to  
 39591 include overlapping lines when a single command is entered. This makes less sense in the  
 39592 presence of a *count*, as there will be, by definition, no overlapping lines. The actual calculation  
 39593 used by historical implementations of the *vi* editor for <control>-B was:

39594  $((\text{current first line}) - \text{count} \times (\text{window edit option})) + 2$

39595 and for <control>-F was:

39596  $((\text{current first line}) + \text{count} \times (\text{window edit option})) - 2$

39597 This calculation does not work well when intermixing commands with and without counts; for  
 39598 example, **3**<control>-F is not equivalent to entering the <control>-F command three times, and is  
 39599 not reversible by entering the <control>-B command three times. For consistency with other *vi*  
 39600 commands that take counts, IEEE Std. 1003.1-200x requires a different calculation.

### 39601 **Scroll Forward**

39602 The 4BSD and System V implementations of *vi* differed on the initial value used by the **scroll**  
 39603 command. 4BSD used:

39604  $(\text{window edit option} + 1) / 2$

39605 while System V used the value of the **scroll** edit option. The System V version is specified by  
 39606 IEEE Std. 1003.1-200x because the standard developers believed that it was more intuitive and  
 39607 permitted the user a method of setting the scroll value initially without also setting the number  
 39608 of lines that are displayed.

### 39609 **Scroll Forward by Line**

39610 Historically, the <control>-E and <control>-Y commands considered it an error if the last and  
 39611 first lines, respectively, were already on the screen. IEEE Std. 1003.1-200x requires conformance  
 39612 to historical practice. Historically, the <control>-E and <control>-Y commands had no effect in  
 39613 open mode. For simplicity and consistency of specification, IEEE Std. 1003.1-200x requires that  
 39614 they behave as usual, albeit with a single line screen.

### 39615 **Clear and Redisplay**

39616 The historical <control>-L command refreshed the screen exactly as it was supposed to be  
 39617 currently displayed, replacing any '@' characters for lines that had been deleted but not  
 39618 updated on the screen with refreshed '@' characters. The intent of the <control>-L command is  
 39619 to refresh when the screen has been accidentally overwritten; for example, by a **write** command  
 39620 from another user, or modem noise.

### 39621 **Redraw Screen**

39622 The historical <control>-R command redisplayed only when necessary to update lines that had  
 39623 been deleted but not updated on the screen and that were flagged with '@' characters. There is  
 39624 no requirement that the screen be in any way refreshed if no lines of this form are currently  
 39625 displayed. IEEE Std. 1003.1-200x permits implementations to extend this command to refresh  
 39626 lines on the screen flagged with '@' characters because they are too long to be displayed in the  
 39627 current framework; however, the current line and column need not be modified.

### 39628 **Search for tagstring**

39629 Historically, the first non-<blank> character at or after the cursor was the first character, and all  
 39630 subsequent characters that were word characters, up to the end of the line, were included. For  
 39631 example, with the cursor on the leading space or on the '#' character in the text "#bar@", the  
 39632 tag was "#bar". On the character 'b' it was "bar", and on the 'a', it was "ar".  
 39633 IEEE Std. 1003.1-200x requires this behavior.

### 39634 **Replace Text with Results from Shell Command**

39635 Historically, the <, >, and ! commands considered most cursor motions other than line-oriented  
 39636 motions an error; for example, the command >/foo<CR> succeeded, while the command >|  
 39637 failed, even though the text region described by the two commands might be identical. For  
 39638 consistency, all three commands only consider entire lines and not partial lines, and the region is  
 39639 defined as any line that contains a character that was specified by the motion.

### 39640 **Move to Matching Character**

39641 Other matching characters have been left implementation-defined in order to allow extensions  
 39642 such as matching '<' and '>' for searching HTML, or #ifdef, #else, and #endif for searching C  
 39643 source.

### 39644 **Repeat Substitution**

39645 IEEE Std. 1003.1-200x requires that any c and g flags specified to the previous substitute  
 39646 command be ignored; however, the r flag may still apply, if supported by the implementation.

### 39647 **Return to Previous (Context or Section)**

39648 The [[, ]], (, ), {, and } commands are all affected by "section boundaries", but in some historical  
 39649 implementations not all of the commands recognize the same section boundaries. This is a bug,  
 39650 not a feature, and a unique section-boundary algorithm was not described for each command.  
 39651 One special case that is preserved is that the sentence command moves to the end of the last line  
 39652 of the edit buffer while the other commands go to the beginning, in order to preserve the  
 39653 traditional character cut semantics of the sentence command. Historically, vi section boundaries  
 39654 at the beginning and end of the edit buffer were the first non-<blank> character on the first and  
 39655 last lines of the edit buffer if one exists; otherwise, the last character of the first and last lines of  
 39656 the edit buffer if one exists. To increase consistency with other section locations, this has been  
 39657 simplified by IEEE Std. 1003.1-200x to the first character of the first and last lines of the edit  
 39658 buffer, or the first and the last lines of the edit buffer if they are empty.

39659 Sentence boundaries were problematic in the historical vi. They were not only the boundaries as  
 39660 defined for the section and paragraph commands, but they were the first non-<blank> character  
 39661 that occurred after those boundaries, as well. Historically, the vi section commands were  
 39662 documented as taking an optional window size as a count preceding the command. This was not  
 39663 implemented in historical versions, so IEEE Std. 1003.1-200x requires that the count repeat the  
 39664 command, for consistency with other vi commands.

39665 **Repeat**

39666 Historically, mapped commands other than text input commands could not be repeated using  
39667 the **period** command. IEEE Std. 1003.1-200x requires conformance to historical practice.

39668 The restrictions on the interpretation of special characters (for example, <control>-H) in the  
39669 repetition of text input mode commands is intended to match historical practice. For example,  
39670 given the input sequence:

```
39671 iab<control>-H<control>-H<control>-Hdef<escape>
```

39672 the user should be informed of an error when the sequence is first entered, but not during a  
39673 command repetition. The character <control>-T is specifically exempted from this restriction.  
39674 Historical implementations of *vi* ignored <control>-T characters that were input in the original  
39675 command during command repetition. IEEE Std. 1003.1-200x prohibits this behavior.

39676 **Find Regular Expression**

39677 Historically, commands did not affect the line searched to or from if the motion command was a  
39678 search (*/*, *?*, **N**, **n**) and the final position was the start/end of the line. There were some special  
39679 cases and *vi* was not consistent. IEEE Std. 1003.1-200x does not permit this behavior, for  
39680 consistency. Historical implementations permitted, but were unable to handle searches as  
39681 motion commands that wrapped (that is, due to the edit option **wrapsan**) to the original  
39682 location. IEEE Std. 1003.1-200x requires that this behavior be treated as an error.

39683 Historically, the syntax `"/RE/0"` was used to force the command to cut text in line mode.  
39684 IEEE Std. 1003.1-200x requires conformance to historical practice.

39685 Historically, in open mode, a **z** specified to a search command redisplayed the current line  
39686 instead of displaying the current screen with the current line highlighted. For consistency and  
39687 simplicity of specification, IEEE Std. 1003.1-200x does not permit this behavior.

39688 Historically, trailing **z** commands were permitted and ignored if entered as part of a search used  
39689 as a motion command. For consistency and simplicity of specification, IEEE Std. 1003.1-200x  
39690 does not permit this behavior.

39691 **Execute an ex Command**

39692 Historically, *vi* implementations restricted the commands that could be entered on the colon  
39693 command line (for example, **append** and **change**), and some other commands were known to  
39694 cause them to fail catastrophically. For consistency, IEEE Std. 1003.1-200x does not permit these  
39695 restrictions. When executing an *ex* command by entering `:`, it is not possible to enter a <newline>  
39696 character as part of the command because it is considered the end of the command. A different  
39697 approach is to enter *ex* command mode by using the *vi* **Q** command (and later resuming visual  
39698 mode with the *ex* **vi** command). In *ex* command mode, the single-line limitation does not exist.  
39699 So, for example, the following is valid:

```
39700 Q
39701 s/break here/break\
39702 here/
39703 vi
```

39704 IEEE Std. 1003.1-200x requires that, if the *ex* command overwrites any part of the screen that  
39705 would be erased by a refresh, *vi* pauses for a character from the user. Historically, this character  
39706 could be any character; for example, a character input by the user before the message appeared,  
39707 or even a mapped character. This is probably a bug, but implementations that have tried to be  
39708 more rigorous by requiring that the user enter a specific character, or that the user enter a  
39709 character after the message was displayed, have been forced by user indignation back into

39710 historical behavior. IEEE Std. 1003.1-200x requires conformance to historical practice.

#### 39711 **Shift Left (Right)**

39712 Refer to the Rationale for the ! and / commands. Historically, the < and > commands sometimes  
39713 moved the cursor to the first non-<blank> character (for example if the command was repeated  
39714 or with \_ as the motion command), and sometimes left it unchanged. IEEE Std. 1003.1-200x does  
39715 not permit this inconsistency, requiring instead that the cursor always move to the first non-  
39716 <blank> character. Historically, the < and > commands did not support buffer arguments,  
39717 although some implementations allow the specification of an optional buffer. This behavior is  
39718 neither required nor disallowed by IEEE Std. 1003.1-200x.

#### 39719 **Execute**

39720 Historically, buffers could execute other buffers, and loops, infinite and otherwise, were  
39721 possible. IEEE Std. 1003.1-200x requires conformance to historical practice. The *\*buffer* syntax of  
39722 *ex* is not required in *vi*, because it is not historical practice and has been used in some *vi*  
39723 implementations to support additional scripting languages.

#### 39724 **Reverse Case**

39725 Historically, the ~ command ignored any associated *count*, and acted only on the characters in  
39726 the current line. For consistency with other *vi* commands, IEEE Std. 1003.1-200x requires that an  
39727 associated *count* act on the next *count* characters, and that the command move to subsequent  
39728 lines if warranted by *count*, to make it possible to modify large pieces of text in a reasonably  
39729 efficient manner. There exist *vi* implementations that optionally require an associated motion  
39730 command for the ~ command. Implementations supporting this functionality are encouraged to  
39731 base it on the **tildedop** edit option and handle the text regions and cursor positioning identically  
39732 to the **yank** command.

#### 39733 **Append**

39734 Historically, *counts* specified to the **A**, **a**, **I**, and **i** commands repeated the input of the first line  
39735 *count* times, and did not repeat the subsequent lines of the input text. IEEE Std. 1003.1-200x  
39736 requires that the entire text input be repeated *count* times.

#### 39737 **Move Backward to Preceding Word**

39738 Historically, *vi* became confused if word commands were used as motion commands in empty  
39739 files. IEEE Std. 1003.1-200x requires that this be an error. Historical implementations of *vi* had a  
39740 large number of bugs in the word movement commands, and they varied greatly in behavior in  
39741 the presence of empty lines, “words” made up of a single character, and lines containing only  
39742 <blank> characters. For consistency and simplicity of specification, IEEE Std. 1003.1-200x does  
39743 not permit this behavior.

#### 39744 **Change to End-of-Line**

39745 Some historical implementations of the **C** command did not behave as described by  
39746 IEEE Std. 1003.1-200x when the **\$** key was remapped because they were implemented by  
39747 pushing the **\$** key onto the input queue and reprocessing it. IEEE Std. 1003.1-200x does not  
39748 permit this behavior. Historically, the **C**, **S**, and **s** commands did not copy replaced text into the  
39749 numeric buffers. For consistency and simplicity of specification, IEEE Std. 1003.1-200x requires  
39750 that they behave like their respective **c** commands in all respects.

**39751 Delete**

39752 Historically, lines in open mode that were deleted were scrolled up, and an @ glyph written over  
39753 the beginning of the line. In the case of terminals that are incapable of the necessary cursor  
39754 motions, the editor erased the deleted line from the screen. IEEE Std. 1003.1-200x requires  
39755 conformance to historical practice; that is, if the terminal cannot display the '@' character, the  
39756 line cannot remain on the screen.

**39757 Delete to End-of-Line**

39758 Some historical implementations of the **D** command did not behave as described by  
39759 IEEE Std. 1003.1-200x when the **\$** key was remapped because they were implemented by  
39760 pushing the **\$** key onto the input queue and reprocessing it. IEEE Std. 1003.1-200x does not  
39761 permit this behavior.

**39762 Join**

39763 An historical oddity of *vi* is that the commands **J**, **1J**, and **2J** are all equivalent.  
39764 IEEE Std. 1003.1-200x requires conformance to historical practice. The *vi* **J** command is specified  
39765 in terms of the *ex* **join** command with an *ex* command *count* value. The address correction for a  
39766 *count* that is past the end of the edit buffer is necessary for historical compatibility for both *ex*  
39767 and *vi*.

**39768 Mark Position**

39769 Historical practice is that only lowercase letters, plus '' and ''', could be used to mark a  
39770 cursor position. IEEE Std. 1003.1-200x requires conformance to historical practice, but  
39771 encourages implementations to support other characters as marks as well.

**39772 Repeat Regular Expression Find (Forward and Reverse)**

39773 Historically, the **N** and **n** commands could not be used as motion components for the **c**  
39774 command. With the exception of the **cN** command, which worked if the search crossed a line  
39775 boundary, the text region would be discarded, and the user would not be in text input mode. For  
39776 consistency and simplicity of specification, IEEE Std. 1003.1-200x does not permit this behavior.

**39777 Insert Empty Line (Below and Above)**

39778 Historically, counts to the **O** and **o** commands were used as the number of physical lines to  
39779 open, if the terminal was dumb and the **slowopen** option was not set. This was intended to  
39780 minimize traffic over slow connections and repainting for dumb terminals. IEEE Std. 1003.1-200x  
39781 does not permit this behavior, requiring that a *count* to the open command behave as for other  
39782 text input commands. This change to historical practice was made for consistency, and because a  
39783 superset of the functionality is provided by the **slowopen** edit option.

**39784 Put from Buffer (Following and Before)**

39785 Historically, *counts* to the **p** and **P** commands were ignored if the buffer was a line mode buffer,  
39786 but were (mostly) implemented as described in IEEE Std. 1003.1-200x if the buffer was a  
39787 character mode buffer. Because implementations exist that do not have this limitation, and  
39788 because pasting lines multiple times is generally useful, IEEE Std. 1003.1-200x requires that *count*  
39789 be supported for all **p** and **P** commands.

39790 Historical implementations of *vi* were widely known to have major problems in the **p** and **P**  
39791 commands, particularly when unusual regions of text were copied into the edit buffer. The  
39792 standard developers viewed these as bugs, and they are not permitted for consistency and

39793 simplicity of specification.

39794 Historically, a **P** or **p** command (or an *ex put* command executed from open or visual mode)  
39795 executed in an empty file, left an empty line as the first line of the file. For consistency and  
39796 simplicity of specification, IEEE Std. 1003.1-200x does not permit this behavior.

### 39797 **Replace Character**

39798 Historically, the **r** command did not correctly handle the *erase* and *word erase* characters as  
39799 arguments, nor did it handle an associated *count* greater than 1 with a <carriage-return>  
39800 character argument, for which it replaced *count* characters with a single <newline> character.  
39801 IEEE Std. 1003.1-200x does not permit these inconsistencies.

39802 Historically, the **r** command permitted the <control>-V escaping of entered characters, such as  
39803 <ESC> and the <carriage-return> character; however, it required two leading <control>-V  
39804 characters instead of one. IEEE Std. 1003.1-200x requires that this be changed for consistency  
39805 with the other text input commands of *vi*.

39806 Historically, it is an error to enter the **r** command if there are less than *count* characters at or after  
39807 the cursor in the line. While a reasonable and unambiguous extension would be to permit the **r**  
39808 command on empty lines, it would require that too large a *count* be adjusted to match the  
39809 number of characters at or after the cursor for consistency, which is sufficiently different from  
39810 historical practice to be avoided. IEEE Std. 1003.1-200x requires conformance to historical  
39811 practice.

### 39812 **Replace Characters**

39813 Historically, if there were **autoindent** characters in the line on which the **R** command was run,  
39814 and **autoindent** was set, the first <newline> character would be properly indented and no  
39815 characters would be replaced by the <newline> character. Each additional <newline> character  
39816 would replace *n* characters, where *n* was the number of characters that were needed to indent  
39817 the rest of the line to the proper indentation level. This behavior is a bug and is not permitted by  
39818 IEEE Std. 1003.1-200x.

### 39819 **Undo**

39820 Historical practice for cursor positioning after undoing commands was mixed. In most cases,  
39821 when undoing commands that affected a single line, the cursor was moved to the start of added  
39822 or changed text, or immediately after deleted text. However, if the user had moved from the line  
39823 being changed, the column was either set to the first non-<blank> character, returned to the  
39824 origin of the command, or remained unchanged. When undoing commands that affected  
39825 multiple lines or entire lines, the cursor was moved to the first character in the first line restored.  
39826 As an example of how inconsistent this was, a search, followed by an **o** text input command,  
39827 followed by an **undo** would return the cursor to the location where the **o** command was entered,  
39828 but a **cw** command followed by an **o** command followed by an **undo** would return the cursor to  
39829 the first non-<blank> character of the line. IEEE Std. 1003.1-200x requires the most useful of  
39830 these behaviors, and discards the least useful, in the interest of consistency and simplicity of  
39831 specification.

39832

**Yank**

39833

39834

39835

39836

39837

39838

39839

39840

39841

39842

Historically, the **yank** command did not move to the end of the motion if the motion was in the forward direction. It moved to the end of the motion if the motion was in the backward direction, except for the **\_** command, or for the **G** and **'** commands when the end of the motion was on the current line. This was further complicated by the fact that for a number of motion commands, the **yank** command moved the cursor but did not update the screen; for example, a subsequent command would move the cursor from the end of the motion, even though the cursor on the screen had not reflected the cursor movement for the **yank** command. IEEE Std. 1003.1-200x requires that all **yank** commands associated with backward motions move the cursor to the end of the motion for consistency, and specifically, to make **'** commands as motions consistent with search patterns as motions.

39843

**Yank Current Line**

39844

39845

39846

39847

Some historical implementations of the **Y** command did not behave as described by IEEE Std. 1003.1-200x when the **'\_'** key was remapped because they were implemented by pushing the **'\_'** key onto the input queue and reprocessing it. IEEE Std. 1003.1-200x does not permit this behavior.

39848

**Redraw Window**

39849

39850

39851

39852

39853

39854

Historically, the **z** command always redrew the screen. This is permitted but not required by IEEE Std. 1003.1-200x, because of the frequent use of the **z** command in macros such as **map n nz** for screen positioning, instead of its use to change the screen size. The standard developers believed that expanding or scrolling the screen offered a better interface for users. The ability to redraw the screen is preserved if the optional new window size is specified, and in the **<control>-L** and **<control>-R** commands.

39855

39856

39857

The semantics of **z^** are confusing at best. Historical practice is that the screen before the screen that ended with the specified line is displayed. IEEE Std. 1003.1-200x requires conformance to historical practice.

39858

39859

39860

39861

39862

Historically, the **z** command would not display a partial line at the top or bottom of the screen. If the partial line would normally have been displayed at the bottom of the screen, the command worked, but the partial line was replaced with **'@'** characters. If the partial line would normally have been displayed at the top of the screen, the command would fail. For consistency and simplicity of specification, IEEE Std. 1003.1-200x does not permit this behavior.

39863

39864

Historically, the **z** command with a line specification of 1 ignored the command. For consistency and simplicity of specification, IEEE Std. 1003.1-200x does not permit this behavior.

39865

39866

39867

Historically, the **z** command did not set the cursor column to the first non-**<blank>** character for the character if the first screen was to be displayed, and was already displayed. For consistency and simplicity of specification, IEEE Std. 1003.1-200x does not permit this behavior.

39868

**Input Mode Commands in vi**

39869

39870

39871

39872

39873

Historical implementations of **vi** did not permit the the user to erase more than a single line of input, or to use normal erase characters such as *line erase*, *worderase*, and *erase* to erase **autoindent** characters. As there exist implementations of **vi** that do not have these limitations, both behaviors are permitted, but only historical practice is required. In the case of these extensions, **vi** is required to pause at the **autoindent** and previous line boundaries.

39874

39875

Historical implementations of **vi** updated only the portion of the screen where the current cursor character was displayed. For example, consider the **vi** input keystrokes:

39876 iabcd<escape>0C<tab>

39877 Historically, the <tab> character would overwrite the characters "abcd" when it was displayed.  
39878 Other implementations replace only the 'a' character with the <tab> character, and then push  
39879 the rest of the characters ahead of the cursor. Both implementations have problems. The  
39880 historical implementation is probably visually nicer for the above example; however, for the  
39881 keystrokes:

39882 iabcd<ESC>0R<tab><ESC>

39883 the historical implementation results in the string "bcd" disappearing and then magically  
39884 reappearing when the <ESC> character is entered. IEEE Std. 1003.1-200x requires the former  
39885 behavior when overwriting erase-columns; that is, overwriting characters that are no longer  
39886 logically part of the edit buffer, and the latter behavior otherwise.

39887 Historical implementations of *vi* discarded the <control>-D and <control>-T characters when  
39888 they were entered at places where their command functionality was not appropriate.  
39889 IEEE Std. 1003.1-200x requires that the <control>-T functionality always be available, and that  
39890 <control>-D be treated as any other key when not operating on **autoindent** characters.

### 39891 NUL

39892 Some historical implementations of *vi* limited the number of characters entered using the NUL  
39893 input character to 256 bytes. IEEE Std. 1003.1-200x permits this limitation; however,  
39894 implementations are encouraged to remove this limit.

### 39895 <control>-D

39896 See also Rationale for the input mode command <newline>. The hidden assumptions in the  
39897 <control>-D command (and in the *vi* **autoindent** specification in general) is that <space>  
39898 characters take up a single column on the screen and that <tab> characters are comprised of an  
39899 integral number of <space> characters.

### 39900 <newline>

39901 Implementations are permitted to rewrite **autoindent** characters in the line when <newline>,  
39902 <carriage-return>, <control>-D, and <control>-T are entered, or when the **shift** commands are  
39903 used, because historical implementations have both done so and found it necessary to do so. For  
39904 example, a <control>-D when the cursor is preceded by a single <tab> character, with **tabstop**  
39905 set to 8, and **shiftwidth** set to 3, will result in the <tab> character being replaced by several  
39906 <space> characters.

### 39907 <control>-T

39908 See also the Rationale for the input mode command <newline>. Historically, <control>-T only  
39909 worked if no non-<blank> characters had yet been input in the current input line. In addition,  
39910 the characters inserted by <control>-T were treated as **autoindent** characters, and could not be  
39911 erased using normal user erase characters. Because implementations exist that do not have  
39912 these limitations, and as moving to a column boundary is generally useful, IEEE Std. 1003.1-200x  
39913 requires that both limitations be removed.



- 39914            **<control>-V**
- 39915            Historically, *vi* used **^V**, regardless of the value of the literal-next character of the terminal.  
39916            IEEE Std. 1003.1-200x requires conformance to historical practice.
- 39917            The uses described for **<control>-V** can also be accomplished with **<control>-Q**, which is useful  
39918            on terminals that use **<control>-V** for the down-arrow function. However, most historical  
39919            implementations use **<control>-Q** for the *termios* START character, so the editor will generally  
39920            not receive the **<control>-Q** unless **stty ixon** mode is set to off. (In addition, some historical  
39921            implementations of *vi* explicitly set **ixon** mode to on, so it was difficult for the user to set it to  
39922            off.) Any of the command characters described in IEEE Std. 1003.1-200x can be made ineffective  
39923            by their selection as *termios* control characters, using the *stty* utility or other methods described  
39924            in the System Interfaces volume of IEEE Std. 1003.1-200x.
- 39925            **<ESC>**
- 39926            Historically, SIGINT alerted the terminal when used to end input mode. This behavior is  
39927            permitted, but not required, by IEEE Std. 1003.1-200x.
- 39928   **FUTURE DIRECTIONS**
- 39929            None.
- 39930   **SEE ALSO**
- 39931            *ex*, *stty*
- 39932   **CHANGE HISTORY**
- 39933            First released in Issue 2.
- 39934   **Issue 4**
- 39935            Aligned with the ISO/IEC 9945-2:1993 standard.
- 39936   **Issue 5**
- 39937            FUTURE DIRECTIONS section added.
- 39938   **Issue 6**
- 39939            This utility is now marked as part of the User Portability Utilities option.
- 39940            The APPLICATION USAGE section is added.
- 39941            The obsolescent SYNOPSIS is removed.
- 39942            The following new requirements on POSIX implementations derive from alignment with the  
39943            Single UNIX Specification:
- 39944
  - The **lisp** mode is added.
- 39945
  - The **reindent** command description is added.
- 39946            The *vi* utility has been extensively rewritten for alignment with the IEEE P1003.2b draft  
39947            standard.

39948 **NAME**

39949           wait — await process completion

39950 **SYNOPSIS**39951           wait [*pid...*]39952 **DESCRIPTION**

39953           When an asynchronous list (see Section 2.9.3.1 (on page 2259)) is started by the shell, the process  
39954           ID of the last command in each element of the asynchronous list shall become known in the  
39955           current shell execution environment; see Section 2.13 (on page 2273).

39956           If the *wait* utility is invoked with no operands, it shall wait until all process IDs known to the  
39957           invoking shell have terminated and exit with a zero exit status.

39958           If one or more *pid* operands are specified that represent known process IDs, the *wait* utility shall  
39959           wait until all of them have terminated. If one or more *pid* operands are specified that represent  
39960           unknown process IDs, *wait* shall treat them as if they were known process IDs that exited with  
39961           exit status 127. The exit status returned by the *wait* utility shall be the exit status of the process  
39962           requested by the last *pid* operand.

39963           The known process IDs are applicable only for invocations of *wait* in the current shell execution  
39964           environment.

39965 **OPTIONS**

39966           None.

39967 **OPERANDS**

39968           The following operand shall be supported:

39969           *pid*           One of the following:

39970                           1. The unsigned decimal integer process ID of a command, for which the utility  
39971                           is to wait for the termination.

39972                           2. A job control job ID (see the Base Definitions volume of  
39973                           IEEE Std. 1003.1-200x, Section 3.205, Job Control Job ID) that identifies a  
39974                           background process group to be waited for. The job control job ID notation is  
39975                           applicable only for invocations of *wait* in the current shell execution  
39976                           environment; see Section 2.13 (on page 2273). The exit status of *wait* shall be  
39977                           determined by the last command in the pipeline.

39978                           **Note:**       The job control job ID type of *pid* is only available on systems  
39979                           supporting the User Portability Utilities option.

39980 **STDIN**

39981           Not used.

39982 **INPUT FILES**

39983           None.

39984 **ENVIRONMENT VARIABLES**39985           The following environment variables shall affect the execution of *wait*:

39986           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
39987           If *LANG* is unset or null, the corresponding value from the implementation-  
39988           defined default locale shall be used. If any of the internationalization variables  
39989           contains an invalid setting, the utility shall behave as if none of the variables had  
39990           been defined.

- 39991 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
39992 internationalization variables.
- 39993 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
39994 characters (for example, single-byte as opposed to multi-byte characters in  
39995 arguments).
- 39996 *LC\_MESSAGES*  
39997 Determine the locale that should be used to affect the format and contents of  
39998 diagnostic messages written to standard error.
- 39999 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 40000 **ASYNCHRONOUS EVENTS**
- 40001 Default.
- 40002 **STDOUT**
- 40003 Not used.
- 40004 **STDERR**
- 40005 Used only for diagnostic messages.
- 40006 **OUTPUT FILES**
- 40007 None.
- 40008 **EXTENDED DESCRIPTION**
- 40009 None.
- 40010 **EXIT STATUS**
- 40011 If one or more operands were specified, all of them have terminated or were not known by the  
40012 invoking shell, and the status of the last operand specified is known, then the exit status of *wait*  
40013 shall be the exit status information of the command indicated by the last operand specified. If  
40014 the process terminated abnormally due to the receipt of a signal, the exit status shall be greater  
40015 than 128 and shall be distinct from the exit status generated by other signals, but the exact value  
40016 is unspecified. (See the *kill -l* option.) Otherwise, the *wait* utility shall exit with one of the  
40017 following values:
- 40018 0 The *wait* utility was invoked with no operands and all process IDs known by the  
40019 invoking shell have terminated.
- 40020 1-126 The *wait* utility detected an error.
- 40021 127 The command identified by the last *pid* operand specified is unknown.
- 40022 **CONSEQUENCES OF ERRORS**
- 40023 Default.
- 40024 **APPLICATION USAGE**
- 40025 On most implementations, *wait* is a shell built-in. If it is called in a subshell or separate utility  
40026 execution environment, such as one of the following:
- 40027 (wait)  
40028 nohup wait ...  
40029 find . -exec wait ... \;
- 40030 it returns immediately because there are no known process IDs to wait for in those  
40031 environments.
- 40032 Historical implementations of interactive shells have discarded the exit status of terminated  
40033 background processes before each shell prompt. Therefore, the status of background processes  
40034 was usually lost unless it terminated while *wait* was waiting for it. This could be a serious

40035 problem when a job that was expected to run for a long time actually terminated quickly with a  
 40036 syntax or initialization error because the exit status returned was usually zero if the requested  
 40037 process ID was not found. This volume of IEEE Std. 1003.1-200x requires the implementation to  
 40038 keep the status of terminated jobs available until the status is requested, so that scripts like:

```
40039 j1&
40040 p1=$!
40041 j2&
40042 wait $p1
40043 echo Job 1 exited with status $?
40044 wait $!
40045 echo Job 2 exited with status $?
```

40046 works without losing status on any of the jobs. The shell is allowed to discard the status of any  
 40047 process that it determines the application cannot get the process ID from the shell. It is also  
 40048 required to remember only {CHILD\_MAX} number of processes in this way. Since the only way  
 40049 to get the process ID from the shell is by using the '!' shell parameter, the shell is allowed to  
 40050 discard the status of an asynchronous list if "\$!" was not referenced before another  
 40051 asynchronous list was started. (This means that the shell only has to keep the status of the last  
 40052 asynchronous list started if the application did not reference "\$!". If the implementation of the  
 40053 shell is smart enough to determine that a reference to "\$!" was not saved anywhere that the  
 40054 application can retrieve it later, it can use this information to trim the list of saved information.  
 40055 Note also that a successful call to *wait* with no operands discards the exit status of all  
 40056 asynchronous lists.)

40057 If the exit status of *wait* is greater than 128, there is no way for the application to know if the  
 40058 waited-for process exited with that value or was killed by a signal. Since most utilities exit with  
 40059 small values, there is seldom any ambiguity. Even in the ambiguous cases, most applications  
 40060 just need to know that the asynchronous job failed; it does not matter whether it detected an  
 40061 error and failed or was killed and did not complete its job normally.

#### 40062 EXAMPLES

40063 Although the exact value used when a process is terminated by a signal is unspecified, if it is  
 40064 known that a signal terminated a process, a script can still reliably figure out which signal using  
 40065 *kill* as shown by the following script:

```
40066 sleep 1000&
40067 pid=$!
40068 kill -kill $pid
40069 wait $pid
40070 echo $pid was terminated by a SIG$(kill -l $?) signal.
```

40071 If the following sequence of commands is run in less than 31 seconds:

```
40072 sleep 257 | sleep 31 &
40073 jobs -l %%
```

40074 either of the following commands returns the exit status of the second *sleep* in the pipeline:

```
40075 wait <pid of sleep 31>
40076 wait %%
```

#### 40077 RATIONALE

40078 The description of *wait* does not refer to the *waitpid()* function from the System Interfaces  
 40079 volume of IEEE Std. 1003.1-200x because that would needlessly overspecify this interface.  
 40080 However, the wording means that *wait* is required to wait for an explicit process when it is given  
 40081 an argument so that the status information of other processes is not consumed. Historical

40082 implementations use the *wait()* function defined in the System Interfaces volume of  
40083 IEEE Std. 1003.1-200x until *wait()* returns the requested process ID or finds that the requested  
40084 process does not exist. Because this means that a shell script could not reliably get the status of  
40085 all background children if a second background job was ever started before the first job finished,  
40086 it is recommended that the *wait* utility use a method such as the functionality provided by the  
40087 *waitpid()* function.

40088 The ability to wait for multiple *pid* operands was adopted from the KornShell.

40089 This new functionality was added because it is needed to determine the exit status of any  
40090 asynchronous list accurately. The only compatibility problem that this change creates is for a  
40091 script like

```
40092 while sleep 60 do
40093 job& echo Job started $(date) as $! done
```

40094 which causes the shell to monitor all of the jobs started until the script terminates or runs out of  
40095 memory. This would not be a problem if the loop did not reference "\$!" or if the script would  
40096 occasionally *wait* for jobs it started.

#### 40097 **FUTURE DIRECTIONS**

40098 None.

#### 40099 **SEE ALSO**

40100 *sh*, the System Interfaces volume of IEEE Std. 1003.1-200x, *waitpid()*

#### 40101 **CHANGE HISTORY**

40102 First released in Issue 2.

#### 40103 **Issue 4**

40104 Aligned with the ISO/IEC 9945-2:1993 standard.

40105 **NAME**40106 `wc` — word, line, and byte or character count40107 **SYNOPSIS**40108 `wc [-c|-m][-lw][file...]`40109 **DESCRIPTION**40110 The `wc` utility shall read one or more input files and, by default, write the number of <newline>  
40111 characters, words, and bytes contained in each input file to the standard output.40112 The utility also shall write a total count for all named files, if more than one input file is  
40113 specified.40114 The `wc` utility shall consider a *word* to be a non-zero-length string of characters delimited by  
40115 white space.40116 **OPTIONS**40117 The `wc` utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
40118 12.2, Utility Syntax Guidelines.

40119 The following options shall be supported:

40120 `-c` Write to the standard output the number of bytes in each input file.40121 `-l` Write to the standard output the number of <newline> characters in each input  
40122 file.40123 `-m` Write to the standard output the number of characters in each input file.40124 `-w` Write to the standard output the number of words in each input file.40125 When any option is specified, `wc` shall report only the information requested by the specified  
40126 options.40127 **OPERANDS**

40128 The following operand shall be supported:

40129 *file* A path name of an input file. If no *file* operands are specified, the standard input  
40130 shall be used.40131 **STDIN**40132 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
40133 section.40134 **INPUT FILES**

40135 The input files may be of any type.

40136 **ENVIRONMENT VARIABLES**40137 The following environment variables shall affect the execution of `wc`:40138 *LANG* Provide a default value for the internationalization variables that are unset or null.  
40139 If *LANG* is unset or null, the corresponding value from the implementation-  
40140 defined default locale shall be used. If any of the internationalization variables  
40141 contains an invalid setting, the utility shall behave as if none of the variables had  
40142 been defined.40143 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
40144 internationalization variables.40145 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
40146 characters (for example, single-byte as opposed to multi-byte characters in  
40147 arguments and input files) and which characters are defined as white space

- 40148 characters.
- 40149 **LC\_MESSAGES**
- 40150 Determine the locale that should be used to affect the format and contents of
- 40151 diagnostic messages written to standard error and informative messages written to
- 40152 standard output.
- 40153 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.
- 40154 **ASYNCHRONOUS EVENTS**
- 40155 Default.
- 40156 **STDOUT**
- 40157 By default, the standard output shall contain an entry for each input file of the form:
- 40158 "%d %d %d %s\n", <newlines>, <words>, <bytes>, <file>
- 40159 If the **-m** option is specified, the number of characters shall replace the <bytes> field in this
- 40160 format.
- 40161 If any options are specified and the **-l** option is not specified, the number of <newline>
- 40162 characters shall not be written.
- 40163 If any options are specified and the **-w** option is not specified, the number of words shall not be
- 40164 written.
- 40165 If any options are specified and neither **-c** nor **-m** is specified, the number of bytes or characters
- 40166 shall not be written.
- 40167 If no input *file* operands are specified, no name shall be written and no <blank> characters
- 40168 preceding the path name shall be written.
- 40169 If more than one input *file* operand is specified, an additional line shall be written, of the same
- 40170 format as the other lines, except that the word **total** (in the POSIX locale) shall be written instead
- 40171 of a path name and the total of each column shall be written as appropriate. Such an additional
- 40172 line, if any, is written at the end of the output.
- 40173 **STDERR**
- 40174 Used only for diagnostic messages.
- 40175 **OUTPUT FILES**
- 40176 None.
- 40177 **EXTENDED DESCRIPTION**
- 40178 None.
- 40179 **EXIT STATUS**
- 40180 The following exit values shall be returned:
- 40181 0 Successful completion.
- 40182 >0 An error occurred.
- 40183 **CONSEQUENCES OF ERRORS**
- 40184 Default.

**40185 APPLICATION USAGE**

40186 The **-m** option is not a switch, but an option at the same level as **-c**. Thus, to produce the full  
40187 default output with character counts instead of bytes, the command required is:

40188 `wc -mlw`

**40189 EXAMPLES**

40190 None.

**40191 RATIONALE**

40192 The output file format pseudo-*printf()* string differs from the the System V version of *wc*:

40193 `"%7d%7d%7d %s\n"`

40194 which produces possibly ambiguous and unparseable results for very large files, as it assumes no  
40195 number shall exceed six digits.

40196 Some historical implementations use only <space>, <tab>, and <newline> as word separators.  
40197 The equivalent of the ISO C standard *isspace()* function is more appropriate.

40198 The **-c** option stands for “character” count, even though it counts bytes. This stems from the  
40199 sometimes erroneous historical view that bytes and characters are the same size. Due to  
40200 international requirements, the **-m** option (reminiscent of “multi-byte”) was added to obtain  
40201 actual character counts.

40202 Early proposals only specified the results when input files were text files. The current  
40203 specification more closely matches historical practice. (Bytes, words, and <newline>s are  
40204 counted separately and the results are written when an end-of-file is detected.)

40205 Historical implementations of the *wc* utility only accepted one argument to specify the options  
40206 **-c**, **-l**, and **-w**. Some of them also had multiple occurrences of an option cause the  
40207 corresponding count to be written multiple times and had the order of specification of the  
40208 options affect the order of the fields on output, but did not document either of these. Because  
40209 common usage either specifies no options or only one option, and because none of this was  
40210 documented, the changes required by this volume of IEEE Std. 1003.1-200x should not break  
40211 many historical applications (and do not break any historical portable applications).

**40212 FUTURE DIRECTIONS**

40213 None.

**40214 SEE ALSO**

40215 *cksum*

**40216 CHANGE HISTORY**

40217 First released in Issue 2.

**40218 Issue 4**

40219 Aligned with the ISO/IEC 9945-2: 1993 standard.



40220 **NAME**40221            *what* — identify SCCS files (**DEVELOPMENT**)40222 **SYNOPSIS**40223 XSI        *what* [-s] *file...*

40224

40225 **DESCRIPTION**

40226            The *what* utility shall search the given files for all occurrences of the pattern that *get* (see *get* (on page 2685)) substitutes for %Z% ("@( # ) ") and shall write to standard output what follows until the first occurrence of one of the following:

40229            "    &gt;    newline    \    NUL

40230 **OPTIONS**

40231            The *what* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

40233            The following option is supported:

40234            -s                Quit after finding the first occurrence of the pattern in each file.

40235 **OPERANDS**

40236            The following operands shall be supported:

40237            *file*            A path name of a file to search.40238 **STDIN**

40239            Not used.

40240 **INPUT FILES**

40241            The input files are of any file type.

40242 **ENVIRONMENT VARIABLES**40243            The following environment variables shall affect the execution of *what*:

40244            *LANG*            Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-defined default locale shall be used. If any of the internationalization variables contains an invalid setting, the utility shall behave as if none of the variables had been defined.

40249            *LC\_ALL*        If set to a non-empty string value, override the values of all the other internationalization variables.

40251            *LC\_CTYPE*    Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

40254            *LC\_MESSAGES*

40255                Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

40257            *NLSPATH*    Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

40258 **ASYNCHRONOUS EVENTS**

40259            Default.

40260 **STDOUT**

40261 The standard output shall consist of the following for each *file* operand:

40262 "%s:\n\t%s\n", <pathname>, <identification string>

40263 **STDERR**

40264 Used only for diagnostic messages.

40265 **OUTPUT FILES**

40266 None.

40267 **EXTENDED DESCRIPTION**

40268 None.

40269 **EXIT STATUS**

40270 The following exit values shall be returned:

40271 0 Any matches were found.

40272 1 Otherwise.

40273 **CONSEQUENCES OF ERRORS**

40274 Default.

40275 **APPLICATION USAGE**

40276 The *what* utility is intended to be used in conjunction with the SCCS command *get*, which automatically inserts identifying information, but it can also be used where the information is inserted by any other means.

40279 When the string "@(#)" is included in a library routine in a shared library, it might not be found  
40280 in an **a.out** file using that library routine.

40281 **EXAMPLES**

40282 If the C-language program in file **f.c** contains:

40283 char ident[] = "@(#)identification information";

40284 and **f.c** is compiled to yield **f.o** and **a.out**, then the command:

40285 what f.c f.o a.out

40286 writes:

40287 f.c:  
40288 identification information

40289 ...

40290 f.o:  
40291 identification information

40292 ...

40293 a.out:  
40294 identification information

40295 ...

40296 **RATIONALE**

40297 None.

40298 **FUTURE DIRECTIONS**

40299 None.

40300 **SEE ALSO**

40301 *get*

40302 **CHANGE HISTORY**

40303 First released in Issue 2.

40304 **Issue 4**

40305 Format reorganized.

40306 Utility Syntax Guidelines support mandated.

40307 Internationalized environment variable support mandated.

## 40308 NAME

40309        **who** — display who is on the system

## 40310 SYNOPSIS

40311 UP        **who** [-mTu]

40312

40313 XSI       **who** [-mu]-s[-bHlprt][*file*]

40314        **who** [-mTu][-abdHlprt][*file*]

40315        **who** -q [*file*]

40316        **who** am i

40317        **who** am I

40318

## 40319 DESCRIPTION

40320        The *who* utility shall list various pieces of information about accessible users. The domain of  
40321 accessibility is implementation-defined.

40322 XSI       Based on the options given, *who* can also list the user's name, terminal line, login time, elapsed  
40323 time since activity occurred on the line, and the process ID of the command interpreter for each  
40324 current system user.

## 40325 OPTIONS

40326        The *who* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
40327 12.2, Utility Syntax Guidelines.

40328        The following options shall be supported. The metavariables, such as *<line>*, refer to fields  
40329 described in the STDOUT section.

40330 XSI        **-a**        Process the implementation-defined database or named file with the **-b**, **-d**, **-l**, **-p**,  
40331 **-r**, **-t**, **-T** and **-u** options turned on.

40332 XSI        **-b**        Write the time and date of the last reboot.

40333 XSI        **-d**        Write a list of all processes that have expired and not been respawned by the *init*  
40334 system process. The *<exit>* field appears for dead processes and contains the  
40335 termination and exit values of the dead process. This can be useful in determining  
40336 why a process terminated.

40337 XSI        **-H**        Write column headings above the regular output.

40338 XSI        **-l**        (The letter ell.) List only those lines on which the system is waiting for someone to  
40339 login. The *<name>* field is **LOGIN** in such cases. Other fields are the same as for  
40340 user entries except that the *<state>* field does not exist.

40341        **-m**        Output only information about the current terminal.

40342 XSI        **-p**        List any other process that is currently active and has been previously spawned by  
40343 *init*.

40344 XSI        **-q**        (Quick.) List only the names and the number of users currently logged on. When  
40345 this option is used, all other options are ignored.

40346 XSI        **-r**        Write the current *run-level* of the *init* process.

40347 XSI        **-s**        List only the *<name>*, *<line>*, and *<time>* fields. This is the default case.

40348 XSI        **-t**        Indicate the last change to the system clock.

40349           **-T**           Show the state of each terminal, as described in the STDOUT section.

40350 XSI       **-u**           This option lists only those users who are currently logged in. Output the user's "idle time" in addition to any other information. The idle time is the time since any activity occurred on the user's terminal. The method of determining this is unspecified. The *<name>* is the user's login name. The *<line>* is the name of the line as found in the directory */dev*. The *<time>* is the time that the user logged in. The *<activity>* is the number of hours and minutes since activity last occurred on that particular line. A dot indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked *<old>*. This field is useful when trying to determine whether a person is working at the terminal or not. The *<pid>* is the process ID of the user's login process.

#### 40361 OPERANDS

40362 XSI       The following operands shall be supported:

40363           **am i, am I**     In the POSIX locale, limit the output to describing the invoking user, equivalent to the **-m** option. The **am** and **i** or **I** must be separate arguments.

40365           *file*           Specify a path name of a file to substitute for the implementation-defined database of logged-on users that *who* uses by default.

#### 40367 STDIN

40368           Not used.

#### 40369 INPUT FILES

40370           None.

#### 40371 ENVIRONMENT VARIABLES

40372           The following environment variables shall affect the execution of *who*:

40373           **LANG**           Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-defined default locale shall be used. If any of the internationalization variables contains an invalid setting, the utility shall behave as if none of the variables had been defined.

40378           **LC\_ALL**          If set to a non-empty string value, override the values of all the other internationalization variables.

40380           **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

40383           **LC\_MESSAGES**     Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

40386           **LC\_TIME**         Determine the locale used for the format and contents of the date and time strings.

40387 XSI       **NLSPATH**         Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 40388 ASYNCHRONOUS EVENTS

40389           Default.

40390 **STDOUT**

40391 XSI OF XSI-conformant systems shall write the default information to the standard output in the  
 40392 following general format:

40393 `<name>[<state>]<line><time>[<activity>][<pid>][<comment>][<exit>]`

40394 The following format shall be used for the **-T** option:

40395 `"%s %c %s %s\n" <name>, <terminal state>, <terminal name>,  
 40396 <time of login>`

40397 where *<terminal state>* is one of the following characters:

- 40398 + The terminal allows write access to other users.
- 40399 - The terminal denies write access to other users.
- 40400 ? The terminal write-access state cannot be determined.

40401 In the POSIX locale, the *<time of login>* shall be equivalent in format to the output of:

40402 `date +"%b %e %H:%M"`

40403 If the **-u** option is used with **-T**, the idle time shall be added to the end of the previous format in  
 40404 an unspecified format.

40405 **STDERR**

40406 Used only for diagnostic messages.

40407 **OUTPUT FILES**

40408 None.

40409 **EXTENDED DESCRIPTION**

40410 None.

40411 **EXIT STATUS**

40412 The following exit values shall be returned:

- 40413 0 Successful completion.
- 40414 >0 An error occurred.

40415 **CONSEQUENCES OF ERRORS**

40416 Default.

40417 **APPLICATION USAGE**

40418 The name *init* used for the system process is the most commonly used on historical systems, but  
 40419 it may vary.

40420 The “domain of accessibility” referred to is a broad concept that permits interpretation either on  
 40421 a very secure basis or even to allow a network-wide implementation like the historical *rwho*.

40422 **EXAMPLES**

40423 None.

40424 **RATIONALE**

40425 Due to differences between historical implementations, the base options provided were a  
 40426 compromise to allow users to work with those functions. The standard developers also  
 40427 considered removing all the options, but felt that these options offered users valuable  
 40428 functionality. Additional options to match historical systems are available on XSI-conformant  
 40429 systems.

- 40430 It is recognized that the *who* command may be of limited usefulness, especially in a multi-level  
40431 secure environment. The standard developers considered, however, that having some standard  
40432 method of determining the “accessibility” of other users would aid user portability.
- 40433 No format was specified for the default *who* output for systems not supporting the XSI  
40434 Extension. In such a user-oriented command, designed only for human use, this was not  
40435 considered to be a deficiency.
- 40436 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, and *write* require  
40437 that they use the same format.
- 40438 **FUTURE DIRECTIONS**
- 40439 None.
- 40440 **SEE ALSO**
- 40441 *msg*
- 40442 **CHANGE HISTORY**
- 40443 First released in Issue 2.
- 40444 **Issue 4**
- 40445 Aligned with the ISO/IEC 9945-2:1993 standard.
- 40446 **Issue 6**
- 40447 This utility is now marked as part of the User Portability Utilities option.

## 40448 NAME

40449 write — write to another user

## 40450 SYNOPSIS

40451 UP write *user\_name* [*terminal*]

40452

## 40453 DESCRIPTION

40454 The *write* utility shall read lines from the user's standard input and write them to the terminal of  
40455 another user. When first invoked, it shall write the message:40456 **Message from** *sender-login-id* (*sending-terminal*) [*date*]...40457 to *user\_name*. When it has successfully completed the connection, the sender's terminal shall be  
40458 alerted twice to indicate that what the sender is typing is being written to the recipient's  
40459 terminal.

40460 If the recipient wants to reply, this can be accomplished by typing:

40461 write *sender-login-id* [*sending-terminal*]40462 upon receipt of the initial message. Whenever a line of input as delimited by a NL, EOF, or EOL  
40463 special character (see the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General  
40464 Terminal Interface) is accumulated while in canonical input mode, the accumulated data shall be  
40465 written on the other user's terminal. Characters shall be processed as follows:

- 40466
- Typing the <alert> character shall write the alert character to the recipient's terminal.
  - 40467 • Typing the erase and kill characters shall affect the sender's terminal in the manner described  
40468 by the **termios** interface in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11,  
40469 General Terminal Interface.
  - 40470 • Typing the interrupt or end-of-file characters shall cause *write* to write an appropriate  
40471 message ("EOT\n" in the POSIX locale) to the recipient's terminal and exit.
  - 40472 • Typing characters from *LC\_CTYPE* classifications **print** or **space** shall cause those characters  
40473 to be sent to the recipient's terminal.
  - 40474 • When and only when the *stty iexten* local mode is enabled, the existence and processing of  
40475 additional special control characters and multi-byte or single-byte functions is  
40476 implementation-defined.
  - 40477 • Typing other non-printable characters shall cause implementation-defined sequences of  
40478 printable characters to be written to the recipient's terminal.

40479 To write to a user who is logged in more than once, the *terminal* argument can be used to indicate  
40480 which terminal to write to; otherwise, the recipient's terminal is selected in an implementation-  
40481 defined manner and an informational message is written to the sender's standard output,  
40482 indicating which terminal was chosen.40483 Permission to be a recipient of a *write* message can be denied or granted by use of the *mesg*  
40484 utility. However, a user's privilege may further constrain the domain of accessibility of other  
40485 users' terminals. The *write* utility shall fail when the user lacks the appropriate privileges to  
40486 perform the requested action.

## 40487 OPTIONS

40488 None.



40489 **OPERANDS**

40490 The following operands shall be supported:

40491 *user\_name* Login name of the person to whom the message shall be written. The application  
40492 shall ensure that this operand is of the form returned by the *who* utility.

40493 *terminal* Terminal identification in the same format provided by the *who* utility.

40494 **STDIN**

40495 Lines to be copied to the recipient's terminal is read from standard input.

40496 **INPUT FILES**

40497 None.

40498 **ENVIRONMENT VARIABLES**

40499 The following environment variables shall affect the execution of *write*:

40500 *LANG* Provide a default value for the internationalization variables that are unset or null.  
40501 If *LANG* is unset or null, the corresponding value from the implementation-  
40502 defined default locale shall be used. If any of the internationalization variables  
40503 contains an invalid setting, the utility shall behave as if none of the variables had  
40504 been defined.

40505 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
40506 internationalization variables.

40507 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
40508 characters (for example, single-byte as opposed to multi-byte characters in  
40509 arguments and input files). If the recipient's locale does not use an *LC\_CTYPE*  
40510 equivalent to the sender's, the results are undefined.

40511 *LC\_MESSAGES*

40512 Determine the locale that should be used to affect the format and contents of  
40513 diagnostic messages written to standard error and informative messages written to  
40514 standard output.

40515 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

40516 **ASYNCHRONOUS EVENTS**

40517 If an interrupt signal is received, *write* shall write an appropriate message on the recipient's  
40518 terminal and exits with a status of zero. It shall take the standard action for all other signals.

40519 **STDOUT**

40520 An informational message shall be written to standard output if a recipient is logged in more  
40521 than once.

40522 **STDERR**

40523 Used only for diagnostic messages.

40524 **OUTPUT FILES**

40525 The recipient's terminal is used for output.

40526 **EXTENDED DESCRIPTION**

40527 None.

40528 **EXIT STATUS**

40529 The following exit values shall be returned:

40530 0 Successful completion.

40531 >0 The addressed user is not logged on or the addressed user denies permission.

40532 **CONSEQUENCES OF ERRORS**

40533 Default.

40534 **APPLICATION USAGE**

40535 The *talk* utility is considered by some users to be a more usable utility on full-screen terminals.

40536 **EXAMPLES**

40537 None.

40538 **RATIONALE**

40539 The *write* utility was included in this volume of IEEE Std. 1003.1-200x since it can be  
40540 implemented on all terminal types. The standard developers considered the *talk* utility, which  
40541 cannot be implemented on certain terminals, to be a “better” communications interface. Both of  
40542 these programs are in widespread use on historical implementations. Therefore, the standard  
40543 developers decided that both utilities should be specified.

40544 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*  
40545 require that they all use or accept the same format.

40546 **FUTURE DIRECTIONS**

40547 None.

40548 **SEE ALSO**

40549 *mesg*, *talk*, *who*, the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11, General  
40550 Terminal Interface

40551 **CHANGE HISTORY**

40552 First released in Issue 2.

40553 **Issue 4**

40554 Aligned with the ISO/IEC 9945-2:1993 standard.

40555 **Issue 5**

40556 FUTURE DIRECTIONS section added.

40557 **Issue 6**

40558 This utility is now marked as part of the User Portability Utilities option.

40559 The normative text is reworded to avoid use of the term “must” for application requirements.

## 40560 NAME

40561 xargs — construct argument lists and invoke utility

## 40562 SYNOPSIS

```
40563 xsi xargs [-t][-p][-E eofstr][-I replstr][-L number][-n number [-x]]
40564 [-s size][utility [argument...]]
```

## 40565 DESCRIPTION

40566 The *xargs* utility shall construct a command line consisting of the *utility* and *argument* operands  
 40567 specified followed by as many arguments read in sequence from standard input as fit in length  
 40568 and number constraints specified by the options. The *xargs* utility shall then invoke the  
 40569 constructed command line and wait for its completion. This sequence shall be repeated until one  
 40570 of the following occurs:

- 40571 • An end-of-file condition is detected on standard input.
- 40572 • The logical end-of-file string (see the **-E eofstr** option) is found on standard input after  
 40573 double-quote processing, apostrophe processing, and backslash escape processing (see next  
 40574 paragraph).
- 40575 • An invocation of a constructed command line returns an exit status of 255.

40576 The application shall ensure that arguments in the standard input are separated by unquoted  
 40577 <blank> characters, or unescaped <blank> characters or <newline> characters. A string of zero  
 40578 or more non-double-quote ( ' " ' ) and non-<newline> characters can be quoted by enclosing  
 40579 them in double-quotes. A string of zero or more non-apostrophe ( ' \ ' ' ) and non-<newline>  
 40580 characters can be quoted by enclosing them in apostrophes. Any unquoted character can be  
 40581 escaped by preceding it with a backslash. The utility shall be executed one or more times until  
 40582 the end-of-file is reached or the logical end-of file string is found. The results are unspecified if  
 40583 the utility named by *utility* attempts to read from its standard input.

40584 The generated command line length shall be the sum of the size in bytes of the utility name and  
 40585 each argument treated as strings, including a null byte terminator for each of these strings. The  
 40586 *xargs* utility shall limit the command line length such that when the command line is invoked,  
 40587 the combined argument and environment lists (see the *exec* family of functions in the System  
 40588 Interfaces volume of IEEE Std. 1003.1-200x) shall not exceed {ARG\_MAX}-2 048 bytes. Within  
 40589 this constraint, if neither the **-n** nor the **-s** option is specified, the default command line length  
 40590 shall be at least {LINE\_MAX}.

## 40591 OPTIONS

40592 The *xargs* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 40593 12.2, Utility Syntax Guidelines.

40594 The following options shall be supported:

40595 **-E eofstr** Use *eofstr* as the logical end-of-file string. If **-E** is not specified, it is unspecified  
 40596 whether the logical end-of-file string is the underscore character ( ' \_ ' ) or the end-  
 40597 of-file string capability is disabled. When *eofstr* is the null string, the logical end-  
 40598 of-file string capability shall be disabled and underscore characters shall be taken  
 40599 literally.

40600 xsi **-I replstr** Insert mode: *utility* is executed for each line from standard input, taking the entire  
 40601 line as a single argument, inserting it in *arguments* for each occurrence of *replstr*. A  
 40602 maximum of five arguments in *arguments* can each contain one or more instances  
 40603 of *replstr*. Any <blank> characters at the beginning of each line shall be ignored.  
 40604 Constructed arguments cannot grow larger than 255 bytes. Option **-x** is forced on.

- 40605 XSI **-L number** The *utility* shall be executed for each non-empty *number* lines of arguments from standard input. The last invocation of *utility* shall be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first <newline> character unless the last character of the line is a <blank> character; a trailing <blank> character signals continuation to the next non-empty line, inclusive. The **-L** and **-n** options are mutually-exclusive; the last one specified shall take effect.
- 40606
- 40607
- 40608
- 40609
- 40610
- 40611 **-n number** Invoke *utility* using as many standard input arguments as possible, up to *number* (a positive decimal integer) arguments maximum. Fewer arguments shall be used if:
- 40612
- The command line length accumulated exceeds the size specified by the **-s** option (or {LINE\_MAX} if there is no **-s** option).
  - The last iteration has fewer than but not zero, operands remaining.
- 40613
- 40614
- 40615
- 40616 **-p** Prompt mode: the user is asked whether to execute *utility* at each invocation. Trace mode (**-t**) is turned on to write the command instance to be executed, followed by a prompt to standard error. An affirmative response read from */dev/tty* shall execute the command; otherwise, that particular invocation of *utility* shall be skipped.
- 40617
- 40618
- 40619
- 40620
- 40621 **-s size** Invoke *utility* using as many standard input arguments as possible yielding a command line length less than *size* (a positive decimal integer) bytes. Fewer arguments shall be used if:
- 40622
- 40623
- The total number of arguments exceeds that specified by the **-n** option.
  - The total number of lines exceeds that specified by the **-L** option.
  - End-of-file is encountered on standard input before *size* bytes are accumulated.
- 40624
- 40625 XSI
- 40626
- 40627 Values of *size* up to at least {LINE\_MAX} bytes shall be supported, provided that the constraints specified in the DESCRIPTION are met. It shall not be considered an error if a value larger than that supported by the implementation or exceeding the constraints specified in the DESCRIPTION is given; *xargs* shall use the largest value it supports within the constraints.
- 40628
- 40629
- 40630
- 40631
- 40632 **-t** Enable trace mode. Each generated command line shall be written to standard error just prior to invocation.
- 40633
- 40634 **-x** Terminate if a command line containing *number* arguments (see the **-n** option above) or *number* lines (see the **-L** option above) will not fit in the implied or specified size (see the **-s** option above).
- 40635 XSI
- 40636

#### 40637 OPERANDS

40638 The following operands shall be supported:

- 40639 *utility* The name of the utility to be invoked, found by search path using the *PATH* environment variable, described in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables. If *utility* is omitted, the default shall be the *echo* utility. If the *utility* operand names any of the special built-in utilities in Section 2.15 (on page 2276), the results are undefined.
- 40640
- 40641
- 40642
- 40643
- 40644 *argument* An initial option or operand for the invocation of *utility*.

#### 40645 STDIN

40646 The standard input shall be a text file. The results are unspecified if an end-of-file condition is detected immediately following an escaped <newline> character.

40647

40648 **INPUT FILES**

40649       The file `/dev/tty` is used to read responses required by the `-p` option.

40650 **ENVIRONMENT VARIABLES**

40651       The following environment variables shall affect the execution of *xargs*:

40652       **LANG**       Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-defined default locale shall be used. If any of the internationalization variables contains an invalid setting, the utility shall behave as if none of the variables had been defined.

40657       **LC\_ALL**      If set to a non-empty string value, override the values of all the other internationalization variables.

40659       **LC\_COLLATE**

40660       Determine the locale for the behavior of ranges, equivalence classes and multi-character collating elements used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

40663       **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

40668       **LC\_MESSAGES**

40669       Determine the locale for the processing of affirmative responses and that should be used to affect the format and contents of diagnostic messages written to standard error.

40672 XSI     **NLS\_PATH**   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

40673       **PATH**        Determine the location of *utility*, as described in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables.

40675 **ASYNCHRONOUS EVENTS**

40676       Default.

40677 **STDOUT**

40678       Not used.

40679 **STDERR**

40680       Used for diagnostic messages and the `-t` and `-p` options. If the `-t` option is specified, the *utility* and its constructed argument list shall be written to standard error, as it will be invoked, prior to invocation. If `-p` is specified, a prompt of the following format shall be written (in the POSIX locale):

40684       " ? . . . "

40685       at the end of the line of the output from `-t`.

40686 **OUTPUT FILES**

40687       None.

40688 **EXTENDED DESCRIPTION**

40689       None.

40690 **EXIT STATUS**

40691 The following exit values shall be returned:

- 40692           0    All invocations of *utility* returned exit status zero.
- 40693           1-125   A command line meeting the specified requirements could not be assembled, one or more of the invocations of *utility* returned a non-zero exit status, or some other error occurred.
- 40694
- 40695
- 40696           126    The utility specified by *utility* was found but could not be invoked.
- 40697           127    The utility specified by *utility* could not be found.

40698 **CONSEQUENCES OF ERRORS**

40699           If a command line meeting the specified requirements cannot be assembled, the utility cannot be invoked, an invocation of the utility is terminated by a signal, or an invocation of the utility exits with exit status 255, the *xargs* utility shall write a diagnostic message and exit without processing any remaining input.

40700

40701

40702

40703 **APPLICATION USAGE**

40704           The 255 exit status allows a utility being used by *xargs* to tell *xargs* to terminate if it knows no further invocations using the current data stream succeeds. Thus, *utility* should explicitly *exit* with an appropriate value to avoid accidentally returning with 255.

40705

40706

40707           Note that input is parsed as lines; <blank> characters separate arguments. If *xargs* is used to bundle output of commands like *find dir -print* or *ls* into commands to be executed, unexpected results are likely if any file names contain any <blank> characters or <newline> characters. This can be fixed by using *find* to call a script that converts each file found into a quoted string that is then piped to *xargs*. Note that the quoting rules used by *xargs* are not the same as in the shell. They were not made consistent here because existing applications depend on the current rules and the shell syntax is not fully compatible with it. An easy rule that can be used to transform any string into a quoted form that *xargs* interprets correctly is to precede each character in the string with a backslash.

40708

40709

40710

40711

40712

40713

40714

40715

40716           On implementations with a large value for {ARG\_MAX}, *xargs* may produce command lines longer than {LINE\_MAX}. For invocation of utilities, this is not a problem. If *xargs* is being used to create a text file, users should explicitly set the maximum command line length with the *-s* option.

40717

40718

40719

40720           The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish “failure to find a utility” from “invoked utility exited with an error indication”. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for “normal error conditions” and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason.

40721

40722

40723

40724

40725

40726

40727

40728

40729

40730 **EXAMPLES**

- 40731           1. The following command combines the output of the parenthesised commands onto one line, which is then written to the end-of-file **log**:
- 40732
- 40733           

```
(logname; date; printf "%s\n" "$0 $*") | xargs >>log
```
- 40734           2. The following command invokes *diff* with successive pairs of arguments originally typed as command line arguments (assuming there are no embedded <blank> characters in the
- 40735

40736 elements of the original argument list):

```
40737 printf "%s\n" "$*" | xargs -n 2 -x diff
```

40738 3. The user is asked which files in the current directory shall be archived. The files are  
40739 archived into **arch**; *a*, one at a time, or *b*, many at a time.

```
40740 a. ls | xargs -p -L 1 ar -r arch
```

```
40741 b. ls | xargs -p -L 1 | xargs ar -r arch
```

40742 4. The following executes with successive pairs of arguments originally typed as command  
40743 line arguments:

```
40744 echo $* | xargs -n 2 diff
```

40745 5. On XSI-conformant systems, the following moves all files from directory **\$1** to directory **\$2**,  
40746 and echo each move command just before doing it:

```
40747 ls $1 | xargs -I {} -t mv $1/{ } $2/{ }
```

#### 40748 RATIONALE

40749 The *xargs* utility was usually found only in System V-based systems; BSD systems included an  
40750 *apply* utility that provided functionality similar to *xargs -n number*. The SVID lists *xargs* as a  
40751 software development extension. This volume of IEEE Std. 1003.1-200x does not share the view  
40752 that it is used only for development, and therefore it is not optional.

40753 The classic application of the *xargs* utility is in conjunction with the *find* utility to reduce the  
40754 number of processes launched by a simplistic use of the *find-exec* combination. The *xargs* utility  
40755 is also used to enforce an upper limit on memory required to launch a process. With this basis in  
40756 mind, this volume of IEEE Std. 1003.1-200x selected only the minimal features required.

40757 Although the 255 exit status is mostly an accident of historical implementations, it allows a  
40758 utility being used by *xargs* to tell *xargs* to terminate if it knows no further invocations using the  
40759 current data stream shall succeed. Any non-zero exit status from a utility falls into the 1-125  
40760 range when *xargs* exits. There is no statement of how the various non-zero utility exit status  
40761 codes are accumulated by *xargs*. The value could be the addition of all codes, their highest  
40762 value, the last one received, or a single value such as 1. Since no algorithm is arguably better  
40763 than the others, and since many of the standard utilities say little more (portably) than  
40764 “pass/fail”, no new algorithm was invented.

40765 Several other *xargs* options were withdrawn because simple alternatives already exist within this  
40766 volume of IEEE Std. 1003.1-200x. For example, the *-e eofstr* option can be replaced by features of  
40767 *sed*. The *-i replstr* option can be just as efficiently performed using a shell *for* loop. Since *xargs*  
40768 calls an *exec* function with each input line, the *-i* option does not usually exploit the grouping  
40769 capabilities of *xargs*.

40770 The requirement that *xargs* never produce command lines such that invocation of *utility* is  
40771 within 2 048 bytes of hitting the POSIX *exec* {ARG\_MAX} limitations is intended to guarantee  
40772 that the invoked utility has room to modify its environment variables and command line  
40773 arguments and still be able to invoke another utility. Note that the minimum {ARG\_MAX}  
40774 allowed by the System Interfaces volume of IEEE Std. 1003.1-200x is 4 096 bytes and the  
40775 minimum value allowed by the this volume of IEEE Std. 1003.1-200x is 2 048 bytes; therefore, the  
40776 2 048 bytes difference seems reasonable. Note, however, that *xargs* may never be able to invoke a  
40777 utility if the environment passed in to *xargs* comes close to using {ARG\_MAX} bytes.

40778 The version of *xargs* required by this volume of IEEE Std. 1003.1-200x is required to wait for the  
40779 completion of the invoked command before invoking another command. This was done because  
40780 historical scripts using *xargs* assumed sequential execution. Implementations wanting to provide

40781 parallel operation of the invoked utilities are encouraged to add an option enabling parallel  
 40782 invocation, but should still wait for termination of all of the children before *xargs* terminates  
 40783 normally.

40784 The `-e` option was omitted from the ISO POSIX-2:1993 standard in the belief that the *eofstr*  
 40785 option-argument was recognized only when it was on a line by itself and before quote and  
 40786 escape processing were performed, and that the logical end-of-file processing was only enabled  
 40787 if a `-e` option was specified. In that case, a simple *sed* script could be used to duplicate the `-e`  
 40788 functionality. Further investigation revealed that:

- 40789 • The logical end-of-file string was checked for after quote and escape processing, making a *sed*  
 40790 script that provided equivalent functionality much more difficult to write.
- 40791 • The default was to perform logical end-of-file processing with an underscore as the logical  
 40792 end-of-file string.

40793 To correct this misunderstanding, the `-E eofstr` option was adopted from the X/Open Portability  
 40794 Guide. Users should note that the description of the `-E` option matches historical documentation  
 40795 of the `-e` option (which was not adopted because it did not support the Utility Syntax  
 40796 Guidelines), by saying that if *eofstr* is the null string, logical end-of-file processing is disabled.  
 40797 Historical implementations of *xargs* actually did not disable logical end-of-file processing; they  
 40798 treated a null argument found in the input as a logical end-of-file string. (A null *string* argument  
 40799 could be generated using single or double quotes ( ' ' or " " ). Since this behavior was not  
 40800 documented historically, it is considered to be a bug.

#### 40801 FUTURE DIRECTIONS

40802 None.

#### 40803 SEE ALSO

40804 *echo*

#### 40805 CHANGE HISTORY

40806 First released in Issue 2.

#### 40807 Issue 4

40808 Aligned with the ISO/IEC 9945-2:1993 standard.

#### 40809 Issue 5

40810 Second FUTURE DIRECTION added.

#### 40811 Issue 6

40812 The obsolescent `-e`, `-i`, and `-l` options are removed.

40813 The following new requirements on POSIX implementations derive from alignment with the  
 40814 Single UNIX Specification:

- 40815 • The `-p` option is added.
- 40816 • In the INPUT FILES section, the file `/dev/tty` is used to read responses required by the `-p`  
 40817 option.
- 40818 • The STDERR section is updated to describe the `-p` option.

40819 The description of the `-E` option is aligned with the ISO POSIX-2:1993 standard.

40820 The normative text is reworded to avoid use of the term “must” for application requirements.



## 40821 NAME

40822 yacc — yet another compiler compiler (DEVELOPMENT)

## 40823 SYNOPSIS

40824 yacc [-dltv][-b *file\_prefix*][-p *sym\_prefix*] *grammar*

## 40825 DESCRIPTION

40826 The *yacc* utility shall read a description of a context-free grammar in *file* and write C source code,  
 40827 conforming to the ISO C standard, to a code file, and optionally header information into a  
 40828 header file, in the current directory. The C code shall define a function and related routines and  
 40829 macros for an automaton that executes a parsing algorithm meeting the requirements in  
 40830 **Algorithms** (on page 3288).

40831 The form and meaning of the grammar are described in the EXTENDED DESCRIPTION section.

40832 The C source code and header file shall be produced in a form suitable as input for the C  
 40833 compiler (see *c99* (on page 2425)).

## 40834 OPTIONS

40835 The *yacc* utility shall conform to the Base Definitions volume of IEEE Std. 1003.1-200x, Section  
 40836 12.2, Utility Syntax Guidelines.

40837 The following options shall be supported:

40838 **-b *file\_prefix*** Use *file\_prefix* instead of *y* as the prefix for all output file names. The code file  
 40839 *y.tab.c*, the header file *y.tab.h* (created when **-d** is specified), and the description  
 40840 file *y.output* (created when **-v** is specified), shall be changed to *file\_prefix.tab.c*,  
 40841 *file\_prefix.tab.h*, and *file\_prefix.output*, respectively.

40842 **-d** Write the header file; by default only the code file is written. The **#define**  
 40843 statements that associate the token codes assigned by *yacc* with the user-declared  
 40844 token names. This allows source files other than *y.tab.c* to access the token codes.

40845 **-l** Produce a code file that does not contain any **#line** constructs. If this option is not  
 40846 present, it is unspecified whether the code file or header file contains **#line**  
 40847 directives. This should only be used after the grammar and the associated actions  
 40848 are fully debugged.

40849 **-p *sym\_prefix*** Use *sym\_prefix* instead of *yy* as the prefix for all external names produced by *yacc*.  
 40850 The names affected shall include the functions *yyparse*, *yylex*, and *yyerror*, and the  
 40851 variables *yylval*, *yychar*, and *yydebug*. (In the remainder of this section, the six  
 40852 symbols cited are referenced using their default names only as a notational  
 40853 convenience.) Local names may also be affected by the **-p** option; however, the **-p**  
 40854 option shall not affect **#define** symbols generated by *yacc*.

40855 **-t** Modify conditional compilation directives to permit compilation of debugging  
 40856 code in the code file. Runtime debugging statements shall always be contained in  
 40857 the code file, but by default conditional compilation directives prevent their  
 40858 compilation.

40859 **-v** Write a file containing a description of the parser and a report of conflicts  
 40860 generated by ambiguities in the grammar.

## 40861 OPERANDS

40862 The following operand is required:

40863 *grammar* A path name of a file containing instructions, hereafter called *grammar*, for which a  
 40864 parser is to be created. The format for the grammar is described in the EXTENDED  
 40865 DESCRIPTION section.

40866 **STDIN**

40867 Not used.

40868 **INPUT FILES**40869 The file *grammar* shall be a text file formatted as specified in the EXTENDED DESCRIPTION  
40870 section.40871 **ENVIRONMENT VARIABLES**40872 The following environment variables shall affect the execution of *yacc*:40873 **LANG** Provide a default value for the internationalization variables that are unset or null.  
40874 If *LANG* is unset or null, the corresponding value from the implementation-  
40875 defined default locale shall be used. If any of the internationalization variables  
40876 contains an invalid setting, the utility shall behave as if none of the variables had  
40877 been defined.40878 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
40879 internationalization variables.40880 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
40881 characters (for example, single-byte as opposed to multi-byte characters in  
40882 arguments and input files).40883 **LC\_MESSAGES**40884 Determine the locale that should be used to affect the format and contents of  
40885 diagnostic messages written to standard error.40886 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.40887 The *LANG* and *LC\_\** variables affect the execution of the *yacc* utility as stated. The *main* function  
40888 defined in **Yacc Library** (on page 3288) shall call:40889 `setlocale(LC_ALL, " ")`40890 and thus, the program generated by *yacc* also shall be affected by the contents of these variables  
40891 at runtime.40892 **ASYNCHRONOUS EVENTS**

40893 Default.

40894 **STDOUT**

40895 Not used.

40896 **STDERR**40897 If shift/reduce or reduce/reduce conflicts are detected in *grammar*, *yacc* writes a report of those  
40898 conflicts to the standard error in an unspecified format.

40899 Standard error is also used for diagnostic messages.

40900 **OUTPUT FILES**40901 The code file, the header file, and the description file shall be text files. All are described in the  
40902 following sections.

**40903 Code File**

40904 This file shall contain the C source code for the *yyparse* routine. It shall contain code for the  
40905 various semantic actions with macro substitution performed on them as described in the  
40906 EXTENDED DESCRIPTION section. It also shall contain a copy of the **#define** statements in the  
40907 header file. If a **%union** declaration is used, the declaration for YYSTYPE shall be also included  
40908 in this file.

**40909 Header File**

40910 The header file shall contain **#define** statements that associate the token numbers with the token  
40911 names. This allows source files other than the code file to access the token codes. If a **%union**  
40912 declaration is used, the declaration for YYSTYPE and an *extern YYSTYPE yylval* declaration shall  
40913 be also included in this file.

**40914 Description File**

40915 The description file shall be a text file containing a description of the state machine  
40916 corresponding to the parser, using an unspecified format. Limits for internal tables (see **Limits**  
40917 (on page 3288)) shall also be reported, in an implementation-defined manner. (Some  
40918 implementations may use dynamic allocation techniques and have no specific limit values to  
40919 report.)

**40920 EXTENDED DESCRIPTION**

40921 The *yacc* command accepts a language that is used to define a grammar for a target language to  
40922 be parsed by the tables and code generated by *yacc*. The language accepted by *yacc* as a  
40923 grammar for the target language is described below using the *yacc* input language itself.

40924 The input *grammar* includes rules describing the input structure of the target language and code  
40925 to be invoked when these rules are recognized to provide the associated semantic action. The  
40926 code to be executed shall appear as bodies of text that are intended to be C-language code. The  
40927 C-language inclusions are presumed to form a correct function when processed by *yacc* into its  
40928 output files. The code included in this way shall be executed during the recognition of the target  
40929 language.

40930 Given a grammar, the *yacc* utility generates the files described in the OUTPUT FILES section.  
40931 The code file can be compiled and linked using *cc* or *c99*. If the declaration and programs  
40932 sections of the grammar file did not include definitions of *main*, *yylex*, and *yyerror*, the compiled  
40933 output requires linking with externally supplied version of those functions. Default versions of  
40934 *main* and *yyerror* are supplied in the *yacc* library and can be linked in by using the *-l y* operand to  
40935 *c99*. The *yacc* library interfaces need not support interfaces with other than the default *yy*  
40936 symbol prefix. The application provides the lexical analyzer function, *yylex*; the *lex* utility is  
40937 specifically designed to generate such a routine.

**40938 Input Language**

40939 The application shall ensure that every specification file consists of three sections in order:  
40940 *declarations*, *grammar rules*, and *programs*, separated by double percent signs ("%"). The  
40941 declarations and programs sections can be empty. If the latter is empty, the preceding "%"  
40942 mark separating it from the rules section can be omitted.

40943 The input is free form text following the structure of the grammar defined below.

40944 **Lexical Structure of the Grammar**

40945 The characters <blank>, <newline>, and <form-feed> shall be ignored, except that the  
 40946 application shall ensure that they do not appear in names or multi-character reserved symbols.  
 40947 Comments shall be enclosed in `"/* . . . */"`, and can appear wherever a name is valid.

40948 Names are of arbitrary length, made up of letters, periods ('. '), underscores ('\_ '), and non-  
 40949 initial digits. Uppercase and lowercase letters are distinct. Portable applications shall not use  
 40950 names beginning in `yy` or `YY` since the `yacc` parser uses such names. Many of the names appear  
 40951 in the final output of `yacc`, and thus they should be chosen to conform with any additional rules  
 40952 created by the C compiler to be used. In particular they appear in `#define` statements.

40953 A literal shall consist of a single character enclosed in single-quotes ('\' '). All of the escape  
 40954 sequences supported for character constants by the ISO C standard shall be supported by `yacc`.

40955 The relationship with the lexical analyzer is discussed in detail below.

40956 The application shall ensure that the NUL character is not used in grammar rules or literals.

40957 **Declarations Section**

40958 The declarations section is used to define the symbols used to define the target language and  
 40959 their relationship with each other. In particular, much of the additional information required to  
 40960 resolve ambiguities in the context-free grammar for the target language is provided here.

40961 Usually `yacc` assigns the relationship between the symbolic names it generates and their  
 40962 underlying numeric value. The declarations section makes it possible to control the assignment  
 40963 of these values.

40964 It is also possible to keep semantic information associated with the tokens currently on the parse  
 40965 stack in a user-defined C-language **union**, if the members of the union are associated with the  
 40966 various names in the grammar. The declarations section provides for this as well.

40967 The first group of declarators below all take a list of names as arguments. That list can optionally  
 40968 be preceded by the name of a C union member (called a *tag* below) appearing within '`<`' and  
 40969 '`>`'. (As an exception to the typographical conventions of the rest of this volume of  
 40970 IEEE Std. 1003.1-200x, in this case `<tag>` does not represent a metavariable, but the literal angle  
 40971 bracket characters surrounding a symbol.) The use of *tag* specifies that the tokens named on this  
 40972 line shall be of the same C type as the union member referenced by *tag*. This is discussed in  
 40973 more detail below.

40974 For lists used to define tokens, the first appearance of a given token can be followed by a  
 40975 positive integer (as a string of decimal digits). If this is done, the underlying value assigned to it  
 40976 for lexical purposes is taken to be that number.

40977 `%token [<tag>] name [number][name [number]]. . .`

40978 Declares *names* to be a token. If *tag* is present, the C type for all tokens on this line shall be  
 40979 declared to be the type referenced by *tag*. If a positive integer, *number*, follows a *name*, that  
 40980 value shall be assigned to the token.

40981 `%left [<tag>] name [number][name [number]]. . .`

40982 `%right [<tag>] name [number][name [number]]. . .`

40983 Declares *name* to be a token, and assigns precedence to it. One or more lines, each beginning  
 40984 with one of these symbols, can appear in this section. All tokens on the same line have the  
 40985 same precedence level and associativity; the lines are in order of increasing precedence or  
 40986 binding strength. `%left` denotes that the operators on that line are left associative, and  
 40987 `%right` similarly denotes right associative operators. If *tag* is present, it shall declare a C  
 40988 type for *names* as described for `%token`.

40989           %nonassoc [<tag>] *name* [*number*][*name* [*number*]]. . .  
40990           Declares *name* to be a token, and indicates that this cannot be used associatively. If the  
40991           parser encounters associative use of this token it reports an error. If *tag* is present, it shall  
40992           declare a C type for *names* as described for **%token**.

40993           %type [<tag>] *name*. . .  
40994           Declares that union member *names* are non-terminals, and thus it is required to have a *tag*  
40995           field at its beginning. Because it deals with non-terminals only, assigning a token number or  
40996           using a literal is also prohibited. If this construct is present, *yacc* shall perform type  
40997           checking; if this construct is not present, the parse stack shall hold only the **int** type.

40998           Every name used in *grammar* undefined by a **%token**, **%left**, **%right**, or **%nonassoc** declaration is  
40999           assumed to represent a non-terminal symbol. The *yacc* utility shall report an error for any non-  
41000           terminal symbol that does not appear on the left side of at least one grammar rule.

41001           Once the type, precedence, or token number of a name is specified, it shall not be changed. If the  
41002           first declaration of a token does not assign a token number, *yacc* shall assign a token number.  
41003           Once this assignment is made, the token number shall not be changed by explicit assignment.

41004           The following declarators do not follow the previous pattern.

41005           **%start** *name*  
41006           Declares the non-terminal *name* to be the *start symbol*, which represents the largest, most  
41007           general structure described by the grammar rules. By default, it is the left-hand side of the  
41008           first grammar rule; this default can be overridden with this declaration.

41009           **%union** { *body of union (in C)* }  
41010           Declares the *yacc* value stack to be a union of the various types of values desired. By default,  
41011           the values returned by actions (see below) and the lexical analyzer shall be integers. The  
41012           *yacc* utility keeps track of types, and it shall insert corresponding union member names in  
41013           order to perform strict type checking of the resulting parser.

41014           Alternatively, given that at least one <tag> construct is used, the union can be declared in a  
41015           header file (which shall be included in the declarations section by using an **#include**  
41016           construct within **%{** and **%}**), and a **typedef** used to define the symbol **YYSTYPE** to  
41017           represent this union. The effect of **%union** is to provide the declaration of **YYSTYPE** directly  
41018           from the *yacc* input.

41019           **%{ ... %}**  
41020           C-language declarations and definitions can appear in the declarations section, enclosed by  
41021           these marks. These statements shall be copied into the code file, and have global scope  
41022           within it so that they can be used in the rules and program sections.

41023           The application shall ensure that the declarations section is terminated by the token **%%**.

41024           **Grammar Rules in yacc**

41025           The rules section defines the context-free grammar to be accepted by the function *yacc* generates,  
41026           and associates with those rules C-language actions and additional precedence information. The  
41027           grammar is described below, and a formal definition follows.

41028           The rules section is comprised of one or more grammar rules. A grammar rule has the form:

41029           A : BODY ;

41030           The symbol **A** represents a non-terminal name, and **BODY** represents a sequence of zero or  
41031           more *names*, *literals*, and *semantic actions* that can then be followed by optional *precedence rules*.  
41032           Only the names and literals participate in the formation of the grammar; the semantic actions  
41033           and precedence rules are used in other ways. The colon and the semicolon are *yacc* punctuation.

41034 If there are several successive grammar rules with the same left-hand side, the vertical bar ' | '   
 41035 can be used to avoid rewriting the left-hand side; in this case the semicolon appears only after   
 41036 the last rule. The BODY part can be empty (or empty of names and literals) to indicate that the   
 41037 non-terminal symbol matches the empty string.

41038 The yacc utility assigns a unique number to each rule. Rules using the vertical bar notation are   
 41039 distinct rules. The number assigned to the rule appears in the description file.

41040 The elements comprising a BODY are:

41041 *name, literal*

41042 These form the rules of the grammar: *name* is either a *token* or a *non-terminal*; *literal*   
 41043 stands for itself (less the lexically required quotation marks).

41044 *semantic action*

41045 With each grammar rule, the user can associate actions to be performed each time   
 41046 the rule is recognized in the input process. (Note that the word "action" can also   
 41047 refer to the actions of the parser—shift, reduce, and so on.)

41048 These actions can return values and can obtain the values returned by previous   
 41049 actions. These values are kept in objects of type YYSTYPE (see %union). The   
 41050 result value of the action shall be kept on the parse stack with the left-hand side of   
 41051 the rule, to be accessed by other reductions as part of their right-hand side. By   
 41052 using the <tag> information provided in the declarations section, the code   
 41053 generated by yacc can be strictly type checked and contain arbitrary information. In   
 41054 addition, the lexical analyzer can provide the same kinds of values for tokens, if   
 41055 desired.

41056 An action is an arbitrary C statement and as such can do input or output, call   
 41057 subprograms and alter external variables. An action is one or more C statements   
 41058 enclosed in curly braces ' { ' and ' } '.

41059 Certain pseudo-variables can be used in the action. These are macros for access to   
 41060 data structures known internally to yacc.

41061 **\$\$** The value of the action can be set by assigning it to **\$\$**. If type   
 41062 checking is enabled and the type of the value to be assigned cannot   
 41063 be determined, a diagnostic message may be generated.

41064 **\$number** This refers to the value returned by the component specified by the   
 41065 token *number* in the right side of a rule, reading from left to right;   
 41066 *number* can be zero or negative. If it is, it refers to the data associated   
 41067 with the name on the parser's stack preceding the leftmost symbol of   
 41068 the current rule. (That is, "\$0" refers to the name immediately   
 41069 preceding the leftmost name in the current rule, to be found on the   
 41070 parser's stack and "\$-1" refers to the symbol to *its* left.) If *number*   
 41071 refers to an element past the current point in the rule, or beyond the   
 41072 bottom of the stack, the result is undefined. If type checking is   
 41073 enabled and the type of the value to be assigned cannot be   
 41074 determined, a diagnostic message may be generated.

41075 **\$(tag)number**

41076 These correspond exactly to the corresponding symbols without the   
 41077 *tag* inclusion, but allow for strict type checking (and preclude   
 41078 unwanted type conversions). The effect is that the macro is expanded   
 41079 to use *tag* to select an element from the YYSTYPE union (using   
 41080 *dataname.tag*). This is particularly useful if *number* is not positive.



```

41126 {...} This indicates C-language source code, with the possible inclusion of '$'
41127 macros as discussed previously.

41128 /* Grammar for the input to yacc. */
41129 /* Basic entries. */
41130 /* The following are recognized by the lexical analyzer. */

41131 %token IDENTIFIER /* Includes identifiers and literals */
41132 %token C_IDENTIFIER /* identifier (but not literal)
41133 followed by a :. */
41134 %token NUMBER /* [0-9][0-9]* */

41135 /* Reserved words : %type=>TYPE %left=>LEFT, and so on */

41136 %token LEFT RIGHT NONASSOC TOKEN PREC TYPE START UNION

41137 %token MARK /* The %% mark. */
41138 %token LCURL /* The %{ mark. */
41139 %token RCURL /* The }% mark. */

```

#### 41140 **Notes to Reviewers**

```

41141 This section with side shading will not appear in the final copy. - Ed.

41142 D3, XCU, ERN 293: An interpretation has been filed against 1003.2 and is likely to change "%}" to
41143 "%}.".

41144 /* 8-bit character literals stand for themselves; */
41145 /* tokens have to be defined for multi-byte characters. */

41146 %start spec

41147 %%

41148 spec : defs MARK rules tail
41149 ;
41150 tail : MARK
41151 {
41152 /* In this action, set up the rest of the file. */
41153 }
41154 | /* Empty; the second MARK is optional. */
41155 ;
41156 defs : /* Empty. */
41157 | defs def
41158 ;
41159 def : START IDENTIFIER
41160 | UNION
41161 {
41162 /* Copy union definition to output. */
41163 }
41164 | LCURL
41165 {
41166 /* Copy C code to output file. */
41167 }
41168 | RCURL
41169 | rword tag nlist
41170 ;
41171 rword : TOKEN

```



```

41172 | LEFT
41173 | RIGHT
41174 | NONASSOC
41175 | TYPE
41176 ;
41177 tag : /* Empty: union tag ID optional. */
41178 | '<' IDENTIFIER '>'
41179 ;
41180 nlist : nmno
41181 | nlist nmno
41182 ;
41183 nmno : IDENTIFIER /* Note: literal invalid with % type. */
41184 | IDENTIFIER NUMBER /* Note: invalid with % type. */
41185 ;

41186 /* Rule section */

41187 rules : C_IDENTIFIER rbody prec
41188 | rules rule
41189 ;
41190 rule : C_IDENTIFIER rbody prec
41191 | '|' rbody prec
41192 ;
41193 rbody : /* empty */
41194 | rbody IDENTIFIER
41195 | rbody act
41196 ;
41197 act : '{'
41198 | {
41199 /* Copy action, translate $$, and so on. */
4200 }
4201 | '}'
4202 ;
4203 prec : /* Empty */
4204 | PREC IDENTIFIER
4205 | PREC IDENTIFIER act
4206 | prec ';'
4207 ;

```

## 41208 Conflicts

41209 The parser produced for an input grammar may contain states in which conflicts occur. The  
41210 conflicts occur because the grammar is not LALR(1). An ambiguous grammar always contains at  
41211 least one LALR(1) conflict. The yacc utility shall resolve all conflicts, using either default rules or  
41212 user-specified precedence rules.

41213 Conflicts are either shift/reduce conflicts or reduce/reduce conflicts. A shift/reduce conflict is  
41214 where, for a given state and lookahead symbol, both a shift action and a reduce action are  
41215 possible. A reduce/reduce conflict is where, for a given state and lookahead symbol, reductions  
41216 by two different rules are possible.

41217 The rules below describe how to specify what actions to take when a conflict occurs. Not all  
41218 shift/reduce conflicts can be successfully resolved this way because the conflict may be due to  
41219 something other than ambiguity, so incautious use of these facilities can cause the language

41220 accepted by the parser to be much different from that which was intended. The description file  
 41221 shall contain sufficient information to understand the cause of the conflict. Where ambiguity is  
 41222 the reason either the default or explicit rules should be adequate to produce a working parser.

41223 The declared precedences and associativities (see **Declarations Section** (on page 3280)) are used  
 41224 to resolve parsing conflicts as follows:

- 41225 1. A precedence and associativity is associated with each grammar rule; it is the precedence  
 41226 and associativity of the last token or literal in the body of the rule. If the **%prec** keyword is  
 41227 used, it overrides this default. Some grammar rules might not have both precedence and  
 41228 associativity.
- 41229 2. If there is a shift/reduce conflict, and both the grammar rule and the input symbol have  
 41230 precedence and associativity associated with them, then the conflict is resolved in favor of  
 41231 the action (shift or reduce) associated with the higher precedence. If the precedences are  
 41232 the same, then the associativity is used; left associative implies reduce, right associative  
 41233 implies shift, and non-associative implies an error in the string being parsed.
- 41234 3. When there is a shift/reduce conflict that cannot be resolved by rule 2, the shift is done.  
 41235 Conflicts resolved this way are counted in the diagnostic output described in **Error**  
 41236 **Handling**.
- 41237 4. When there is a reduce/reduce conflict, a reduction is done by the grammar rule that  
 41238 occurs earlier in the input sequence. Conflicts resolved this way are counted in the  
 41239 diagnostic output described in **Error Handling**.

41240 Conflicts resolved by precedence or associativity shall not be counted in the shift/reduce and  
 41241 reduce/reduce conflicts reported by yacc on either standard error or in the description file.

#### 41242 **Error Handling**

41243 The token **error** shall be reserved for error handling. The name **error** can be used in grammar  
 41244 rules. It indicates places where the parser can recover from a syntax error. The default value of  
 41245 **error** shall be 256. Its value can be changed using a **%token** declaration. The lexical analyzer  
 41246 should not return the value of **error**.

41247 The parser shall detect a syntax error when it is in a state where the action associated with the  
 41248 lookahead symbol is **error**. A semantic action can cause the parser to initiate error handling by  
 41249 executing the macro YYERROR. When YYERROR is executed, the semantic action passes  
 41250 control back to the parser. YYERROR cannot be used outside of semantic actions.

41251 When the parser detects a syntax error, it normally calls *yyerror* with the character string  
 41252 "syntax error" as its argument. The call shall not be made if the parser is still recovering  
 41253 from a previous error when the error is detected. The parser is considered to be recovering from  
 41254 a previous error until the parser has shifted over at least three normal input symbols since the  
 41255 last error was detected or a semantic action has executed the macro *yyerrok*. The parser shall not  
 41256 call *yyerror* when YYERROR is executed.

41257 The macro function YYRECOVERING shall return 1 if a syntax error has been detected and the  
 41258 parser has not yet fully recovered from it. Otherwise, zero shall be returned.

41259 When a syntax error is detected by the parser, the parser shall check if a previous syntax error  
 41260 has been detected. If a previous error was detected, and if no normal input symbols have been  
 41261 shifted since the preceding error was detected, the parser checks if the lookahead symbol is an  
 41262 endmarker (see **Interface to the Lexical Analyzer** (on page 3287)). If it is, the parser shall return  
 41263 with a non-zero value. Otherwise, the lookahead symbol shall be discarded and normal parsing  
 41264 shall resume.

41265 When YYERROR is executed or when the parser detects a syntax error and no previous error has  
41266 been detected, or at least one normal input symbol has been shifted since the previous error was  
41267 detected, the parser shall pop back one state at a time until the parse stack is empty or the  
41268 current state allows a shift over **error**. If the parser empties the parse stack, it shall return with a  
41269 non-zero value. Otherwise, it shall shift over **error** and then resume normal parsing. If the parser  
41270 reads a lookahead symbol before the error was detected, that symbol shall still be the lookahead  
41271 symbol when parsing is resumed.

41272 The macro *yyerrok* in a semantic action shall cause the parser to act as if it has fully recovered  
41273 from any previous errors. The macro *yyclearin* shall cause the parser to discard the current  
41274 lookahead token. If the current lookahead token has not yet been read, *yyclearin* shall have no  
41275 effect.

41276 The macro YYACCEPT shall cause the parser to return with the value zero. The macro  
41277 YYABORT shall cause the parser to return with a non-zero value.

### 41278 **Interface to the Lexical Analyzer**

41279 The *yylex* function is an integer-valued function that returns a *token number* representing the kind  
41280 of token read. If there is a value associated with the token returned by *yylex* (see the discussion  
41281 of *tag* above), it shall be assigned to the external variable *yyval*.

41282 If the parser and *yylex* do not agree on these token numbers, reliable communication between  
41283 them cannot occur. For (one character) literals, the token is simply the numeric value of the  
41284 character in the current character set. The numbers for other tokens can either be chosen by *yacc*,  
41285 or chosen by the user. In either case, the **#define** construct of C is used to allow *yylex* to return  
41286 these numbers symbolically. The **#define** statements are put into the code file, and the header  
41287 file if that file is requested. The set of characters permitted by *yacc* in an identifier is larger than  
41288 that permitted by C. Token names found to contain such characters shall not be included in the  
41289 **#define** declarations.

41290 If the token numbers are chosen by *yacc*, the tokens other than literals shall be assigned numbers  
41291 greater than 256, although no order is implied. A token can be explicitly assigned a number by  
41292 following its first appearance in the declarations section with a number. Names and literals not  
41293 defined this way retain their default definition. All token numbers assigned by *yacc* shall be  
41294 unique and distinct from the token numbers used for literals and user-assigned tokens. If  
41295 duplicate token numbers cause conflicts in parser generation, *yacc* shall report an error;  
41296 otherwise, it is unspecified whether the token assignment is accepted or an error is reported.

41297 The end of the input is marked by a special token called the *endmarker*, which has a token  
41298 number that is zero or negative. (These values are invalid for any other token.) All lexical  
41299 analyzers shall return zero or negative as a token number upon reaching the end of their input. If  
41300 the tokens up to, but excluding, the endmarker form a structure that matches the start symbol,  
41301 the parser shall accept the input. If the endmarker is seen in any other context, it shall be  
41302 considered an error.

### 41303 **Completing the Program**

41304 In addition to *yparse* and *yylex*, the functions *yyerror* and *main* are required to make a complete  
41305 program. The application can supply *main* and *yyerror*, or those routines can be obtained from  
41306 the *yacc* library.

**41307 Yacc Library**

41308 The following functions appear only in the *yacc* library accessible through the `-l y` operand to *cc*  
41309 or *c99*; they can therefore be redefined by a portable application:

**41310 `int main(void)`**

41311 This function shall call *yparse* and exit with an unspecified value. Other actions within this  
41312 function are unspecified.

**41313 `int yyerror(const char *s)`**

41314 This function shall write the NUL-terminated argument to standard error, followed by a  
41315 `<newline>` character.

41316 The order of the `-l y` and `-l l` operands given to *cc* or *c99* is significant; the application shall  
41317 either provide its own *main* function or ensure that `-l y` precedes `-l l`.

**41318 Debugging the Parser**

41319 The parser generated by *yacc* shall have diagnostic facilities in it that can be optionally enabled  
41320 at either compile time or at runtime (if enabled at compile time). The compilation of the runtime  
41321 debugging code is under the control of `YYDEBUG`, a preprocessor symbol. If `YYDEBUG` has a  
41322 non-zero value, the debugging code shall be included. If its value is zero, the code shall not be  
41323 included.

41324 In parsers where the debugging code has been included, the external `int yydebug` can be used to  
41325 turn debugging on (with a non-zero value) and off (zero value) at runtime. The initial value of  
41326 *yydebug* shall be zero.

41327 When `-t` is specified, the code file shall be built such that, if `YYDEBUG` is not already defined at  
41328 compilation time (using the *c99* `-D YYDEBUG` option, for example), `YYDEBUG` shall be set  
41329 explicitly to 1. When `-t` is not specified, the code file shall be built such that, if `YYDEBUG` is not  
41330 already defined, it shall be set explicitly to zero.

41331 The format of the debugging output is unspecified but includes at least enough information to  
41332 determine the shift and reduce actions, and the input symbols. It also provides information  
41333 about error recovery.

**41334 Algorithms**

41335 The parser constructed by *yacc* implements an LALR(1) parsing algorithm as documented in the  
41336 literature. It is unspecified whether the parser is table-driven or direct-coded.

41337 A parser generated by *yacc* shall never request an input symbol from *yylex* while in a state where  
41338 the only actions other than the error action are reductions by a single rule.

41339 The literature of parsing theory defines these concepts.

**41340 Limits**

41341 The *yacc* utility may have several internal tables. The minimum maximums for these tables are  
41342 shown in the following table. The exact meaning of these values is implementation-defined. The  
41343 implementation shall define the relationship between these values and between them and any  
41344 error messages that the implementation may generate should it run out of space for any internal  
41345 structure. An implementation may combine groups of these resources into a single pool as long  
41346 as the total available to the user does not fall below the sum of the sizes specified by this section.

41347

Table 4-22 Internal Limits in yacc

41348

41349

41350

41351

41352

41353

41354

41355

41356

41357

41358

41359

41360

41361

41362

41363

41364

| Limit      | Minimum<br>Maximum | Description                                                                                                                                                                                                                                               |
|------------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| {NTERMS}   | 126                | Number of tokens.                                                                                                                                                                                                                                         |
| {NNONTERM} | 200                | Number of non-terminals.                                                                                                                                                                                                                                  |
| {NPROD}    | 300                | Number of rules.                                                                                                                                                                                                                                          |
| {NSTATES}  | 600                | Number of states.                                                                                                                                                                                                                                         |
| {MEMSIZE}  | 5 200              | Length of rules. The total length, in names (tokens and non-terminals), of all the rules of the grammar. The left-hand side is counted for each rule, even if it is not explicitly repeated, as specified in <b>Grammar Rules in yacc</b> (on page 3281). |
| {ACTSIZE}  | 4 000              | Number of actions. “Actions” here (and in the description file) refer to parser actions (shift, reduce, and so on) not to semantic actions defined in <b>Grammar Rules in yacc</b> (on page 3281).                                                        |

41365 **EXIT STATUS**

41366 The following exit values shall be returned:

41367 0 Successful completion.

41368 &gt;0 An error occurred.

41369 **CONSEQUENCES OF ERRORS**

41370 If any errors are encountered, the run is aborted and yacc exits with a non-zero status. Partial  
 41371 code files and header files may be produced. The summary information in the description  
 41372 file always shall be produced if the `-v` flag is present.

41373 **APPLICATION USAGE**

41374 Historical implementations experience name conflicts on the names `yacc.tmp`, `yacc.acts`,  
 41375 `yacc.debug`, `y.tab.c`, `y.tab.h`, and `y.output` if more than one copy of yacc is running in a single  
 41376 directory at one time. The `-b` option was added to overcome this problem. The related problem  
 41377 of allowing multiple yacc parsers to be placed in the same file was addressed by adding a `-p`  
 41378 option to override the previously hard-coded `yy` variable prefix.

41379 The description of the `-p` option specifies the minimal set of function and variable names that  
 41380 cause conflict when multiple parsers are linked together. `YYSTYPE` does not need to be changed.  
 41381 Instead, the programmer can use `-b` to give the header files for different parsers different names,  
 41382 and then the file with the `yylex` for a given parser can include the header for that parser. Names  
 41383 such as `yyclearerr` do not need to be changed because they are used only in the actions; they do  
 41384 not have linkage. It is possible that an implementation has other names, either internal ones for  
 41385 implementing things such as `yyclearerr`, or providing non-standard features that it wants to  
 41386 change with `-p`.

41387 Unary operators that are the same token as a binary operator in general need their precedence  
 41388 adjusted. This is handled by the `%prec` advisory symbol associated with the particular grammar  
 41389 rule defining that unary operator. (See **Grammar Rules in yacc** (on page 3281).) Applications  
 41390 are not required to use this operator for unary operators, but the grammars that do not require it  
 41391 are rare.

41392 **EXAMPLES**

41393 Access to the *yacc* library is obtained with library search operands to *cc* or *c99*. To use the *yacc* library *main*:

```
41395 c99 y.tab.c -l y
```

41396 Both the *lex* library and the *yacc* library contain *main*. To access the *yacc main*:

```
41397 c99 y.tab.c lex.yy.c -l y -l l
```

41398 This ensures that the *yacc* library is searched first, so that its *main* is used.

41399 The historical *yacc* libraries have contained two simple functions that are normally coded by the application programmer. These library functions are similar to the following code:

```
41401 #include <locale.h>
41402 int main(void)
41403 {
41404 extern int yyparse();
41405 setlocale(LC_ALL, "");
41406 /* If the following parser is one created by lex, the
41407 application must be careful to ensure that LC_CTYPE
41408 and LC_COLLATE are set to the POSIX locale. */
41409 (void) yyparse();
41410 return (0);
41411 }
41412 #include <stdio.h>
41413 int yyerror(const char *msg)
41414 {
41415 (void) fprintf(stderr, "%s\n", msg);
41416 return (0);
41417 }
```

41418 **RATIONALE**

41419 The references in **Referenced Documents** (on page xv) may be helpful in constructing the parser generator. The referenced DeRemer and Pennello article (along with the works it references) describes a technique to generate parsers that conform to this volume of IEEE Std. 1003.1-200x. Work in this area continues to be done, so implementors should consult current literature before doing any new implementations. The original Knuth article is the theoretical basis for this kind of parser, but the tables it generates are impractically large for reasonable grammars and should not be used. The “equivalent to” wording is intentional to assure that the best tables that are LALR(1) can be generated.

41427 There has been confusion between the class of grammars, the algorithms needed to generate parsers, and the algorithms needed to parse the languages. They are all reasonably orthogonal. In particular, a parser generator that accepts the full range of LR(1) grammars need not generate a table any more complex than one that accepts SLR(1) (a relatively weak class of LR grammars) for a grammar that happens to be SLR(1). Such an implementation need not recognize the case, either; table compression can yield the SLR(1) table (or one even smaller than that) without recognizing that the grammar is SLR(1). The speed of an LR(1) parser for any class is dependent more upon the table representation and compression (or the code generation if a direct parser is generated) than upon the class of grammar that the table generator handles.

41436 The speed of the parser generator is somewhat dependent upon the class of grammar it handles. However, the original Knuth article algorithms for constructing LR parsers was judged by its

41438 author to be impractically slow at that time. Although full LR is more complex than LALR(1), as  
41439 computer speeds and algorithms improve, the difference (in terms of acceptable wall-clock  
41440 execution time) is becoming less significant.

41441 Potential authors are cautioned that the referenced DeRemer and Pennello article previously  
41442 cited identifies a bug (an over-simplification of the computation of LALR(1) lookahead sets) in  
41443 some of the LALR(1) algorithm statements that preceded it to publication. They should take the  
41444 time to seek out that paper, as well as current relevant work, particularly Aho's.

41445 The **-b** option was added to provide a portable method for permitting *yacc* to work on multiple  
41446 separate parsers in the same directory. If a directory contains more than one *yacc* grammar, and  
41447 both grammars are constructed at the same time (by, for example, a parallel *make* program),  
41448 conflict results. While the solution is not historical practice, it corrects a known deficiency in  
41449 historical implementations. Corresponding changes were made to all sections that referenced  
41450 the file names **y.tab.c** (now "the code file"), **y.tab.h** (now "the header file"), and **y.output** (now  
41451 "the description file").

41452 The grammar for *yacc* input is based on System V documentation. The textual description shows  
41453 there that the `' ; '` is required at the end of the rule. The grammar and the implementation do not  
41454 require this. (The use of **C\_IDENTIFIER** causes a reduce to occur in the right place.)

41455 Also, in that implementation, the constructs such as **%token** can be terminated by a semicolon,  
41456 but this is not permitted by the grammar. The keywords such as **%token** can also appear in  
41457 uppercase, which is again not discussed. In most places where `'%'` is used, `'\'` can be  
41458 substituted, and there are alternate spellings for some of the symbols (for example, **%LEFT** can  
41459 be `"%<"` or even `"\<"`).

41460 Historically, `<tag>` can contain any characters except `'>'`, including white space, in the  
41461 implementation. However, since the *tag* must reference a ISO C standard union member, in  
41462 practice conforming implementations need to support only the set of characters for ISO C  
41463 standard identifiers in this context.

41464 Some historical implementations are known to accept actions that are terminated by a period.  
41465 Historical implementations often allow `'$'` in names. A conforming implementation does not  
41466 need to support either of these behaviors.

41467 Deciding when to use **%prec** illustrates the difficulty in specifying the behavior of *yacc*. There  
41468 may be situations in which the *grammar* is not, strictly speaking, in error, and yet *yacc* cannot  
41469 interpret it unambiguously. The resolution of ambiguities in the grammar can in many instances  
41470 be resolved by providing additional information, such as using **%type** or **%union** declarations. It  
41471 is often easier and it usually yields a smaller parser to take this alternative when it is  
41472 appropriate.

41473 The size and execution time of a program produced without the runtime debugging code is  
41474 usually smaller and slightly faster in historical implementations.

41475 Statistics messages from several historical implementations include the following types of  
41476 information:

41477 *n*/512 terminals, *n*/300 non-terminals  
 41478 *n*/600 grammar rules, *n*/1 500 states  
 41479 *n* shift/reduce, *n* reduce/reduce conflicts reported  
 41480 *n*/350 working sets used  
 41481 Memory: states, etc. *n*/15 000, parser *n*/15 000  
 41482 *n*/600 distinct lookahead sets  
 41483 *n* extra closures  
 41484 *n* shift entries, *n* exceptions  
 41485 *n* goto entries  
 41486 *n* entries saved by goto default  
 41487 Optimizer space used: input *n*/15 000, output *n*/15 000  
 41488 *n* table entries, *n* zero  
 41489 Maximum spread: *n*, Maximum offset: *n*

41490 The report of internal tables in the description file is left implementation-defined because all  
 41491 aspects of these limits are also implementation-defined. Some implementations may use  
 41492 dynamic allocation techniques and have no specific limit values to report.

41493 The format of the **y.output** file is not given because specification of the format was not seen to  
 41494 enhance applications portability. The listing is primarily intended to help human users  
 41495 understand and debug the parser; use of **y.output** by a portable application script would be  
 41496 unusual. Furthermore, implementations have not produced consistent output and no popular  
 41497 format was apparent. The format selected by the implementation should be human-readable, in  
 41498 addition to the requirement that it be a text file.

41499 Standard error reports are not specifically described because they are seldom of use to portable  
 41500 applications and there was no reason to restrict implementations.

41501 Some implementations recognize "*= {*" as equivalent to '*{*' because it appears in historical  
 41502 documentation. This construction was recognized and documented as obsolete as long ago as  
 41503 1978, in the referenced *Yacc: Yet Another Compiler-Compiler*. This volume of IEEE Std. 1003.1-200x  
 41504 chose to leave it as obsolete and omit it.

41505 Multi-byte characters should be recognized by the lexical analyzer and returned as tokens. They  
 41506 should not be returned as multi-byte character literals. The token **error** that is used for error  
 41507 recovery is normally assigned the value 256 in the historical implementation. Thus, the token  
 41508 value 256, which used in many multi-byte character sets, is not available for use as the value of a  
 41509 user-defined token.

#### 41510 **FUTURE DIRECTIONS**

41511 None.

#### 41512 **SEE ALSO**

41513 *c99*, *lex*

#### 41514 **CHANGE HISTORY**

41515 First released in Issue 2.

#### 41516 **Issue 4**

41517 Aligned with the ISO/IEC 9945-2: 1993 standard.

#### 41518 **Issue 5**

41519 FUTURE DIRECTIONS section added.



41520 **Issue 6**

41521 Minor changes have been added to align with the IEEE P1003.2b draft standard.

41522 The normative text is reworded to avoid use of the term “must” for application requirements.

41523 **NAME**

41524 zcat — expand and concatenate data

41525 **SYNOPSIS**41526 XSI zcat [*file...*]

41527

41528 **DESCRIPTION**

41529 The *zcat* utility shall write to standard output the uncompressed form of files that have been  
41530 compressed using the *compress* utility. It is the equivalent of *uncompress -c*. Input files are not  
41531 affected.

41532 **OPTIONS**

41533 None.

41534 **OPERANDS**

41535 The following operand shall be supported:

41536 *file* The path name of a file previously processed by the *compress* utility. If *file* already  
41537 has the *.Z* suffix specified, it is used as submitted. Otherwise, the *.Z* suffix is  
41538 appended to the file name prior to processing.

41539 **STDIN**41540 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '- '.41541 **INPUT FILES**41542 Input files shall be compressed files that are in the format produced by the *compress* utility.41543 **ENVIRONMENT VARIABLES**41544 The following environment variables shall affect the execution of *zcat*:

41545 *LANG* Provide a default value for the internationalization variables that are unset or null.  
41546 If *LANG* is unset or null, the corresponding value from the implementation-  
41547 defined default locale shall be used. If any of the internationalization variables  
41548 contains an invalid setting, the utility shall behave as if none of the variables had  
41549 been defined.

41550 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
41551 internationalization variables.

41552 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
41553 characters (for example, single-byte as opposed to multi-byte characters in  
41554 arguments).

41555 *LC\_MESSAGES*  
41556 Determine the locale that should be used to affect the format and contents of  
41557 diagnostic messages written to standard error.

41558 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

41559 **ASYNCHRONOUS EVENTS**

41560 Default.

41561 **STDOUT**

41562 The compressed files given as input shall be written on standard output in their uncompressed  
41563 form.

41564 **STDERR**

41565           Used only for diagnostic messages.

41566 **OUTPUT FILES**

41567           None.

41568 **EXTENDED DESCRIPTION**

41569           None.

41570 **EXIT STATUS**

41571           The following exit values shall be returned:

41572           0   Successful completion.

41573           >0  An error occurred.

41574 **CONSEQUENCES OF ERRORS**

41575           Default.

41576 **APPLICATION USAGE**

41577           None.

41578 **EXAMPLES**

41579           None.

41580 **RATIONALE**

41581           None.

41582 **FUTURE DIRECTIONS**

41583           None.

41584 **SEE ALSO**

41585           *compress, uncompress*

41586 **CHANGE HISTORY**

41587           First released in Issue 4.



# Index

1

|    |                                          |             |                                       |             |
|----|------------------------------------------|-------------|---------------------------------------|-------------|
| 2  | <control>-V.....                         | 2578        | background work                       |             |
| 3  | <control>-W .....                        | 2578        | at .....                              | 2358        |
| 4  | _POSIX_VDISABLE.....                     | 3102        | batch.....                            | 2404        |
| 5  | Account_Name .....                       | 2321        | bg.....                               | 2422        |
| 6  | actions equivalent to functions .....    | 2211        | crontab.....                          | 2488        |
| 7  | admin.....                               | <b>2340</b> | fg.....                               | 2652        |
| 8  | ADV.....                                 | <b>2212</b> | jobs .....                            | 2730        |
| 9  | AIO .....                                | <b>2212</b> | nice .....                            | 2872        |
| 10 | alias.....                               | <b>2345</b> | nohup.....                            | 2885        |
| 11 | alias substitution .....                 | 2239        | renice.....                           | 3028        |
| 12 | AND lists.....                           | 2260        | BAR.....                              | <b>2212</b> |
| 13 | AND-OR list .....                        | 2259        | basename.....                         | <b>2401</b> |
| 14 | appending redirected output.....         | 2252        | batch.....                            | <b>2404</b> |
| 15 | ar .....                                 | <b>2348</b> | batch administration.....             | 2316        |
| 16 | archives                                 |             | batch authorization.....              | 2316        |
| 17 | ar command.....                          | 2348        | batch client-server interaction ..... | 2313        |
| 18 | ARG_MAX.....                             | 3275        | batch environment                     |             |
| 19 | arithmetic expansion .....               | 2248        | services .....                        | 2313        |
| 20 | arithmetic language                      |             | utilities.....                        | 2313        |
| 21 | bc.....                                  | 2407        | Batch Job Abort.....                  | 2316, 2327  |
| 22 | array identifiers.....                   | 2412        | batch job creation .....              | 2314        |
| 23 | asa.....                                 | <b>2355</b> | Batch Job Execution.....              | 2315, 2319  |
| 24 | asynchronous lists.....                  | 2259        | Batch Job Exit .....                  | 2316, 2326  |
| 25 | at .....                                 | <b>2358</b> | batch job identifier .....            | 2336        |
| 26 | at-job.....                              | 2358        | Batch Job Message Request .....       | 2329        |
| 27 | automatic storage class .....            | 2416        | Batch Job Routing .....               | 2315, 2326  |
| 28 | awk .....                                | <b>2367</b> | batch job states.....                 | 2318        |
| 29 | actions.....                             | 2378        | Batch Job Status Request.....         | 2330        |
| 30 | arithmetic functions.....                | 2380        | batch job tracking .....              | 2315        |
| 31 | escape sequences.....                    | 2376        | batch notification.....               | 2317        |
| 32 | expression patterns.....                 | 2378        | batch queue.....                      | 2314        |
| 33 | expressions .....                        | 2370        | Batch Queue Status Request .....      | 2332        |
| 34 | functions.....                           | 2380        | Batch Server Restart.....             | 2327        |
| 35 | grammar.....                             | 2384        | batch services .....                  | 2317        |
| 36 | input/output and general functions ..... | 2382        | bc.....                               | <b>2407</b> |
| 37 | lexical conventions.....                 | 2390        | grammar .....                         | 2408        |
| 38 | output statements .....                  | 2379        | lexical conventions.....              | 2410        |
| 39 | overall program structure .....          | 2369        | operations .....                      | 2412        |
| 40 | pattern ranges .....                     | 2378        | operators .....                       | 2412        |
| 41 | patterns.....                            | 2377        | bcc (mailer blind carbon copy) .....  | 2816        |
| 42 | regular expressions.....                 | 2375        | BC_BASE_MAX .....                     | 2221        |
| 43 | special patterns.....                    | 2377        | BC_DIM_MAX.....                       | 2221        |
| 44 | string functions.....                    | 2381        | BC_SCALE_MAX.....                     | 2221        |
| 45 | user-defined functions .....             | 2383        | BC_STRING_MAX .....                   | 2221, 2410  |
| 46 | variables and special variables.....     | 2373        | BE.....                               | <b>2212</b> |
| 47 |                                          |             | bg .....                              | <b>2422</b> |

|    |                                              |             |                                        |             |
|----|----------------------------------------------|-------------|----------------------------------------|-------------|
| 48 | binary primaries .....                       | 3119        | write .....                            | 3268        |
| 49 | break.....                                   | <b>2277</b> | compilers                              |             |
| 50 | break special built-in .....                 | 2277        | fort77 .....                           | 2672        |
| 51 | builtin.....                                 | 2475        | yacc.....                              | 3277        |
| 52 | c89                                          |             | compound commands.....                 | 2261        |
| 53 | external symbols.....                        | 2429        | compound-list .....                    | 2259        |
| 54 | standard libraries .....                     | 2428        | compress .....                         | <b>2477</b> |
| 55 | c99.....                                     | <b>2425</b> | compression                            |             |
| 56 | cal.....                                     | <b>2433</b> | compress .....                         | 2477        |
| 57 | can.....                                     | 2205        | uncompress .....                       | 3163        |
| 58 | carriage-control characters.....             | 2355        | zcat .....                             | 3294        |
| 59 | case conditional construct .....             | 2262        | concurrent execution of processes..... | 2208        |
| 60 | cat.....                                     | <b>2435</b> | configuration values .....             | 2692        |
| 61 | cc (mailer carbon copy) .....                | 2816        | conforming application.....            | 2353        |
| 62 | CD.....                                      | <b>2212</b> | consequences of shell errors.....      | 2255        |
| 63 | cd.....                                      | <b>2439</b> | continue .....                         | <b>2281</b> |
| 64 | cflow.....                                   | <b>2444</b> | control characters .....               | 3099        |
| 65 | changing the current working directory ..... | 2211        | controlling terminal .....             | 2208        |
| 66 | character counting.....                      | 3258        | Coordinated Universal Time (UTC).....  | 2513        |
| 67 | charmap                                      |             | copy files commands                    |             |
| 68 | with localedef.....                          | 2766        | cp .....                               | 2480        |
| 69 | writing names with locale .....              | 2761        | dd.....                                | 2515        |
| 70 | charmap file.....                            | 2765, 3102  | ln .....                               | 2757        |
| 71 | CHAR_BIT.....                                | 2558        | mv.....                                | 2863        |
| 72 | Checkpoint.....                              | 2321        | pax .....                              | 2910        |
| 73 | checksums                                    |             | cp.....                                | <b>2480</b> |
| 74 | cksum.....                                   | 2460        | cpio format.....                       | 2929        |
| 75 | chgrp .....                                  | <b>2447</b> | CPT .....                              | <b>2213</b> |
| 76 | chmod .....                                  | <b>2450</b> | CPU time .....                         | 3127        |
| 77 | grammar.....                                 | 2453        | cron daemon.....                       | 2491        |
| 78 | chown.....                                   | <b>2456</b> | crontab .....                          | <b>2488</b> |
| 79 | cksum.....                                   | <b>2460</b> | CS.....                                | <b>2213</b> |
| 80 | cmp.....                                     | <b>2465</b> | csplit .....                           | <b>2492</b> |
| 81 | codeset conversion.....                      | 2714        | ctags.....                             | <b>2496</b> |
| 82 | tr.....                                      | 3136        | current working directory .....        | 2208        |
| 83 | COLL_WEIGHTS_MAX .....                       | 2221        | cut .....                              | <b>2501</b> |
| 84 | colon.....                                   | <b>2279</b> | CX .....                               | <b>2213</b> |
| 85 | colon special built-in .....                 | 2279, 2281  | cxref.....                             | <b>2505</b> |
| 86 | comm .....                                   | <b>2468</b> | data keywords.....                     | 2956        |
| 87 | command .....                                | <b>2471</b> | date.....                              | <b>2508</b> |
| 88 | command mode.....                            | 2546        | field descriptors .....                | 2508        |
| 89 | command search and execution.....            | 2257        | modified field descriptors .....       | 2509        |
| 90 | command substitution .....                   | 2247        | dd .....                               | <b>2515</b> |
| 91 | communications commands                      |             | default queue.....                     | 2332        |
| 92 | mailx .....                                  | 2794        | deferred batch services.....           | 2319        |
| 93 | talk.....                                    | 3111        | delta.....                             | <b>2522</b> |
| 94 | uucp .....                                   | 3178        | destination .....                      | 2337        |
| 95 | uudecode.....                                | 3182        | df.....                                | <b>2525</b> |
| 96 | uuencode.....                                | 3185        | diff.....                              | <b>2529</b> |
| 97 | uustat .....                                 | 3190        | default output format.....             | 2531        |
| 98 | uux.....                                     | 3193        | directory comparison format.....       | 2530        |

## Index

|     |                                             |             |                                    |             |
|-----|---------------------------------------------|-------------|------------------------------------|-------------|
| 99  | -c or -C output format .....                | 2532        | substitute command.....            | 2554        |
| 100 | -e output format .....                      | 2532        | undo command.....                  | 2555        |
| 101 | -f output format .....                      | 2532        | write command.....                 | 2556        |
| 102 | directory commands                          |             | edit buffer .....                  | 2565, 3200  |
| 103 | cd .....                                    | 2439        | editors                            |             |
| 104 | pwd .....                                   | 2967        | ed .....                           | 2546        |
| 105 | directory lister .....                      | 2779        | ex.....                            | 2565        |
| 106 | dirname.....                                | <b>2536</b> | sed.....                           | 3051        |
| 107 | disk space commands                         |             | vi.....                            | 3200        |
| 108 | df.....                                     | 2525        | ED_FILE_MAX.....                   | 2558        |
| 109 | du.....                                     | 2539        | ED_LINE_MAX .....                  | 2558        |
| 110 | ulimit.....                                 | 3151        | effective group ID.....            | 2208        |
| 111 | documentation .....                         | 2838        | effective user ID.....             | 2208        |
| 112 | dot.....                                    | <b>2283</b> | Eighth Edition UNIX .....          | 2475        |
| 113 | dot special built-in.....                   | 2283        | env .....                          | <b>2562</b> |
| 114 | double-quotes.....                          | 2236        | EPERM.....                         | 2485        |
| 115 | du .....                                    | <b>2539</b> | Error_Path .....                   | 2322        |
| 116 | duplicating an input file descriptor.....   | 2253        | escape character (backslash) ..... | 2236        |
| 117 | duplicating an output file descriptor ..... | 2253        | escape sequences                   |             |
| 118 | echo .....                                  | <b>2543</b> | awk .....                          | 2376        |
| 119 | ed.....                                     | <b>2546</b> | gencat.....                        | 2683        |
| 120 | addresses.....                              | 2548        | lex .....                          | 2748        |
| 121 | append command .....                        | 2550        | establish the locale .....         | 2211        |
| 122 | change command .....                        | 2550        | eval .....                         | <b>2285</b> |
| 123 | commands .....                              | 2549        | eval special built-in.....         | 2285        |
| 124 | copy command .....                          | 2555        | ex.....                            | <b>2565</b> |
| 125 | delete command .....                        | 2551        | <backslash>.....                   | 2578        |
| 126 | edit command .....                          | 2551        | <control>-D command.....           | 2600        |
| 127 | edit without checking command.....          | 2551        | <newline> .....                    | 2577        |
| 128 | file name command .....                     | 2551        | abbreviate command.....            | 2580        |
| 129 | global command.....                         | 2551        | addressing.....                    | 2571        |
| 130 | global non-matched command .....            | 2556        | adjust window command.....         | 2598        |
| 131 | help command .....                          | 2552        | append command .....               | 2581        |
| 132 | help-mode command .....                     | 2552        | args command .....                 | 2581        |
| 133 | insert command.....                         | 2552        | autoindent option.....             | 2602        |
| 134 | interactive global command .....            | 2552        | autoprint option .....             | 2603        |
| 135 | interactive global not-matched command..... | 2556        | autowrite option.....              | 2603        |
| 136 | join command .....                          | 2553        | beautify option.....               | 2603        |
| 137 | line number command.....                    | 2556        | change command .....               | 2581        |
| 138 | list command.....                           | 2553        | chdir command.....                 | 2582        |
| 139 | mark command.....                           | 2553        | command descriptions.....          | 2578        |
| 140 | move command.....                           | 2553        | copy command .....                 | 2582        |
| 141 | null command.....                           | 2557        | delete command .....               | 2582        |
| 142 | number command.....                         | 2553        | directory option .....             | 2603        |
| 143 | print command .....                         | 2554        | edcompatible option.....           | 2604        |
| 144 | prompt command .....                        | 2554        | edit command .....                 | 2582        |
| 145 | quit command .....                          | 2554        | edit options.....                  | 2602        |
| 146 | quit without checking command.....          | 2554        | errorbells option .....            | 2604        |
| 147 | read command .....                          | 2554        | escape command .....               | 2599        |
| 148 | regular expressions .....                   | 2547        | execute command .....              | 2601        |
| 149 | shell escape command .....                  | 2556        | execr command.....                 | 2604        |

|     |                           |            |                                  |                                    |
|-----|---------------------------|------------|----------------------------------|------------------------------------|
| 150 | file command .....        | 2583       | tags command.....                | 2608                               |
| 151 | global command.....       | 2584       | term command.....                | 2608                               |
| 152 | ignorecase option.....    | 2604       | terse command .....              | 2608                               |
| 153 | initialization .....      | 2569       | unabbrev command.....            | 2595                               |
| 154 | input editing.....        | 2576       | undo command.....                | 2595                               |
| 155 | insert command.....       | 2585       | unmap command.....               | 2595                               |
| 156 | join command .....        | 2585       | version command.....             | 2596                               |
| 157 | lisp command.....         | 2604       | visual command .....             | 2596                               |
| 158 | list command .....        | 2586, 2604 | warn command.....                | 2608                               |
| 159 | magic command.....        | 2605       | window command.....              | 2609                               |
| 160 | map command.....          | 2586       | wrapmargin option.....           | 2609                               |
| 161 | mark command.....         | 2587       | wrapscan option.....             | 2609                               |
| 162 | mesg command .....        | 2605       | write command.....               | 2596                               |
| 163 | move command.....         | 2588       | write line number command .....  | 2601                               |
| 164 | next command .....        | 2588       | writeln option .....             | 2610                               |
| 165 | number command.....       | 2589       | xit command .....                | 2597                               |
| 166 | number option .....       | 2605       | yank command .....               | 2598                               |
| 167 | open command .....        | 2589       | exec.....                        | <b>2287</b>                        |
| 168 | paragraphs option.....    | 2605       | Fam.....                         | 2886                               |
| 169 | preserve.....             | 2565       | exec family .....                | 2475, 2870, 3275                   |
| 170 | preserve command .....    | 2589       | exec special built-in.....       | 2287                               |
| 171 | print command .....       | 2590       | Execution_Time .....             | 2323                               |
| 172 | prompt command .....      | 2605       | EXINIT.....                      | 2565                               |
| 173 | put command.....          | 2590       | exit .....                       | <b>2289</b>                        |
| 174 | quit command.....         | 2590       | exit special built-in.....       | 2289                               |
| 175 | read command.....         | 2591       | exit status and errors .....     | 2255                               |
| 176 | readonly command.....     | 2606       | exit status for commands.....    | 2255                               |
| 177 | recover command .....     | 2591       | expand .....                     | <b>2636</b>                        |
| 178 | redraw command.....       | 2606       | export .....                     | <b>2291</b>                        |
| 179 | regular expressions ..... | 2601       | export special built-in.....     | 2291                               |
| 180 | remap command .....       | 2606       | expr.....                        | <b>2639</b>                        |
| 181 | replacement strings.....  | 2602       | matching expression.....         | 2641                               |
| 182 | report command.....       | 2606       | string operand.....              | 2641                               |
| 183 | rewind command.....       | 2592       | expression argument .....        | 2379                               |
| 184 | scroll command.....       | 2577, 2607 | expression list.....             | 2379                               |
| 185 | sections command .....    | 2607       | EXPR_NEST_MAX.....               | 2221                               |
| 186 | set command.....          | 2592       | extended regular expression..... | 2367, 2375                         |
| 187 | shell command.....        | 2592       | .....                            | 2484, 2664, 2703, 2747, 2865, 2919 |
| 188 | shell option .....        | 2607       | .....                            | 3033, 3273                         |
| 189 | shift left command .....  | 2600       | extension                        |                                    |
| 190 | shift right command ..... | 2600       | CX.....                          | 2213                               |
| 191 | shiftwidth option.....    | 2607       | OH.....                          | 2214                               |
| 192 | showmatch option .....    | 2607       | XSI .....                        | 2218                               |
| 193 | showmode command.....     | 2607       | false.....                       | <b>2644</b>                        |
| 194 | slowopen command .....    | 2608       | fc .....                         | <b>2646</b>                        |
| 195 | source command .....      | 2593       | FD.....                          | <b>2213</b>                        |
| 196 | substitute command.....   | 2593       | fg.....                          | <b>2652</b>                        |
| 197 | suspend command.....      | 2594       | field splitting .....            | 2249                               |
| 198 | tabstop option.....       | 2608       | FIFO special files .....         | 2848                               |
| 199 | tag command .....         | 2594       | file .....                       | <b>2654</b>                        |
| 200 | taglength option .....    | 2608       | file access permissions .....    | 2208                               |



## Index

|     |                                  |             |  |
|-----|----------------------------------|-------------|--|
| 201 | file comparisons                 |             |  |
| 202 | cmp.....                         | 2465        |  |
| 203 | comm.....                        | 2468        |  |
| 204 | diff.....                        | 2529        |  |
| 205 | uniq.....                        | 3172        |  |
| 206 | file contents.....               | 2210        |  |
| 207 | file conversion                  |             |  |
| 208 | cut.....                         | 2501        |  |
| 209 | dd.....                          | 2515        |  |
| 210 | expand.....                      | 2636        |  |
| 211 | fold.....                        | 2669        |  |
| 212 | head.....                        | 2711        |  |
| 213 | join.....                        | 2734        |  |
| 214 | od.....                          | 2888        |  |
| 215 | paste.....                       | 2896        |  |
| 216 | patch.....                       | 2900        |  |
| 217 | sort.....                        | 3080        |  |
| 218 | strings.....                     | 3089        |  |
| 219 | tail.....                        | 3107        |  |
| 220 | tr.....                          | 3136        |  |
| 221 | tsort.....                       | 3144        |  |
| 222 | unexpand.....                    | 3166        |  |
| 223 | uniq.....                        | 3172        |  |
| 224 | uudecode.....                    | 3182        |  |
| 225 | uuencode.....                    | 3185        |  |
| 226 | file creation.....               | 2209        |  |
| 227 | file descriptor.....             | 2208, 2251  |  |
| 228 | file mode creation mask.....     | 2208        |  |
| 229 | file permission commands         |             |  |
| 230 | chgrp.....                       | 2447        |  |
| 231 | chmod.....                       | 2450        |  |
| 232 | chown.....                       | 2456        |  |
| 233 | umask.....                       | 3154        |  |
| 234 | file read.....                   | 2209        |  |
| 235 | file removal.....                | 2210        |  |
| 236 | file searching                   |             |  |
| 237 | grep.....                        | 2703        |  |
| 238 | file time values.....            | 2210        |  |
| 239 | file tree commands               |             |  |
| 240 | diff.....                        | 2529        |  |
| 241 | find.....                        | 2661        |  |
| 242 | ls.....                          | 2779        |  |
| 243 | mkdir.....                       | 2845        |  |
| 244 | rmdir.....                       | 3040        |  |
| 245 | file write.....                  | 2209        |  |
| 246 | filters                          |             |  |
| 247 | asa.....                         | 2355        |  |
| 248 | awk.....                         | 2367        |  |
| 249 | compress.....                    | 2477        |  |
| 250 | dd.....                          | 2515        |  |
| 251 | expand.....                      | 2636        |  |
|     | fold.....                        | 2669        |  |
|     | head.....                        | 2711        |  |
|     | iconv.....                       | 2714        |  |
|     | more.....                        | 2851        |  |
|     | nl.....                          | 2876        |  |
|     | paste.....                       | 2896        |  |
|     | pax.....                         | 2910        |  |
|     | pr.....                          | 2945        |  |
|     | read.....                        | 3025        |  |
|     | sed.....                         | 3051        |  |
|     | tail.....                        | 3107        |  |
|     | tee.....                         | 3115        |  |
|     | tr.....                          | 3136        |  |
|     | uncompress.....                  | 3163        |  |
|     | unexpand.....                    | 3166        |  |
|     | zcat.....                        | 3294        |  |
|     | find.....                        | <b>2661</b> |  |
|     | fold.....                        | <b>2669</b> |  |
|     | for loop.....                    | 2261        |  |
|     | fort77.....                      | <b>2672</b> |  |
|     | external symbols.....            | 2675        |  |
|     | standard libraries.....          | 2674        |  |
|     | FR.....                          | <b>2213</b> |  |
|     | FSC.....                         | <b>2213</b> |  |
|     | function definition command..... | 2263        |  |
|     | function identifiers.....        | 2412        |  |
|     | fuser.....                       | 2204, 2678  |  |
|     | g-file.....                      | 2522        |  |
|     | gencat.....                      | <b>2681</b> |  |
|     | escape sequences.....            | 2683        |  |
|     | generated file.....              | 2522        |  |
|     | get.....                         | <b>2685</b> |  |
|     | getconf.....                     | <b>2692</b> |  |
|     | getopts.....                     | <b>2698</b> |  |
|     | global storage class.....        | 2416        |  |
|     | GNU make.....                    | 2834        |  |
|     | grep.....                        | <b>2703</b> |  |
|     | grouping commands.....           | 2261        |  |
|     | hash.....                        | <b>2708</b> |  |
|     | head.....                        | <b>2711</b> |  |
|     | here-document.....               | 2252        |  |
|     | history command                  |             |  |
|     | fc.....                          | 2646        |  |
|     | Hold Batch Job Request.....      | 2329        |  |
|     | Hold_Types.....                  | 2323        |  |
|     | HOME.....                        | 2582        |  |
|     | hunk.....                        | 2902        |  |
|     | iconv.....                       | <b>2714</b> |  |
|     | id.....                          | <b>2717</b> |  |
|     | if conditional construct.....    | 2262        |  |
|     | implementation-defined.....      | 2205        |  |

|     |                               |                        |                                      |             |
|-----|-------------------------------|------------------------|--------------------------------------|-------------|
| 252 | inference rule.....           | 2818                   | copy messages.....                   | 2805        |
| 253 | input field separator.....    | 3063                   | declare aliases.....                 | 2804        |
| 254 | input mode.....               | 2546                   | declare alternatives.....            | 2805        |
| 255 | IP6.....                      | <b>2213</b>            | delete aliases.....                  | 2811        |
| 256 | ipcrm.....                    | 2204, 2721             | delete messages.....                 | 2805        |
| 257 | ipcs.....                     | 2204, 2723             | delete messages and display.....     | 2806        |
| 258 | jobs.....                     | <b>2730</b>            | direct messages to mbox.....         | 2808        |
| 259 | Job_Owner.....                | 2323                   | discard header fields.....           | 2805        |
| 260 | join.....                     | <b>2734</b>            | display beginning of messages.....   | 2811        |
| 261 | Join_Path.....                | 2323                   | display current message number.....  | 2812        |
| 262 | Keep_Files.....               | 2323                   | display header summary.....          | 2807        |
| 263 | keyword-value pairs.....      | 2337                   | display list of folders.....         | 2807        |
| 264 | kill.....                     | <b>2739</b>            | display message.....                 | 2809        |
| 265 | legacy.....                   | 2205                   | display message size.....            | 2810        |
| 266 | lex.....                      | <b>2743</b>            | echo a string.....                   | 2806        |
| 267 | actions.....                  | 2749                   | edit message.....                    | 2806, 2811  |
| 268 | definitions.....              | 2745                   | execute commands conditionally.....  | 2808        |
| 269 | escape sequences.....         | 2748                   | exit.....                            | 2806        |
| 270 | regular expressions.....      | 2747                   | follow up specified messages.....    | 2807        |
| 271 | rules.....                    | 2746                   | help.....                            | 2807        |
| 272 | table sizes.....              | 2746                   | hold messages.....                   | 2807        |
| 273 | user subroutines.....         | 2747                   | internal variables.....              | 2801        |
| 274 | lex, translation table.....   | 2753                   | invoke a shell.....                  | 2810        |
| 275 | libraries                     |                        | invoke shell command.....            | 2812        |
| 276 | ar command.....               | 2348                   | list available commands.....         | 2808        |
| 277 | LIMIT.....                    | 2220                   | mail a message.....                  | 2808        |
| 278 | line counting.....            | 3258                   | null command.....                    | 2812        |
| 279 | LINE_MAX.....                 | 2221, 2368, 2558-2559  | pipe message.....                    | 2808        |
| 280 | .....                         | 2566, 2815, 3059, 3175 | process next specified message.....  | 2808        |
| 281 | link.....                     | 2204, 2755             | quit.....                            | 2809        |
| 282 | lists.....                    | 2259                   | read mailx commands from a file..... | 2810        |
| 283 | AND-OR.....                   | 2259                   | receive mode.....                    | 2794        |
| 284 | compound-list.....            | 2259                   | reply to a message.....              | 2809        |
| 285 | ln.....                       | <b>2757</b>            | reply to a message list.....         | 2809        |
| 286 | locale.....                   | <b>2761</b>            | retain header fields.....            | 2810        |
| 287 | localedef.....                | <b>2766</b>            | save messages.....                   | 2810        |
| 288 | Locate Batch Job Request..... | 2330                   | scroll header display.....           | 2812        |
| 289 | locking file.....             | 2454                   | send mode.....                       | 2794        |
| 290 | logger.....                   | <b>2770</b>            | set variables.....                   | 2810        |
| 291 | logname.....                  | <b>2772</b>            | start-up.....                        | 2801        |
| 292 | lp.....                       | <b>2774</b>            | touch messages.....                  | 2811        |
| 293 | LR(1) grammars.....           | 3290                   | undelete messages.....               | 2811        |
| 294 | ls.....                       | <b>2779</b>            | unset variables.....                 | 2811        |
| 295 | m4.....                       | <b>2787</b>            | write messages to a file.....        | 2811        |
| 296 | macro processor.....          | 2787                   | Mail_Points.....                     | 2324        |
| 297 | magic file.....               | 2659                   | Mail_Users.....                      | 2324        |
| 298 | mailx.....                    | <b>2794</b>            | make.....                            | <b>2818</b> |
| 299 | change current directory..... | 2805                   | default rules.....                   | 2828        |
| 300 | change folder.....            | 2806                   | inference rules.....                 | 2825        |
| 301 | command escapes.....          | 2812                   | internal macros.....                 | 2826        |
| 302 | commands.....                 | 2804                   | libraries.....                       | 2826        |

## Index

|     |                                         |             |                                               |                  |
|-----|-----------------------------------------|-------------|-----------------------------------------------|------------------|
| 303 | macros .....                            | 2824        | NAME_MAX.....                                 | 2352, 2936       |
| 304 | makefile execution.....                 | 2822        | newgrp.....                                   | <b>2868</b>      |
| 305 | makefile syntax.....                    | 2821        | NGROUPS_MAX .....                             | 2871             |
| 306 | target rules.....                       | 2822        | nice.....                                     | <b>2872</b>      |
| 307 | make, GNU version .....                 | 2834        | Ninth Edition UNIX.....                       | 2419, 2545, 2954 |
| 308 | MAN .....                               | <b>2213</b> | nl .....                                      | <b>2876</b>      |
| 309 | man.....                                | <b>2838</b> | nm.....                                       | <b>2880</b>      |
| 310 | may.....                                | 2205        | noclobber option.....                         | 2909             |
| 311 | mesg.....                               | <b>2842</b> | nohup.....                                    | <b>2885</b>      |
| 312 | message catalog generation .....        | 2681        | non-printable.....                            | 2558, 3058, 3114 |
| 313 | MF.....                                 | <b>2213</b> | OB .....                                      | <b>2214</b>      |
| 314 | MIL-STD-1753.....                       | 2676        | object files.....                             | 2880             |
| 315 | Minimum_Cpu_Interval.....               | 2321        | od .....                                      | <b>2888</b>      |
| 316 | mkdir.....                              | <b>2845</b> | OF .....                                      | <b>2214</b>      |
| 317 | mkfifo.....                             | <b>2848</b> | OH .....                                      | <b>2214</b>      |
| 318 | ML.....                                 | <b>2214</b> | open file descriptors for reading and writing | 2254             |
| 319 | MLR.....                                | <b>2214</b> | open mode .....                               | 2565             |
| 320 | Modify Batch Job Request .....          | 2330        | option                                        |                  |
| 321 | MON .....                               | <b>2214</b> | ADV .....                                     | 2212             |
| 322 | more .....                              | <b>2851</b> | AIO .....                                     | 2212             |
| 323 | discard and refresh .....               | 2857        | BAR .....                                     | 2212             |
| 324 | display position.....                   | 2860        | BE.....                                       | 2212             |
| 325 | examine new file.....                   | 2859        | CD.....                                       | 2212             |
| 326 | examine next file .....                 | 2859        | CPT.....                                      | 2213             |
| 327 | examine previous file .....             | 2859        | CS.....                                       | 2213             |
| 328 | go to beginning of file.....            | 2857        | FD .....                                      | 2213             |
| 329 | go to end-of-file .....                 | 2857        | FR.....                                       | 2213             |
| 330 | go to tag.....                          | 2859        | FSC .....                                     | 2213             |
| 331 | help.....                               | 2856        | IP6.....                                      | 2213             |
| 332 | invoke editor .....                     | 2859        | MF.....                                       | 2213             |
| 333 | mark position .....                     | 2858        | ML .....                                      | 2214             |
| 334 | quit .....                              | 2860        | MLR.....                                      | 2214             |
| 335 | refresh the screen.....                 | 2857        | MON .....                                     | 2214             |
| 336 | repeat search .....                     | 2858        | MPR.....                                      | 2214             |
| 337 | repeat search in reverse .....          | 2859        | MSG.....                                      | 2214             |
| 338 | return to mark.....                     | 2858        | PIO.....                                      | 2215             |
| 339 | return to previous position .....       | 2858        | PS .....                                      | 2215             |
| 340 | scroll backward one half screenful..... | 2857        | RTS .....                                     | 2215             |
| 341 | scroll backward one line .....          | 2856        | SD .....                                      | 2215             |
| 342 | scroll backward one screenful.....      | 2856        | SEM .....                                     | 2215             |
| 343 | scroll forward one half screenful ..... | 2857        | SHM.....                                      | 2215             |
| 344 | scroll forward one line .....           | 2856        | SIO .....                                     | 2215             |
| 345 | scroll forward one screenful .....      | 2856        | SPI.....                                      | 2216             |
| 346 | search backward for pattern.....        | 2858        | SPN.....                                      | 2216             |
| 347 | search forward for pattern .....        | 2858        | SS .....                                      | 2216             |
| 348 | skip forward one line.....              | 2857        | TCT.....                                      | 2216             |
| 349 | motion command.....                     | 3066        | TEF .....                                     | 2217             |
| 350 | Move Batch Job Request .....            | 2331        | THR.....                                      | 2216             |
| 351 | MPR.....                                | <b>2214</b> | TMO.....                                      | 2216             |
| 352 | MSG.....                                | <b>2214</b> | TMR.....                                      | 2216             |
| 353 | mv.....                                 | <b>2863</b> | TPI .....                                     | 2216             |

|     |                                              |                        |                                          |             |
|-----|----------------------------------------------|------------------------|------------------------------------------|-------------|
| 354 | TPP .....                                    | 2217                   | extended header file times .....         | 2926        |
| 355 | TPS.....                                     | 2217                   | extended header keyword precedence ..... | 2925        |
| 356 | TRC .....                                    | 2217                   | list mode format specifications .....    | 2918        |
| 357 | TRI.....                                     | 2217                   | ustar format .....                       | 2926        |
| 358 | TRL.....                                     | 2217                   | ustar interchange format .....           | 2926        |
| 359 | TSA.....                                     | 2217                   | PIO .....                                | <b>2215</b> |
| 360 | TSF.....                                     | 2218                   | pipelines .....                          | 2258        |
| 361 | TSH.....                                     | 2218                   | portable character set .....             | 2824        |
| 362 | TSP.....                                     | 2218                   | positional parameters.....               | 2241        |
| 363 | TSS.....                                     | 2218                   | POSIX2_BC_BASE_MAX.....                  | 2220-2221   |
| 364 | TYM.....                                     | 2218                   | POSIX2_BC_DIM_MAX.....                   | 2220-2221   |
| 365 | UP .....                                     | 2218                   | POSIX2_BC_SCALE_MAX.....                 | 2220-2221   |
| 366 | XSR.....                                     | 2219                   | POSIX2_BC_STRING_MAX.....                | 2220-2221   |
| 367 | OR lists.....                                | 2260                   | POSIX2_COLL_WEIGHTS_MAX.....             | 2220-2221   |
| 368 | ordinary identifiers .....                   | 2412                   | POSIX2_EXPR_NEST_MAX .....               | 2220-2221   |
| 369 | Output_Path.....                             | 2324                   | POSIX2_LINE_MAX .....                    | 2220, 2222  |
| 370 | paginators                                   |                        | POSIX2_RE_DUP_MAX.....                   | 2220, 2222  |
| 371 | more .....                                   | 2851                   | POSIX2_SYMLINKS .....                    | 2222        |
| 372 | parameter expansion.....                     | 2245                   | POSIX2_VERSION .....                     | 2220        |
| 373 | parameters and variables.....                | 2241                   | pr .....                                 | <b>2945</b> |
| 374 | paste .....                                  | <b>2896</b>            | print-related commands                   |             |
| 375 | patch.....                                   | <b>2900</b>            | fold .....                               | 2669        |
| 376 | file name determination.....                 | 2903                   | lp .....                                 | 2774        |
| 377 | patch application.....                       | 2903                   | pr.....                                  | 2945        |
| 378 | patchfile format .....                       | 2902                   | printf.....                              | <b>2950</b> |
| 379 | path name expansion.....                     | 2249                   | Priority .....                           | 2325        |
| 380 | path name manipulation                       |                        | privileges .....                         | 2843, 2874  |
| 381 | basename .....                               | 2401                   | process attributes .....                 | 2208        |
| 382 | dirname .....                                | 2536                   | process group ID .....                   | 2208        |
| 383 | pathchk.....                                 | 2906                   | process ID.....                          | 2208        |
| 384 | path name resolution.....                    | 2211                   | process status report.....               | 2960        |
| 385 | pathchk.....                                 | <b>2906</b>            | prs .....                                | <b>2955</b> |
| 386 | PATH_MAX.....                                | 2221, 2942, 3035       | PS .....                                 | <b>2215</b> |
| 387 | pattern matching.....                        | 2661, 2919, 3179, 3195 | ps.....                                  | <b>2960</b> |
| 388 | definition.....                              | 2274                   | public locale.....                       | 2761        |
| 389 | in case statements .....                     | 2262                   | pwd.....                                 | <b>2967</b> |
| 390 | in shell variables .....                     | 2246                   | qalter .....                             | <b>2969</b> |
| 391 | pattern matching notation.....               | 2274, 2666, 2935       | qdel .....                               | <b>2978</b> |
| 392 | pattern scanning and processing language     |                        | qhold .....                              | <b>2981</b> |
| 393 | at .....                                     | 2367                   | qmove .....                              | <b>2984</b> |
| 394 | patterns matching a single character .....   | 2274                   | qmsg.....                                | <b>2987</b> |
| 395 | patterns matching multiple characters.....   | 2274                   | qrun.....                                | <b>2990</b> |
| 396 | patterns used for file name expansion.....   | 2275                   | qrls .....                               | <b>2992</b> |
| 397 | pax .....                                    | <b>2910</b>            | qselect .....                            | <b>2995</b> |
| 398 | archive character set encoding/decoding..... | 2941                   | qsig .....                               | <b>3004</b> |
| 399 | cpio file data .....                         | 2932                   | qstat .....                              | <b>3007</b> |
| 400 | cpio file name .....                         | 2932                   | qsub .....                               | <b>3012</b> |
| 401 | cpio header .....                            | 2930                   | Queue Batch Job Request.....             | 2331        |
| 402 | cpio interchange format.....                 | 2929                   | quote removal .....                      | 2250        |
| 403 | cpio special entries.....                    | 2932                   | quoting.....                             | 2236        |
| 404 | extended header .....                        | 2923                   | read.....                                | <b>3025</b> |

## Index

|     |                                    |                                                |                                                    |                  |
|-----|------------------------------------|------------------------------------------------|----------------------------------------------------|------------------|
| 405 | readonly.....                      | <b>2293</b>                                    | SEM .....                                          | <b>2215</b>      |
| 406 | readonly special built-in .....    | 2293                                           | sequential lists.....                              | 2260             |
| 407 | real group ID .....                | 2208                                           | Server Shutdown Request .....                      | 2334             |
| 408 | real user ID.....                  | 2208                                           | Server Status Request .....                        | 2334             |
| 409 | redirecting input.....             | 2252                                           | session membership.....                            | 2208             |
| 410 | redirecting output .....           | 2252                                           | set.....                                           | <b>2297</b>      |
| 411 | redirection .....                  | 2251                                           | set special built-in.....                          | 2297             |
| 412 | regular expressions.....           | 2374-2375, 2484, 2547                          | set-group-ID.....                                  | 2208, 2486       |
| 413 | .....                              | 2571, 2601, 2641, 2664, 2703, 2747, 2855       | set-user-ID.....                                   | 2208, 2455, 2486 |
| 414 | .....                              | 2865, 2876, 2916, 3033, 3053, 3216, 3219, 3273 | set-user-ID scripts .....                          | 3074             |
| 415 | related to shell patterns.....     | 2274                                           | sh.....                                            | <b>3060</b>      |
| 416 | relational database operator ..... | 2734                                           | command history list.....                          | 3064             |
| 417 | Release Batch Job Request .....    | 2333                                           | command line editing .....                         | 3065             |
| 418 | remove directories.....            | 3040                                           | vi line editing command mode .....                 | 3066             |
| 419 | remove files .....                 | 3032                                           | vi line editing insert mode.....                   | 3065             |
| 420 | renice.....                        | <b>3028</b>                                    | vi-mode command line editing.....                  | 3065             |
| 421 | requested batch services .....     | 2328                                           | shall .....                                        | 2205             |
| 422 | Rerun Batch Job Request.....       | 2333                                           | shell command language.....                        | 2235             |
| 423 | Rerunable .....                    | 2325                                           | alias substitution .....                           | 2239             |
| 424 | reserved words .....               | 2240                                           | appending redirected output.....                   | 2252             |
| 425 | Resource_List .....                | 2325                                           | arithmetic expansion .....                         | 2248             |
| 426 | return.....                        | <b>2295</b>                                    | command substitution.....                          | 2247             |
| 427 | return special built-in .....      | 2295                                           | compound commands .....                            | 2261             |
| 428 | RE_DUP_MAX.....                    | 2221                                           | consequences of shell errors .....                 | 2255             |
| 429 | rm.....                            | <b>3032</b>                                    | double-quotes .....                                | 2236             |
| 430 | rmdel.....                         | <b>3037</b>                                    | duplicating an input file descriptor.....          | 2253             |
| 431 | rmdir .....                        | <b>3040</b>                                    | duplicating an output file descriptor.....         | 2253             |
| 432 | root directory.....                | 2208                                           | escape character (backslash).....                  | 2236             |
| 433 | RTS .....                          | <b>2215</b>                                    | exit status and errors .....                       | 2255             |
| 434 | sact.....                          | <b>3043</b>                                    | exit status for commands .....                     | 2255             |
| 435 | saved set-group-ID .....           | 2208                                           | field splitting.....                               | 2249             |
| 436 | saved set-user-ID.....             | 2208                                           | function definition command.....                   | 2263             |
| 437 | sccs.....                          | <b>3046</b>                                    | grammar.....                                       | 2266             |
| 438 | SCCS commands                      |                                                | here-document.....                                 | 2252             |
| 439 | admin.....                         | 2340                                           | introduction.....                                  | 2235             |
| 440 | delta.....                         | 2522                                           | lists .....                                        | 2259             |
| 441 | get .....                          | 2685                                           | open file descriptors for reading and writing..... | 2254             |
| 442 | prs.....                           | 2955                                           | parameter expansion.....                           | 2245             |
| 443 | rmdel.....                         | 3037                                           | parameters and variables .....                     | 2241             |
| 444 | sact.....                          | 3043                                           | path name expansion .....                          | 2249             |
| 445 | sccs .....                         | 3046                                           | pattern matching notation.....                     | 2274             |
| 446 | unget .....                        | 3169                                           | patterns matching a single character .....         | 2274             |
| 447 | val .....                          | 3197                                           | patterns matching multiple characters.....         | 2274             |
| 448 | what .....                         | 3261                                           | patterns used for file name expansion.....         | 2275             |
| 449 | SD.....                            | <b>2215</b>                                    | pipelines .....                                    | 2258             |
| 450 | search pattern.....                | 2496                                           | positional parameters.....                         | 2241             |
| 451 | sed.....                           | <b>3051</b>                                    | quote removal.....                                 | 2250             |
| 452 | addresses.....                     | 3053                                           | quoting .....                                      | 2236             |
| 453 | editing commands .....             | 3053                                           | redirecting input.....                             | 2252             |
| 454 | regular expressions .....          | 3053                                           | redirecting output .....                           | 2252             |
| 455 | Select Batch Jobs Request .....    | 2334                                           |                                                    |                  |

|     |                                        |                        |                                            |                  |
|-----|----------------------------------------|------------------------|--------------------------------------------|------------------|
| 456 | redirection.....                       | 2251                   | exit.....                                  | 2289             |
| 457 | reserved words.....                    | 2240                   | export.....                                | 2291             |
| 458 | shell commands.....                    | 2256                   | readonly.....                              | 2293             |
| 459 | shell execution environment.....       | 2273                   | return.....                                | 2295             |
| 460 | shell grammar lexical conventions..... | 2266                   | set.....                                   | 2297             |
| 461 | shell grammar rules.....               | 2266                   | shift.....                                 | 2303             |
| 462 | shell variables.....                   | 2242                   | times.....                                 | 2305             |
| 463 | signals and error handling.....        | 2272                   | trap.....                                  | 2307             |
| 464 | simple commands.....                   | 2256                   | unset.....                                 | 2310             |
| 465 | single-quotes.....                     | 2236                   | special parameters.....                    | 2241             |
| 466 | special built-in utilities.....        | 2276                   | special targets.....                       | 2823             |
| 467 | special parameters.....                | 2241                   | SPI.....                                   | <b>2216</b>      |
| 468 | tilde expansion.....                   | 2244                   | split.....                                 | <b>3086</b>      |
| 469 | token recognition.....                 | 2238                   | split files                                |                  |
| 470 | word expansions.....                   | 2244                   | csplit.....                                | 2492             |
| 471 | shell commands.....                    | 2256                   | split.....                                 | 3086             |
| 472 | shell execution environment.....       | 2273, 2346             | SPN.....                                   | <b>2216</b>      |
| 473 | .....                                  | 3027, 3156             | spoofing.....                              | 2294             |
| 474 | shell grammar.....                     | 2266                   | SS.....                                    | <b>2216</b>      |
| 475 | shell grammar lexical conventions..... | 2266                   | standard error.....                        | 2251             |
| 476 | shell grammar rules.....               | 2266                   | standard input.....                        | 2251             |
| 477 | shell introduction.....                | 2235                   | standard output.....                       | 2251             |
| 478 | shell variables.....                   | 2242                   | strings.....                               | <b>3089</b>      |
| 479 | Shell_Path_List.....                   | 2325                   | strip.....                                 | <b>3092</b>      |
| 480 | shift.....                             | <b>2303</b>            | stty.....                                  | <b>3094</b>      |
| 481 | shift special built-in.....            | 2303                   | combination modes.....                     | 3099             |
| 482 | SHM.....                               | <b>2215</b>            | control modes.....                         | 3094             |
| 483 | should.....                            | 2205                   | input modes.....                           | 3095             |
| 484 | SIGCONT.....                           | 2568                   | local modes.....                           | 3097             |
| 485 | SIGHUP.....                            | 2547, 2568, 3200, 3241 | output modes.....                          | 3096             |
| 486 | SIGINT.....                            | 2547, 2567, 3253       | special control character assignments..... | 3098             |
| 487 | Signal Batch Job Request.....          | 2335                   | st_gid.....                                | 2353             |
| 488 | signal processes.....                  | 2739                   | st_mode.....                               | 2353             |
| 489 | signals and error handling.....        | 2272                   | st_mtime.....                              | 2353             |
| 490 | SIGQUIT.....                           | 2547                   | st_size.....                               | 2353             |
| 491 | SIGTERM.....                           | 2568                   | st_uid.....                                | 2353             |
| 492 | simple commands.....                   | 2256                   | superuser.....                             | 2650, 2785, 2934 |
| 493 | single-quotes.....                     | 2236                   | supplementary group IDs.....               | 2208             |
| 494 | SIO.....                               | <b>2215</b>            | system configuration values.....           | 2692             |
| 495 | sleep.....                             | <b>3077</b>            | system name.....                           | 3160             |
| 496 | SLR(1) grammars.....                   | 3290                   | tabs.....                                  | <b>3103</b>      |
| 497 | sort.....                              | <b>3080</b>            | tag file creation.....                     | 2496             |
| 498 | special built-in.....                  | 2475, 2874, 2887       | tail.....                                  | <b>3107</b>      |
| 499 | .....                                  | 2968, 3073, 3128, 3143 | talk.....                                  | <b>3111</b>      |
| 500 | special built-in utilities.....        | 2276                   | tar format.....                            | 2926             |
| 501 | break.....                             | 2277, 2281             | target queue.....                          | 2332             |
| 502 | characteristics.....                   | 2276                   | target rule.....                           | 2818             |
| 503 | colon.....                             | 2279                   | TCT.....                                   | <b>2216</b>      |
| 504 | dot.....                               | 2283                   | tee.....                                   | <b>3115</b>      |
| 505 | eval.....                              | 2285                   | TEF.....                                   | <b>2217</b>      |
| 506 | exec.....                              | 2287                   |                                            |                  |

## Index

|     |                              |             |
|-----|------------------------------|-------------|
| 507 | terminal characteristics     |             |
| 508 | stty.....                    | 3094        |
| 509 | tabs.....                    | 3103        |
| 510 | tput.....                    | 3133        |
| 511 | tty.....                     | 3147        |
| 512 | terminate processes.....     | 2739        |
| 513 | terminology.....             | <b>2205</b> |
| 514 | test.....                    | <b>3118</b> |
| 515 | THR.....                     | <b>2216</b> |
| 516 | tilde expansion.....         | 2244        |
| 517 | time.....                    | <b>3125</b> |
| 518 | times.....                   | <b>2305</b> |
| 519 | times special built-in.....  | 2305        |
| 520 | TMO.....                     | <b>2216</b> |
| 521 | TMPDIR.....                  | 2914        |
| 522 | TMR.....                     | <b>2216</b> |
| 523 | token recognition.....       | 2238        |
| 524 | touch.....                   | <b>3129</b> |
| 525 | TPI.....                     | <b>2216</b> |
| 526 | TPP.....                     | <b>2217</b> |
| 527 | TPS.....                     | <b>2217</b> |
| 528 | tput.....                    | <b>3133</b> |
| 529 | tr.....                      | <b>3136</b> |
| 530 | Track Batch Job Request..... | 2335        |
| 531 | trap.....                    | <b>2307</b> |
| 532 | trap special built-in.....   | 2307        |
| 533 | TRC.....                     | <b>2217</b> |
| 534 | TRI.....                     | <b>2217</b> |
| 535 | TRL.....                     | <b>2217</b> |
| 536 | trojan horse.....            | 2785        |
| 537 | true.....                    | <b>3142</b> |
| 538 | TSA.....                     | <b>2217</b> |
| 539 | TSF.....                     | <b>2218</b> |
| 540 | TSH.....                     | <b>2218</b> |
| 541 | tsort.....                   | <b>3144</b> |
| 542 | TSP.....                     | <b>2218</b> |
| 543 | TSS.....                     | <b>2218</b> |
| 544 | tty.....                     | <b>3147</b> |
| 545 | TYM.....                     | <b>2218</b> |
| 546 | type.....                    | <b>3149</b> |
| 547 | ulimit.....                  | <b>3151</b> |
| 548 | umask.....                   | <b>3154</b> |
| 549 | UN.....                      | <b>2218</b> |
| 550 | unalias.....                 | <b>3158</b> |
| 551 | uname.....                   | <b>3160</b> |
| 552 | unary primaries.....         | 3119        |
| 553 | uncompress.....              | <b>3163</b> |
| 554 | undefined.....               | 2205        |
| 555 | unexpand.....                | <b>3166</b> |
| 556 | unset.....                   | <b>3169</b> |
| 557 | uniq.....                    | <b>3172</b> |
|     | unlink.....                  | 2204, 3176  |
|     | unset.....                   | <b>2310</b> |
|     | unset special built-in.....  | 2310        |
|     | unspecified.....             | 2206        |
|     | until loop.....              | 2263        |
|     | UP.....                      | <b>2218</b> |
|     | user identity                |             |
|     | id.....                      | 2717        |
|     | logname.....                 | 2772        |
|     | newgrp.....                  | 2868        |
|     | who.....                     | 3264        |
|     | User_List.....               | 2326        |
|     | ustar format.....            | 2926        |
|     | utility option parsing.....  | 2698        |
|     | uucp.....                    | <b>3178</b> |
|     | uudecode.....                | <b>3182</b> |
|     | uuencode.....                | <b>3185</b> |
|     | uustat.....                  | <b>3190</b> |
|     | uux.....                     | <b>3193</b> |
|     | val.....                     | <b>3197</b> |
|     | Variable_List.....           | 2326        |
|     | vi.....                      | <b>3200</b> |
|     | <ESC>.....                   | 3240        |
|     | append.....                  | 3221        |
|     | change.....                  | 3221        |
|     | change to end-of-line.....   | 3222        |
|     | clear and redisplay.....     | 3208        |
|     | command descriptions.....    | 3201        |
|     | control-D.....               | 3237        |
|     | control-H.....               | 3237        |
|     | control-T.....               | 3238        |
|     | control-U.....               | 3239        |
|     | control-V.....               | 3239        |
|     | current and line above.....  | 3214        |
|     | delete.....                  | 3222        |
|     | delete character.....        | 3232-3233   |
|     | delete to end-of-line.....   | 3223        |
|     | display information.....     | 3207        |
|     | edit the alternate file..... | 3209        |
|     | enter ex mode.....           | 3229        |
|     | execute.....                 | 3220        |
|     | execute an ex command.....   | 3218        |
|     | exit.....                    | 3235        |
|     | find character.....          | 3224        |
|     | find regular expression..... | 3216        |
|     | Initialization.....          | 3201        |
|     | input mode commands.....     | 3235        |
|     | insert.....                  | 3225        |
|     | insert empty line.....       | 3227        |
|     | join.....                    | 3225        |
|     | mark position.....           | 3226        |

|     |                                       |                  |                                        |             |
|-----|---------------------------------------|------------------|----------------------------------------|-------------|
| 558 | move back .....                       | 3215, 3221       | warning                                |             |
| 559 | move cursor .....                     | 3207, 3210, 3230 | MAN .....                              | 2213        |
| 560 | move down.....                        | 3207             | OB .....                               | 2214        |
| 561 | move forward .....                    | 3215             | OF .....                               | 2214        |
| 562 | move to bigword.....                  | 3223, 3232       | UN .....                               | 2218        |
| 563 | move to bottom of screen.....         | 3226             | wc.....                                | <b>3258</b> |
| 564 | move to first character in line ..... | 3218             | what.....                              | <b>3261</b> |
| 565 | move to first non-<blank>.....        | 3214             | while loop .....                       | 2263        |
| 566 | move to line .....                    | 3224             | who.....                               | <b>3264</b> |
| 567 | move to matching character .....      | 3211             | word counting.....                     | 3258        |
| 568 | move to middle of screen .....        | 3226             | word expansions .....                  | 2244        |
| 569 | move to next section.....             | 3213             | write .....                            | <b>3268</b> |
| 570 | move to specific column.....          | 3215             | xargs .....                            | <b>3271</b> |
| 571 | move to top of screen .....           | 3225             | XSI.....                               | <b>2218</b> |
| 572 | move to word .....                    | 3223, 3231       | XSR .....                              | <b>2219</b> |
| 573 | move up .....                         | 3208             | yacc.....                              | <b>3277</b> |
| 574 | newline .....                         | 3238             | algorithms.....                        | 3288        |
| 575 | nul .....                             | 3236, 3240       | code file .....                        | 3279        |
| 576 | page backwards.....                   | 3205             | completing the program .....           | 3287        |
| 577 | page forward.....                     | 3206             | conflicts .....                        | 3285        |
| 578 | put from buffer .....                 | 3227-3228        | debugging the parser.....              | 3288        |
| 579 | redraw screen.....                    | 3208             | declarations section .....             | 3280        |
| 580 | redraw window .....                   | 3234             | description file .....                 | 3279        |
| 581 | regular expression.....               | 3219             | error handling .....                   | 3286        |
| 582 | reindent .....                        | 3220             | grammar rules.....                     | 3281        |
| 583 | repeat .....                          | 3216             | header file .....                      | 3279        |
| 584 | repeat find .....                     | 3219, 3227       | input grammar.....                     | 3283        |
| 585 | repeat substitution.....              | 3212             | input language.....                    | 3279        |
| 586 | replace character.....                | 3229             | interface to the lexical analyzer..... | 3287        |
| 587 | replace text with command .....       | 3210             | lexical structure of the grammar.....  | 3280        |
| 588 | return to previous context.....       | 3212             | library.....                           | 3288        |
| 589 | return to previous section .....      | 3213             | limits .....                           | 3288        |
| 590 | reverse case.....                     | 3220             | programs section.....                  | 3283        |
| 591 | reverse find character.....           | 3216             | zcat.....                              | <b>3294</b> |
| 592 | scroll backward.....                  | 3208             |                                        |             |
| 593 | scroll backward by line .....         | 3209             |                                        |             |
| 594 | scroll forward.....                   | 3206             |                                        |             |
| 595 | scroll forward by line .....          | 3206             |                                        |             |
| 596 | search for tagstring .....            | 3209             |                                        |             |
| 597 | shift left.....                       | 3219             |                                        |             |
| 598 | shift right.....                      | 3219             |                                        |             |
| 599 | substitute character.....             | 3230             |                                        |             |
| 600 | substitute lines .....                | 3230             |                                        |             |
| 601 | terminate command or input mode.....  | 3209             |                                        |             |
| 602 | undo .....                            | 3231             |                                        |             |
| 603 | undo current line .....               | 3231             |                                        |             |
| 604 | yank.....                             | 3233             |                                        |             |
| 605 | yank current line.....                | 3233             |                                        |             |
| 606 | visual mode .....                     | 2565             |                                        |             |
| 607 | wait.....                             | <b>3254</b>      |                                        |             |



1 / *Rationale (Informative)*

2 **Part A:**

3 **Base Definitions**

4 *The Open Group*



# Rationale for Base Definitions

5

## 6 A.1 Introduction

### 7 A.1.1 Scope

8 IEEE Std. 1003.1-200x is one of a family of standards known as POSIX. The family of standards  
9 extends to many topics; IEEE Std. 1003.1-200x is known as POSIX.1 and consists of both  
10 operating system interfaces and shell and utilities.

#### 11 Scope of IEEE Std. 1003.1-200x

12 The (paraphrased) goals of this development were to produce a single common revision to the  
13 overlapping POSIX.1 and POSIX.2 standards, and the Single UNIX Specification, Version 2. As  
14 such, the scope of the revision includes the scopes of the original documents merged.

15 Since the revision includes merging the Base volumes of the Single UNIX Specification, many  
16 features that were previously not *adopted* into earlier revisions of POSIX.1 and POSIX.2 are now  
17 included in IEEE Std. 1003.1-200x. In most cases, these additions are part of the XSI extension; in  
18 other cases the standard developers decided that now was the time to migrate these to the base  
19 standard.

20 The Single UNIX Specification programming environment provides a broad-based functional set  
21 of interfaces to support the porting of existing UNIX applications and the development of new  
22 applications. The environment also supports a rich set of tools for application development.

23 The majority of the obsolescent material from the existing POSIX.1 and POSIX.2 standards, and  
24 material marked LEGACY from The Open Group's Base specifications, has been removed in this  
25 revision.

26 The following IEEE Standards have been added to the base documents in this revision:

- 27 • IEEE Std. 1003.1d-1999
- 28 • IEEE Std. 1003.1j-2000
- 29 • IEEE Std. 1003.1q-2000
- 30 • IEEE P1003.1a draft standard
- 31 • IEEE Std. 1003.2d-1994
- 32 • IEEE P1003.2b draft standard
- 33 • Selected parts of IEEE Std. 1003.1g-2000

34 Only selected parts of IEEE Std. 1003.1g-2000 were included. This was because there is much  
35 duplication between the XNS, Issue 5.2 specification (another base document) and the material  
36 from IEEE Std. 1003.1g-2000, the former document being aligned with the latest networking  
37 specifications for IPv6. Only the following sections of IEEE Std. 1003.1g-2000 were considered  
38 for inclusion:

- 39 • General terms related to sockets (clause 2.2.2)

- 40 • Socket concepts (clauses 5.1 through 5.3 inclusive)
- 41 • The *pselect()* function (clauses 6.2.2.1 and 6.2.3)
- 42 • The *isfdtype()* function (clause 5.4.8)
- 43 • The `<sys/select.h>` header (clause 6.2)

44 The following were requirements on IEEE Std. 1003.1-200x:

- 45 • Backward-compatibility

46 It was agreed that there should be no breakage of functionality in the existing base  
 47 documents. This requirement was tempered by changes introduced due to interpretations  
 48 and corrigenda on the base documents, and any changes introduced in the  
 49 ISO/IEC 9899:1999 standard (C Language).

- 50 • Architecture and n-bit neutral

51 The common standard should not make any implicit assumptions about the system  
 52 architecture or size of data types; for example, previously some 32-bit implicit assumptions  
 53 had crept into the standards.

- 54 • Extensibility

55 It should be possible to extend the common standard without breaking backward-  
 56 compatibility. For example, the name space should be reserved and structured to avoid  
 57 duplication of names between the standard and extensions to it.

#### 58 **POSIX.1 and the ISO C standard**

59 Previous revisions of POSIX.1 built upon the ISO C standard by reference only. This revision  
 60 takes a different approach.

61 The standard developers believed it essential for a programmer to have a single complete  
 62 reference place, but recognized that deference to the formal standard had to be addressed for the  
 63 the duplicate interface definitions between the ISO C standard and the Single UNIX  
 64 Specification.

65 It was agreed that where an interface has a version in the ISO C standard, the DESCRIPTION  
 66 section should describe the relationship to the ISO C standard and markings should be added as  
 67 appropriate to show where the ISO C standard has been extended in the text.

68 The following block of text was added to each reference page affected:

69 *The functionality described on this reference page is aligned with the ISO C standard. Any conflict*  
 70 *between the requirements described here and the ISO C standard is unintentional. This volume of*  
 71 *IEEE Std. 1003.1-200x defers to the ISO C standard.*

72 and each page was parsed for additions beyond the ISO C standard (that is, including both  
 73 POSIX and UNIX extensions), and these extensions are marked as CX extensions (for C  
 74 Extensions).

**75 A.1.2 Conformance**

76 See Section A.2 (on page 3317).

**77 A.1.3 Normative References**

78 There is no additional rationale for this section.

**79 A.1.4 Terminology**

80 The meanings specified in IEEE Std. 1003.1-200x for the words *shall*, *should*, and *may* are  
81 mandated by ISO/IEC directives.

82 In the Rationale (Informative) volume of IEEE Std. 1003.1-200x, the words *shall*, *should*, and *may*  
83 are sometimes used to illustrate similar usages in IEEE Std. 1003.1-200x. However, the rationale  
84 itself does not specify anything regarding implementations or applications.

**85 conformance document**

86 As a practical matter, the conformance document is effectively part of the system  
87 documentation. Conformance documents are distinguished by IEEE Std. 1003.1-200x so that  
88 they can be referred to distinctly.

**89 implementation-defined**

90 This definition is analogous to that of the ISO C standard and, together with *undefined* and  
91 *unspecified*, provides a range of specification of freedom allowed to the interface  
92 implementor.

**93 may**

94 The use of *may* has been limited as much as possible, due both to confusion stemming from  
95 its ordinary English meaning and to objections regarding the desirability of having as few  
96 options as possible and those as clearly specified as possible.

97 The usage of *can* and *may* were selected to contrast optional application behavior (can)  
98 against optional implementation behavior (may).

**99 shall**

100 Declarative sentences are sometimes used in IEEE Std. 1003.1-200x as if they included the  
101 word *shall*, and facilities thus specified are no less required. For example, the two  
102 statements:

- 103 1. The *foo()* function shall return zero.
- 104 2. The *foo()* function returns zero.

105 are meant to be exactly equivalent.

**106 should**

107 In IEEE Std. 1003.1-200x, the word *should* does not usually apply to the implementation, but  
108 rather to the application. Thus, the important words regarding implementations are *shall*,  
109 which indicates requirements, and *may*, which indicates options.

**110 obsolescent**

111 The term *obsolescent* means “do not use this feature in new applications”. The obsolescence  
112 concept is not an ideal solution, but was used as a method of increasing consensus: many  
113 more objections would be heard from the user community if some of these historical  
114 features were suddenly withdrawn without the grace period obsolescence implies. The  
115 phrase “may be considered for withdrawal in future revisions” implies that the result of  
116 that consideration might in fact keep those features indefinitely if the predominance of  
117 applications do not migrate away from them quickly.

118 **legacy**

119 The term *legacy* was added for compatibility with the Single UNIX Specification. It means  
120 “this feature is historic and optional; do not use this feature in new applications. There are  
121 alternate interfaces that are more suitable.”. It is used exclusively for XSI extensions, and  
122 includes facilities that were mandatory in previous versions of the base document but are  
123 optional in this revision. This is a way to “sunset” the usage of certain functions.  
124 Application writers should not rely on the existence of these facilities in new applications,  
125 but should follow the migration path detailed in the APPLICATION USAGE sections of the  
126 relevant pages.

127 The terms *legacy* and *obsolescent* are different: a feature marked LEGACY is not  
128 recommended for new work and need not be present on an implementation (if the XSI  
129 Legacy Option Group is not supported). A feature noted as obsolescent is supported by all  
130 implementations, but may be removed in a future revision; new applications should not use  
131 these features.

132 **system documentation**

133 The system documentation should normally describe the whole of the implementation,  
134 including any extensions provided by the implementation. Such documents normally  
135 contain information at least as detailed as the specifications in IEEE Std. 1003.1-200x. Few  
136 requirements are made on the system documentation, but the term is needed to avoid a  
137 dangling pointer where the conformance document is permitted to point to the system  
138 documentation.

139 **undefined**

140 See *implementation-defined*.

141 **unspecified**

142 See *implementation-defined*.

143 The definitions for *unspecified* and *undefined* appear nearly identical at first examination, but  
144 are not. The term *unspecified* means that a conforming program may deal with the  
145 unspecified behavior, and it should not care what the outcome is. The term *undefined* says  
146 that a conforming program should not do it because no definition is provided for what it  
147 does (and implicitly it would care what the outcome was if it tried it). It is important to  
148 remember, however, that if the syntax permits the statement at all, it must have some  
149 outcome in a real implementation.

150 Thus, the terms *undefined* and *unspecified* apply to the way the application should think  
151 about the feature. In terms of the implementation, it is always “defined”—there is always  
152 some result, even if it is an error. The implementation is free to choose the behavior it  
153 prefers.

154 This also implies that an implementation, or another standard, could specify or define the  
155 result in a useful fashion. The terms apply to IEEE Std. 1003.1-200x specifically.

156 The term *implementation-defined* implies requirements for documentation that are not  
157 required for *undefined* (or *unspecified*). Where there is no need for a conforming program to  
158 know the definition, the term *undefined* is used, even though *implementation-defined* could  
159 also have been used in this context. There could be a fourth term, specifying “this standard  
160 does not say what this does; it is acceptable to define it in an implementation, but it does not  
161 need to be documented”, and *undefined* would then be used very rarely for the few things  
162 for which any definition is not useful.

163 In many places IEEE Std. 1003.1-200x is silent about the behavior of some possible construct.  
164 For example, a variable may be defined for a specified range of values and behaviors are  
165 described for those values; nothing is said about what happens if the variable has any other

166 value. That kind of silence can imply an error in the standard, but it may also imply that the  
 167 standard was intentionally silent and that any behavior is permitted. There is a natural  
 168 tendency to infer that if the standard is silent, a behavior is prohibited. That is not the intent.  
 169 Silence is intended to be equivalent to the term *unspecified*.

170 The term *application* is not defined in IEEE Std. 1003.1-200x; it is assumed to be a part of  
 171 general computer science terminology.

## 172 A.1.5 Portability

173 To aid the identification of options within IEEE Std. 1003.1-200x, a notation consisting of margin  
 174 codes and shading is used. This is based on the notation used in previous revisions of The Open  
 175 Group's Base specifications.

176 The benefits of this approach is a reduction in the number of *if* statements within the running  
 177 text, that makes the text easier to read, and also an identification to the programmer that they  
 178 need to ensure that their target platforms support the underlying options. For example, if  
 179 functionality is marked with THR in the margin, it will be available on all systems supporting  
 180 the Threads option, but may not be available on some.

### 181 A.1.5.1 Codes

182 The set of code includes codes for options defined in clause 2.1.6, Options, and the following  
 183 additional codes for other purposes:

184 CX This margin code is used to denote extensions beyond the ISO C standard, and is used  
 185 in interfaces that are duplicated between IEEE Std. 1003.1-200x and the ISO C standard.

186 MAN This margin code was used during the development of the drafts and should not be  
 187 present in the final published standard.

188 OB This margin code is used to denote obsolescent behavior and thus flag a possible future  
 189 application portability warning.

190 OH The Single UNIX Specification has historically tried to reduce the number of headers an  
 191 application has had to include when using a particular interface. Sometimes this was  
 192 fewer than the base standard, and hence a notation is used to flag which headers are  
 193 optional if you are using a system supporting the XSI extension.

194 PI This is another code used in the XSI extension only. It is used to denote a possible  
 195 application portability warning related to behavior of an interface which may not be  
 196 consistent between all conformant systems.

197 UN This is another code used in the XSI extension only. It is used to denote a possible  
 198 application portability warning related to possibly unsupported functionality.

199 XSI This code is used to denote interfaces and facilities within interfaces only required on  
 200 systems supporting the XSI extension. This is introduced to support the Single UNIX  
 201 Specification.

202 XSR This code is used to denote interfaces and facilities within interfaces only required on  
 203 systems supporting STREAMS. This is introduced to support the Single UNIX  
 204 Specification, although it is defined in a way so that it can standalone from the XSI  
 205 notation.

206 *A.1.5.2 Margin Code Notation*

207 Since some features may depend on one or more options, or require more than one options, a  
208 notation is used. Where a feature requires support of a single option, a single margin code will  
209 occur in the margin. If it depends on two options and both are required, then the codes will  
210 appear with a <space> separator. If either of two options are required then a logical OR is  
211 denoted using the ' | ' symbol. If more than two codes are used, a special margin code is used.



## 212 A.2 Conformance

213 The terms *profile* and *profiling* are used throughout this section.

214 A profile of a standard or standards is a codified set of option selections, such that by being  
215 conformant to a profile, particular classes of users are specifically supported.

216 These conformance definitions are descended from those in the ISO POSIX-1: 1996 standard, but  
217 with changes for the following:

- 218 • The addition of profiling options, allowing both sub-profiling as per IEEE Std. 1003.13-1998,  
219 and larger profiles of options such as the XSI extension used by the Single UNIX  
220 Specification. In effect, it has profiled itself (that is, created a self-profile).
- 221 • The addition of a hierarchy of super-options for XSI; these were formerly known as *Feature*  
222 *Groups* in The Open Group System Interfaces and Headers, Issue 5 specification.
- 223 • Options from the ISO POSIX-2: 1993 standard are also now included as IEEE Std. 1003.1-200x  
224 merges the functionality from it.

### 225 A.2.1 Implementation Conformance

226 These definitions allow application developers to know what to depend on in an  
227 implementation.

228 There is no definition of a *strictly conforming implementation*; that would be an implementation  
229 that provides *only* those facilities specified by POSIX.1 with no extensions whatsoever. This is  
230 because no actual operating system implementation can exist without system administration  
231 and initialization facilities that are beyond the scope of POSIX.1.

#### 232 A.2.1.1 Requirements

233 The word “support” is used in certain instances, rather than “provide”, in order to allow an  
234 implementation that has no resident software development facilities, but that supports the  
235 execution of a *Strictly Conforming POSIX.1 Application*, to be a *conforming implementation*.

#### 236 A.2.1.2 Documentation

237 The conformance documentation is required to use the same numbering scheme as POSIX.1 for  
238 purposes of cross-referencing. All options that an implementation chooses shall be reflected in  
239 `<limits.h>` and `<unistd.h>`.

240 Note that the use of “may” in terms of where conformance documents record where  
241 implementations may vary, implies that it is not required to describe those features identified as  
242 undefined or unspecified.

243 Other aspects of systems must be evaluated by purchasers for suitability. Many systems  
244 incorporate buffering facilities, maintaining updated data in volatile storage and transferring  
245 such updates to non-volatile storage asynchronously. Various exception conditions, such as a  
246 power failure or a system crash, can cause this data to be lost. The data may be associated with a  
247 file that is still open, with one that has been closed, with a directory, or with any other internal  
248 system data structures associated with permanent storage. This data can be lost, in whole or  
249 part, so that only careful inspection of file contents could determine that an update did not  
250 occur.

251 Also, interrelated file activities, where multiple files and/or directories are updated, or where  
252 space is allocated or released in the file system structures, can leave inconsistencies in the  
253 relationship between data in the various files and directories, or in the file system itself. Such  
254 inconsistencies can break applications that expect updates to occur in a specific sequence, so that

255 updates in one place correspond with related updates in another place.

256 For example, if a user creates a file, places information in the file, and then records this action in  
 257 another file, a system or power failure at this point followed by restart may result in a state in  
 258 which the record of the action is permanently recorded, but the file created (or some of its  
 259 information) has been lost. The consequences of this to the user may be undesirable. For a user  
 260 on such a system, the only safe action may be to require the system administrator to have a  
 261 policy that requires, after any system or power failure, that the entire file system must be  
 262 restored from the most recent backup copy (causing all intervening work to be lost).

263 The characteristics of each implementation will vary in this respect and may or may not meet the  
 264 requirements of a given application or user. Enforcement of such requirements is beyond the  
 265 scope of POSIX.1. It is up to the purchaser to determine what facilities are provided in an  
 266 implementation that affect the exposure to possible data or sequence loss, and also what  
 267 underlying implementation techniques and/or facilities are provided that reduce or limit such  
 268 loss or its consequences.

### 269 A.2.1.3 *POSIX Conformance*

270 This really means conformance to the base standard; however, since this revision includes the  
 271 core material of the Single UNIX Specification, the standard developers decided that it was  
 272 appropriate to segment the conformance requirements into two, the former for the base  
 273 standard, and the latter for the Single UNIX Specification.

274 Within POSIX.1 there are some symbolic constants that, if defined, indicate that a certain option  
 275 is enabled. Other symbolic constants exist in POSIX.1 for other reasons.

276 To enable support for sub-profiling the following options were defined:

- 277 • `_POSIX_C_LANG_SUPPORT`
- 278 • `_POSIX_DEVICE_IO`
- 279 • `_POSIX_DEVICE_SPECIFIC`
- 280 • `_POSIX_FD_MGMT`
- 281 • `_POSIX_FIFO`
- 282 • `_POSIX_FILE_ATTRIBUTES`
- 283 • `_POSIX_FILE_SYSTEM`
- 284 • `_POSIX_MULTIPLE_PROCESS`
- 285 • `_POSIX_PIPE`
- 286 • `_POSIX_SIGNALS`
- 287 • `_POSIX_SINGLE_PROCESS`
- 288 • `_POSIX_SYSTEM_DATABASE`
- 289 • `_POSIX_USER_GROUPS`
- 290 • `_POSIX_NETWORKING`

291 These are all mandatory in the base standard.

292 As part of the revision some alignment has occurred of the options with the FIPS 151-2 profile on  
 293 the POSIX.1-1990 standard. The following options from the POSIX.1-1990 standard are now  
 294 mandatory:

- 295           • `_POSIX_JOB_CONTROL`
- 296           • `_POSIX_SAVED_IDS`
- 297           • `_POSIX_VDISABLE`

298           A POSIX-conformant system may support the XSI extensions of the Single UNIX Specification.  
299           This was intentional since the standard developers intend them to be upwards-compatible, so  
300           that a system conforming to the Single UNIX Specification can also conform to the base standard  
301           at the same time.

#### 302   A.2.1.4   *XSI Conformance*

303           This section is added since the revision merges in the base volumes of the Single UNIX  
304           Specification.

305           XSI conformance can be thought of as a profile, selecting certain options from  
306           IEEE Std. 1003.1-200x.

#### 307   A.2.1.5   *Option Groups*

308           The concept of *Option Groups* is introduced to IEEE Std. 1003.1-200x to allow collections of  
309           related functions or options to be grouped together. This is used in two ways in  
310           IEEE Std. 1003.1-200x:

- 311           1. Firstly, for profiling, a set of *Profiling Option Groups* has been created to support subsetting  
312           of the system interfaces provided in IEEE Std. 1003.1-200x. The subsets used by  
313           IEEE Std. 1003.13-1998 were used as an initial model for those created.
- 314           2. Secondly, the *XSI Option Groups* have been created to allow super-options, collections of  
315           underlying options and related functions, to be collectively supported by XSI-conforming  
316           systems. These reflect the *Feature Groups* from The Open Group System Interfaces and  
317           Headers, Issue 5 specification.

#### 318   A.2.1.6   *Options*

319           The final subsections within *Implementation Conformance* list the core options within  
320           IEEE Std. 1003.1-200x. This includes both options for the System Interfaces volume of  
321           IEEE Std. 1003.1-200x and the Shell and Utilities volume of IEEE Std. 1003.1-200x.

### 322   A.2.2   **Application Conformance**

323           These definitions guide users or adaptors of applications in determining on which  
324           implementations an application will run and how much adaptation would be required to make  
325           it run on others. These definitions are modeled after related ones in the the ISO C standard.

326           POSIX.1 occasionally uses the expressions *portable application* or *conforming application*. As they  
327           are used, these are synonyms for any of these three terms. The differences between the classes of  
328           application conformance relate to the requirements for other standards, the options supported  
329           (such as the XSI extension) or, in the case of the Conforming POSIX.1 Application Using  
330           Extensions, to implementation extensions. When one of the less explicit expressions is used, it  
331           should be apparent from the context of the discussion which of the more explicit names is  
332           appropriate

333 A.2.2.1 *Strictly Conforming POSIX Application*334 This definition is analogous to that of a ISO C standard *conforming program*.335 The major difference between a *Strictly Conforming POSIX Application* and a ISO C standard  
336 *strictly conforming program* is that the latter is not allowed to use features of POSIX that are not in  
337 the ISO C standard.338 A.2.2.2 *Conforming POSIX Application*

339 Examples of &lt;National Bodies&gt; include ANSI, BSI, and AFNOR.

340 A.2.2.3 *Conforming POSIX Application Using Extensions*341 Due to possible requirements for configuration or implementation characteristics in excess of the  
342 specifications in <limits.h> or related to the hardware (such as array size or file space), not every  
343 Conforming POSIX Application Using Extensions will run on every conforming  
344 implementation.345 A.2.2.4 *Strictly Conforming XSI Application*346 This is intended to be upwards-compatible with the definition of a Strictly Conforming POSIX  
347 Application, with the addition of the facilities and functionality included in the XSI extension.348 A.2.2.5 *Conforming XSI Application Using Extensions*349 Such applications may use extensions beyond the facilities defined by IEEE Std. 1003.1-200x  
350 including the XSI extension, but need to document the additional requirements.351 **A.2.3 Language-Dependent Services for the C Programming Language**352 POSIX.1 is, for historical reasons, both a specification of an operating system interface, shell and  
353 utilities, and a C binding for that specification. Efforts had been previously undertaken to  
354 generate a language-independent specification; however, that had failed, and the fact that the  
355 ISO C standard is the *de facto* primary language on POSIX and the UNIX system makes this a  
356 necessary and workable situation.

### 357 A.3 Definitions

358 The definitions in this section are stated so that they can be used as exact substitutes for the  
359 terms in text. They should not contain requirements or cross-references to sections within  
360 IEEE Std. 1003.1-200x; that is accomplished by using an informative note. In addition, the term  
361 should not be included in its own definition. Where requirements or descriptions need to be  
362 addressed but cannot be included in the definitions, due to not meeting the above criteria, these  
363 occur in the General Concepts chapter.

364 Many of these definitions are necessarily circular, and some of the terms (such as *process*) are  
365 variants of basic computing science terms that are inherently hard to define. Where some  
366 definitions are more conceptual and contain requirements, these appear in the General Concepts  
367 chapter. Those listed in this section appear in an alphabetical glossary format of terms.

368 Some definitions must allow extension to cover terms or facilities that are not explicitly  
369 mentioned in IEEE Std. 1003.1-200x. For example, the definition of *Extended Security Controls*  
370 permits implementations beyond those defined in IEEE Std. 1003.1-200x.

371 Some terms in the following list of notes do not appear in POSIX.1; these are marked prefixed  
372 with an asterisk (\*). Many of them have been specifically excluded from POSIX.1 because they  
373 concern system administration, implementation, or other issues that are not specific to the  
374 programming interface. Those are marked with a reason, such as “implementation-defined”.

#### 375 **Appropriate Privileges**

376 One of the fundamental security problems with many historical UNIX systems has been that the  
377 privilege mechanism is monolithic—a user has either no privileges or *all* privileges. Thus, a  
378 successful “trojan horse” attack on a privileged process defeats all security provisions.  
379 Therefore, POSIX.1 allows more granular privilege mechanisms to be defined. For many  
380 historical implementations of the UNIX system, the presence of the term *appropriate privileges* in  
381 POSIX.1 may be understood as a synonym for *superuser* (UID 0). However, other systems have  
382 emerged where this is not the case and each discrete controllable action has *appropriate privileges*  
383 associated with it. Because this mechanism is implementation-defined, it must be described in  
384 the conformance document. Although that description affects several parts of POSIX.1 where  
385 the term *appropriate privilege* is used, because the term *implementation-defined* only appears here,  
386 the description of the entire mechanism and its effects on these other sections belongs in this  
387 equivalent section of the conformance document. This is especially convenient for  
388 implementations with a single mechanism that applies in all areas, since it only needs to be  
389 described once.

#### 390 **Character**

391 The term *character* is used to mean a sequence of one or more bytes representing a single graphic  
392 symbol. The deviation in the exact text of the ISO C standard definition for *byte* meets the intent  
393 of the rationale of the ISO C standard also clears up the ambiguity raised by the term *basic*  
394 *execution character set*. The octet-minimum requirement is a reflection of the {CHAR\_BIT} value.

395 **Clock Tick**

396 The ISO C standard defines a similar interval for use by the *clock()* function. There is no  
397 requirement that these intervals be the same. In historical implementations these intervals are  
398 different.

399 **Command**

400 The terms *command* and *utility* are related but have distinct meanings. to perform a specific task.  
401 The directive can be in the form of a single utility name (for example, *ls*), or the directive can take  
402 the form of a compound command (for example, "`ls | grep name | pr`"). A utility is a  
403 program that can be called by name from a shell. Issuing only the name of the utility to a shell is  
404 the equivalent of a one-word command. A utility may be invoked as a separate program that  
405 executes in a different process than the command language interpreter, or it may be  
406 implemented as a part of the command language interpreter. For example, the *echo* command  
407 (the directive to perform a specific task) may be implemented such that the *echo* utility (the logic  
408 that performs the task of echoing) is in a separate program; therefore, it is executed in a process  
409 that is different than the command language interpreter. Conversely, the logic that performs the  
410 *echo* utility could be built into the command language interpreter; therefore, it could execute in  
411 the same process as the command language interpreter.

412 The terms *tool* and *application* can be thought of as being synonymous with *utility* from the  
413 perspective of the operating system kernel. Tools, applications, and utilities historically have  
414 run, typically, in processes above the kernel level. Tools and utilities historically have been a part  
415 of the operating system non-kernel code and have performed system-related functions, such as  
416 listing directory contents, checking file systems, repairing file systems, or extracting system  
417 status information. Applications have not generally been a part of the operating system, and  
418 they perform non-system-related functions, such as word processing, architectural design,  
419 mechanical design, workstation publishing, or financial analysis. Utilities have most frequently  
420 been provided by the operating system distributor, applications by third-party software  
421 distributors, or by the users themselves. Nevertheless, IEEE Std. 1003.1-200x does not  
422 differentiate between tools, utilities, and applications when it comes to receiving services from  
423 the system, a shell, or the standard utilities. (For example, the *xargs* utility invokes another  
424 utility; it would be of fairly limited usefulness if the users could not run their own applications  
425 in place of the standard utilities.) Utilities are not applications in the sense that they are not  
426 themselves subject to the restrictions of IEEE Std. 1003.1-200x or any other standard—there is no  
427 requirement for *grep*, *stty*, or any of the utilities defined here to be any of the classes of  
428 conforming applications.

429 **Column Positions**

430 In most 1-bit character sets, such as ASCII, the concept of column positions is identical to  
431 character positions and to bytes. Therefore, it has been historically acceptable for some  
432 implementations to describe line folding or tab stops or table column alignment in terms of bytes  
433 or character positions. Other character sets pose complications, as they can have internal  
434 representations longer than one octet and they can have display characters that have different  
435 widths on the terminal screen or printer.

436 In IEEE Std. 1003.1-200x the term *column positions* has been defined to mean character—not  
437 byte—positions in input files (such as “column position 7 of the FORTRAN input”). Output files  
438 describe the column position in terms of the display width of the narrowest printable character  
439 in the character set, adjusted to fit the characteristics of the output device. It is very possible that  
440 *n* column positions will not be able to hold *n* characters in some character sets, unless all of those  
441 characters are of the narrowest width. It is assumed that the implementation is aware of the  
442 width of the various characters, deriving this information from the value of *LC\_CTYPE*, and thus

443 can determine how many column positions to allot for each character in those utilities where it is  
444 important.

445 The term *column position* was used instead of the more natural *column* because the latter is  
446 frequently used in the different contexts of columns of figures, columns of table values, and so  
447 on. Wherever confusion might result, these latter types of columns are referred to as *text*  
448 *columns*.

#### 449 **Controlling Terminal**

450 The question of which of possibly several special files referring to the terminal is meant is not  
451 addressed in POSIX.1. The file name `/dev/tty` is a synonym for the controlling terminal  
452 associated with a process.

#### 453 **Device Number\***

454 The concept is handled in *stat()* as *ID of device*.

#### 455 **Direct I/O**

456 Historically, direct I/O refers to the system bypassing intermediate buffering, but may be  
457 extended to cover implementation-defined optimizations.

#### 458 **Directory**

459 The format of the directory file is implementation-defined and differs radically between  
460 System V and 4.3 BSD. However, routines (derived from 4.3 BSD) for accessing directories and  
461 certain constraints on the format of the information returned by those routines are described in  
462 the `<dirent.h>` header.

#### 463 **Directory Entry**

464 Throughout IEEE Std. 1003.1-200x, the term *link* is used (about the *link()* function, for example)  
465 in describing the objects that point to files from directories.

#### 466 **Display**

467 The Shell and Utilities volume of IEEE Std. 1003.1-200x assigns precise requirements for the  
468 terms *display* and *write*. Some historical systems have chosen to implement certain utilities  
469 without using the traditional file descriptor model. For example, the *vi* editor might employ  
470 direct screen memory updates on a personal computer, rather than a *write()* system call. An  
471 instance of user prompting might appear in a dialog box, rather than with standard error. When  
472 the Shell and Utilities volume of IEEE Std. 1003.1-200x uses the term *display*, the method of  
473 outputting to the terminal is unspecified; many historical implementations use *termcap* or  
474 *terminfo*, but this is not a requirement. The term *write* is used when the Shell and Utilities volume  
475 of IEEE Std. 1003.1-200x mandates that a file descriptor be used and that the output can be  
476 redirected. However, it is assumed that when the writing is directly to the terminal (it has not  
477 been redirected elsewhere), there is no practical way for a user or test suite to determine whether  
478 a file descriptor is being used. Therefore, the use of a file descriptor is mandated only for the  
479 redirection case and the implementation is free to use any method when the output is not  
480 redirected. The verb *write* is used almost exclusively, with the very few exceptions of those  
481 utilities where output redirection need not be supported: *tabs*, *talk*, *tput*, and *vi*.

**482 Dot**

483 The symbolic name *dot* is carefully used in POSIX.1 to distinguish the working directory file  
484 name from a period or a decimal point.

**485 Dot-Dot**

486 Historical implementations permit the use of these file names without their special meanings.  
487 Such use precludes any meaningful use of these file names by a Conforming POSIX.1  
488 Application. Therefore, such use is considered an extension, the use of which makes an  
489 implementation non-conforming; see also Section A.4.9 (on page 3346).

**490 Epoch**

491 Historically, the origin of UNIX system time was referred to as “00:00:00 GMT, January 1, 1970”.  
492 Greenwich Mean Time is actually not a term acknowledged by the international standards  
493 community; therefore, this term, *Epoch*, is used to abbreviate the reference to the actual standard,  
494 Coordinated Universal Time.

**495 FIFO Special File**

496 See *pipe* in **Pipe** (on page 3331).

**497 File**

498 It is permissible for an implementation-defined file type to be non-readable or non-writable.

**499 File Classes**

500 These classes correspond to the historical sets of permission bits. The classes are general to  
501 allow implementations flexibility in expanding the access mechanism for more stringent security  
502 environments. Note that a process is in one and only one class, so there is no ambiguity.

**503 File Name**

504 At the present time, the primary responsibility for truncating file names containing multi-byte  
505 characters must reside with the application. Some industry groups involved in  
506 internationalization believe that in the future the responsibility must reside with the kernel. For  
507 the moment, a clearer understanding of the implications of making the kernel responsible for  
508 truncation of multi-byte file names is needed.

509 Character-level truncation was not adopted because there is no support in POSIX.1 that advises  
510 how the kernel distinguishes between single and multi-byte characters. Until that time, it must  
511 be incumbent upon application writers to determine where multi-byte characters must be  
512 truncated.

**513 File System**

514 Historically, the meaning of this term has been overloaded with two meanings: that of the  
515 complete file hierarchy, and that of a mountable subset of that hierarchy; that is, a mounted file  
516 system. POSIX.1 uses the term *file system* in the second sense, except that it is limited to the scope  
517 of a process (and a process’ root directory). This usage also clarifies the domain in which a file  
518 serial number is unique.



**519 Graphic Character**

520 This definition is made available for those definitions (in particular, *TZ*) which must exclude  
521 control characters.

**522 Group Database**

523 See **User Database** (on page 3340).

**524 Group File\***

525 Implementation-defined; see **User Database** (on page 3340).

**526 Historical Implementations\***

527 This refers to previously existing implementations of programming interfaces and operating  
528 systems that are related to the interface specified by POSIX.1.

**529 Hosted Implementation\***

530 This refers to a POSIX.1 implementation that is accomplished through interfaces from the  
531 POSIX.1 services to some alternate form of operating system kernel services. Note that the line  
532 between a hosted implementation and a native implementation is blurred, since most  
533 implementations will provide some services directly from the kernel and others through some  
534 indirect path. (For example, *fopen()* might use *open()*; or *mkfifo()* might use *mknod()*.) There is  
535 no necessary relationship between the type of implementation and its correctness, performance,  
536 and/or reliability.

**537 Implementation\***

538 This term is generally used instead of its synonym, *system*, to emphasize the consequences of  
539 decisions to be made by system implementors. Perhaps if no options or extensions to POSIX.1  
540 were allowed, this usage would not have occurred.

541 The term *specific implementation* is sometimes used as a synonym for *implementation*. This should  
542 not be interpreted too narrowly; both terms can represent a relatively broad group of systems.  
543 For example, a hardware vendor could market a very wide selection of systems that all used the  
544 same instruction set, with some systems desktop models and others large multi-user  
545 minicomputers. This wide range would probably share a common POSIX.1 operating system,  
546 allowing an application compiled for one to be used on any of the others; this is a  
547 [*specific*]implementation.

548 However, that wide range of machines probably has some differences between the models.  
549 Some may have different clock rates, different file systems, different resource limits, different  
550 network connections, and so on, depending on their sizes or intended usages. Even on two  
551 identical machines, the system administrators may configure them differently. Each of these  
552 different systems is known by the term a *specific instance of a specific implementation*. This term is  
553 only used in the portions of POSIX.1 dealing with runtime queries: *sysconf()* and *pathconf()*.

554 **Incomplete Path Name\***

555 Absolute path name has been adequately defined.

556 **Job Control**

557 In order to understand the job control facilities in POSIX.1 it is useful to understand how they  
558 are used by a job control-cognizant shell to create the user interface effect of job control.

559 While the job control facilities supplied by POSIX.1 can, in theory, support different types of  
560 interactive job control interfaces supplied by different types of shells, there is historically one  
561 particular interface that is most common (provided by BSD C Shell). This discussion describes  
562 that interface as a means of illustrating how the POSIX.1 job control facilities can be used.

563 Job control allows users to selectively stop (suspend) the execution of processes and continue  
564 (resume) their execution at a later point. The user typically employs this facility via the  
565 interactive interface jointly supplied by the terminal I/O driver and a command interpreter  
566 (shell).

567 The user can launch jobs (command pipelines) in either the foreground or background. When  
568 launched in the foreground, the shell waits for the job to complete before prompting for  
569 additional commands. When launched in the background, the shell does not wait, but  
570 immediately prompts for new commands.

571 If the user launches a job in the foreground and subsequently regrets this, the user can type the  
572 suspend character (typically set to <control>-Z), which causes the foreground job to stop and the  
573 shell to begin prompting for new commands. The stopped job can be continued by the user (via  
574 special shell commands) either as a foreground job or as a background job. Background jobs can  
575 also be moved into the foreground via shell commands.

576 If a background job attempts to access the login terminal (controlling terminal), it is stopped by  
577 the terminal driver and the shell is notified, which, in turn, notifies the user. (Terminal access  
578 includes *read()* and certain terminal control functions, and conditionally includes *write()*.) The  
579 user can continue the stopped job in the foreground, thus allowing the terminal access to  
580 succeed in an orderly fashion. After the terminal access succeeds, the user can optionally move  
581 the job into the background via the suspend character and shell commands.

582 *Implementing Job Control Shells*

583 The interactive interface described previously can be accomplished using the POSIX.1 job  
584 control facilities in the following way.

585 The key feature necessary to provide job control is a way to group processes into jobs. This  
586 grouping is necessary in order to direct signals to a single job and also to identify which job is in  
587 the foreground. (There is at most one job that is in the foreground on any controlling terminal at  
588 a time.)

589 The concept of *process groups* is used to provide this grouping. The shell places each job in a  
590 separate process group via the *setpgid()* function. To do this, the *setpgid()* function is invoked by  
591 the shell for each process in the job. It is actually useful to invoke *setpgid()* twice for each  
592 process: once in the child process, after calling *fork()* to create the process, but before calling one  
593 of the *exec* family of functions to begin execution of the program, and once in the parent shell  
594 process, after calling *fork()* to create the child. The redundant invocation avoids a race condition  
595 by ensuring that the child process is placed into the new process group before either the parent  
596 or the child relies on this being the case. The *process group ID* for the job is selected by the shell to  
597 be equal to the *process ID* of one of the processes in the job. Some shells choose to make one  
598 process in the job be the parent of the other processes in the job (if any). Other shells (for  
599 example, the C Shell) choose to make themselves the parent of all processes in the pipeline (job).

600 In order to support this latter case, the *setpgid()* function accepts a process group ID parameter  
601 since the correct process group ID cannot be inherited from the shell. The shell itself is  
602 considered to be a job and is the sole process in its own process group.

603 The shell also controls which job is currently in the foreground. A foreground and background  
604 job differ in two ways: the shell waits for a foreground command to complete (or stop) before  
605 continuing to read new commands, and the terminal I/O driver inhibits terminal access by  
606 background jobs (causing the processes to stop). Thus, the shell must work cooperatively with  
607 the terminal I/O driver and have a common understanding of which job is currently in the  
608 foreground. It is the user who decides which command should be currently in the foreground,  
609 and the user informs the shell via shell commands. The shell, in turn, informs the terminal I/O  
610 driver via the *tcsetpgrp()* function. This indicates to the terminal I/O driver the process group ID  
611 of the foreground process group (job). When the current foreground job either stops or  
612 terminates, the shell places itself in the foreground via *tcsetpgrp()* before prompting for  
613 additional commands. Note that when a job is created the new process group begins as a  
614 background process group. It requires an explicit act of the shell via *tcsetpgrp()* to move a  
615 process group (job) into the foreground.

616 When a process in a job stops or terminates, its parent (for example, the shell) receives  
617 synchronous notification by calling the *waitpid()* function with the WUNTRACED flag set.  
618 Asynchronous notification is also provided when the parent establishes a signal handler for  
619 SIGCHLD and does not specify the SA\_NOCLDSTOP flag. Usually all processes in a job stop as  
620 a unit since the terminal I/O driver always sends job control stop signals to all processes in the  
621 process group.

622 To continue a stopped job, the shell sends the SIGCONT signal to the process group of the job. In  
623 addition, if the job is being continued in the foreground, the shell invokes *tcsetpgrp()* to place the  
624 job in the foreground before sending SIGCONT. Otherwise, the shell leaves itself in the  
625 foreground and reads additional commands.

626 There is additional flexibility in the POSIX.1 job control facilities that allows deviations from the  
627 typical interface. Clearing the TOSTOP terminal flag allows background jobs to perform *write()*  
628 functions without stopping. The same effect can be achieved on a per-process basis by having a  
629 process set the signal action for SIGTTOU to SIG\_IGN.

630 Note that the terms *job* and *process group* can be used interchangeably. A login session that is not  
631 using the job control facilities can be thought of as a large collection of processes that are all in  
632 the same job (process group). Such a login session may have a partial distinction between  
633 foreground and background processes; that is, the shell may choose to wait for some processes  
634 before continuing to read new commands and may not wait for other processes. However, the  
635 terminal I/O driver will consider all these processes to be in the foreground since they are all  
636 members of the same process group.

637 In addition to the basic job control operations already mentioned, a job control-cognizant shell  
638 needs to perform the following actions.

639 When a foreground (not background) job stops, the shell must sample and remember the current  
640 terminal settings so that it can restore them later when it continues the stopped job in the  
641 foreground (via the *tcgetattr()* and *tcsetattr()* functions).

642 Because a shell itself can be spawned from a shell, it must take special action to ensure that  
643 subshells interact well with their parent shells.

644 A subshell can be spawned to perform an interactive function (prompting the terminal for  
645 commands) or a non-interactive function (reading commands from a file). When operating non-  
646 interactively, the job control shell will refrain from performing the job control-specific actions  
647 described above. It will behave as a shell that does not support job control. For example, all *jobs*

648 will be left in the same process group as the shell, which itself remains in the process group  
649 established for it by its parent. This allows the shell and its children to be treated as a single job  
650 by a parent shell, and they can be affected as a unit by terminal keyboard signals.

651 An interactive subshell can be spawned from another job control-cognizant shell in either the  
652 foreground or background. (For example, from the C Shell, the user can execute the command,  
653 `csch &`.) Before the subshell activates job control by calling `setpgid()` to place itself in its own  
654 process group and `tcsetpgrp()` to place its new process group in the foreground, it needs to  
655 ensure that it has already been placed in the foreground by its parent. (Otherwise, there could  
656 be multiple job control shells that simultaneously attempt to control mediation of the terminal.)  
657 To determine this, the shell retrieves its own process group via `getpgrp()` and the process group  
658 of the current foreground job via `tcgetpgrp()`. If these are not equal, the shell sends SIGTTIN to  
659 its own process group, causing itself to stop. When continued later by its parent, the shell  
660 repeats the process group check. When the process groups finally match, the shell is in the  
661 foreground and it can proceed to take control. After this point, the shell ignores all the job  
662 control stop signals so that it does not inadvertently stop itself.

### 663 *Implementing Job Control Applications*

664 Most applications do not need to be aware of job control signals and operations; the intuitively  
665 correct behavior happens by default. However, sometimes an application can inadvertently  
666 interfere with normal job control processing, or an application may choose to overtly effect job  
667 control in cooperation with normal shell procedures.

668 An application can inadvertently subvert job control processing by “blindly” altering the  
669 handling of signals. A common application error is to learn how many signals the system  
670 supports and to ignore or catch them all. Such an application makes the assumption that it does  
671 not know what this signal is, but knows the right handling action for it. The system may  
672 initialize the handling of job control stop signals so that they are being ignored. This allows  
673 shells that do not support job control to inherit and propagate these settings and hence to be  
674 immune to stop signals. A job control shell will set the handling to the default action and  
675 propagate this, allowing processes to stop. In doing so, the job control shell is taking  
676 responsibility for restarting the stopped applications. If an application wishes to catch the stop  
677 signals itself, it should first determine their inherited handling states. If a stop signal is being  
678 ignored, the application should continue to ignore it. This is directly analogous to the  
679 recommended handling of SIGINT described in the referenced UNIX Programmer’s Manual.

680 If an application is reading the terminal and has disabled the interpretation of special characters  
681 (by clearing the ISIG flag), the terminal I/O driver will not send SIGTSTP when the suspend  
682 character is typed. Such an application can simulate the effect of the suspend character by  
683 recognizing it and sending SIGTSTP to its process group as the terminal driver would have  
684 done. Note that the signal is sent to the process group, not just to the application itself; this  
685 ensures that other processes in the job also stop. (Note also that other processes in the job could  
686 be children, siblings, or even ancestors.) Applications should not assume that the suspend  
687 character is `<control>-Z` (or any particular value); they should retrieve the current setting at  
688 startup.

### 689 *Implementing Job Control Systems*

690 The intent in adding 4.2 BSD-style job control functionality was to adopt the necessary 4.2 BSD  
691 programmatic interface with only minimal changes to resolve syntactic or semantic conflicts  
692 with System V or to close recognized security holes. The goal was to maximize the ease of  
693 providing both conforming implementations and Conforming POSIX.1 Applications.

694 It is only useful for a process to be affected by job control signals if it is the descendant of a job  
695 control shell. Otherwise, there will be nothing that continues the stopped process.

696 POSIX.1 does not specify how controlling terminal access is affected by a user logging out (that  
697 is, by a controlling process terminating). 4.2 BSD uses the *vhangup()* function to prevent any  
698 access to the controlling terminal through file descriptors opened prior to logout. System V does  
699 not prevent controlling terminal access through file descriptors opened prior to logout (except  
700 for the case of the special file, */dev/tty*). Some implementations choose to make processes  
701 immune from job control after logout (that is, such processes are always treated as if in the  
702 foreground); other implementations continue to enforce foreground/background checks after  
703 logout. Therefore, a Conforming POSIX.1 Application should not attempt to access the  
704 controlling terminal after logout since such access is unreliable. If an implementation chooses to  
705 deny access to a controlling terminal after its controlling process exits, POSIX.1 requires a certain  
706 type of behavior (see **Controlling Terminal** (on page 3323)).

#### 707 **Kernel\***

708 See *system call*.

#### 709 **Library Routine\***

710 See *system call*.

#### 711 **Logical Device\***

712 Implementation-defined.

#### 713 **Map**

714 The definition of map is included to clarify the usage of mapped pages in the description of the  
715 behavior of process memory locking.

#### 716 **Memory-Resident**

717 The term *memory-resident* is historically understood to mean that the so-called resident pages are  
718 actually present in the physical memory of the computer system and are immune from  
719 swapping, paging, copy-on-write faults, and so on. This is the actual intent of  
720 IEEE Std. 1003.1-200x in the process memory locking section for implementations where this is  
721 logical. But for some implementations—primarily mainframes—actually locking pages into  
722 primary storage is not advantageous to other system objectives, such as maximizing throughput.  
723 For such implementations, memory locking is a “hint” to the implementation that the  
724 application wishes to avoid situations that would cause long latencies in accessing memory.  
725 Furthermore, there are other implementation-defined issues with minimizing memory access  
726 latencies that “memory residency” does not address—such as MMU reload faults. The definition  
727 attempts to accommodate various implementations while allowing portable applications to  
728 specify to the implementation that they want or need the best memory access times that the  
729 implementation can provide.

#### 730 **Memory Object\***

731 The term *memory object* usually implies shared memory. If the object is the same as a file name in  
732 the file system name space of the implementation, it is expected that the data written into the  
733 memory object be preserved on disk. A memory object may also apply to a physical device on an  
734 implementation. In this case, writes to the memory object are sent to the controller for the device  
735 and reads result in control registers being returned.

736 **Mount Point\***

737 The directory on which a *mounted file system* is mounted. This term, like *mount()* and *umount()*,  
738 was not included because it was implementation-defined.

739 **Mounted File System\***

740 See *file system*.

741 **name**

742 There are no explicit limits in IEEE Std. 1003.1-200x on the sizes of names, words (see the  
743 definition of word in the Base Definitions volume of IEEE Std. 1003.1-200x ), lines, or other  
744 objects. However, other implicit limits do apply: shell script lines produced by many of the  
745 standard utilities cannot exceed {LINE\_MAX} and the sum of exported variables comes under the  
746 {ARG\_MAX} limit. Historical shells dynamically allocate memory for names and words and  
747 parse incoming lines a byte at a time. Lines cannot have an arbitrary {LINE\_MAX} limit because  
748 of historical practice, such as makefiles, where *make* removes the <newline> characters  
749 associated with the commands for a target and presents the shell with one very long line. The  
750 text on INPUT FILES in the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 1.11,  
751 Utility Description Defaults does allow a shell to run out of memory, but it cannot have arbitrary  
752 programming limits.

753 **Native Implementation\***

754 This refers to an implementation of POSIX.1 that interfaces directly to an operating system  
755 kernel; see also *hosted implementation* and *cooperating implementation*. A similar concept is a  
756 native UNIX system, which would be a kernel derived from one of the original UNIX system  
757 products.

758 **Nice Value**

759 This definition is not intended to suggest that all processes in a system have priorities that are  
760 comparable. Scheduling policy extensions, such as adding realtime priorities, make the notion of  
761 a single underlying priority for all scheduling policies problematic. Some systems may  
762 implement the features related to *nice* to affect all processes on the system, others to affect just  
763 the general time-sharing activities implied by IEEE Std. 1003.1-200x, and others may have no  
764 effect at all. Because of the use of “implementation-defined” in *nice* and *renice*, a wide range of  
765 implementation strategies is possible.

766 **Open File Description**

767 An *open file description*, as it is currently named, describes how a file is being accessed. What is  
768 currently called a *file descriptor* is actually just an identifier or “handle”; it does not actually  
769 describe anything.

770 The following alternate names were discussed:

- 771 • For *open file description*:  
772 *open instance*, *file access description*, *open file information*, and *file access information*.
- 773 • For *file descriptor*:  
774 *file handle*, *file number* (c.f., *fileno()*). Some historical implementations use the term *file table*  
775 *entry*.

**776 Orphaned Process Group**

777 Historical implementations have a concept of an orphaned process, which is a process whose  
778 parent process has exited. When job control is in use, it is necessary to prevent processes from  
779 being stopped in response to interactions with the terminal after they no longer are controlled by  
780 a job control-cognizant program. Because signals generated by the terminal are sent to a process  
781 group and not to individual processes, and because a signal may be provoked by a process that  
782 is not orphaned, but sent to another process that is orphaned, it is necessary to define an  
783 orphaned process group. The definition assumes that a process group will be manipulated as a  
784 group and that the job control-cognizant process controlling the group is outside of the group  
785 and is the parent of at least one process in the group (so that state changes may be reported via  
786 *waitpid()*). Therefore, a group is considered to be controlled as long as at least one process in the  
787 group has a parent that is outside of the process group, but within the session.

788 This definition of orphaned process groups ensures that a session leader's process group is  
789 always considered to be orphaned, and thus it is prevented from stopping in response to  
790 terminal signals.

**791 Page**

792 The term *page* is defined to support the description of the behavior of memory mapping for  
793 shared memory and memory mapped files, and the description of the behavior of process  
794 memory locking. It is not intended to imply that shared memory/file mapping and memory  
795 locking are applicable only to "paged" architectures. For the purposes of IEEE Std. 1003.1-200x,  
796 whatever the granularity on which an architecture supports mapping or locking is considered to  
797 be a "page". If an architecture cannot support the memory mapping or locking functions  
798 specified by IEEE Std. 1003.1-200x on any granularity, then these options will not be  
799 implemented on the architecture.

**800 Passwd File\***

801 Implementation-defined; see **User Database** (on page 3340).

**802 Parent Directory**

803 There may be more than one directory entry pointing to a given directory in some  
804 implementations. The wording here identifies that exactly one of those is the parent directory. In  
805 *path name resolution*, dot-dot is identified as the way that the unique directory is identified. (That  
806 is, the parent directory is the one to which dot-dot points.) In the case of a remote file system, if  
807 the same file system is mounted several times, it would appear as if they were distinct file  
808 systems (with interesting synchronization properties).

**809 Pipe**

810 It proved convenient to define a pipe as a special case of a FIFO, even though historically the  
811 latter was not introduced until System III and does not exist at all in 4.3 BSD.

**812 Portable File Name Character Set**

813 The encoding of this character set is not specified—specifically, ASCII is not required. But the  
814 implementation must provide a unique character code for each of the printable graphics  
815 specified by POSIX.1; see also Section A.4.5 (on page 3342).

816 Situations where characters beyond the portable file name character set (or historically ASCII or  
817 the ISO/IEC 646:1991 standard) would be used (in a context where the portable file name  
818 character set or the ISO/IEC 646:1991 standard is required by POSIX.1) are expected to be  
819 common. Although such a situation renders the use technically non-compliant, mutual  
820 agreement among the users of an extended character set will make such use portable between  
821 those users. Such a mutual agreement could be formalized as an optional extension to POSIX.1.  
822 (Making it required would eliminate too many possible systems, as even those systems using the  
823 ISO/IEC 646:1991 standard as a base character set extend their character sets for Western  
824 Europe and the rest of the world in different ways.)

825 Nothing in POSIX.1 is intended to preclude the use of extended characters where interchange is  
826 not required or where mutual agreement is obtained. It has been suggested that in several places  
827 “should” be used instead of “shall”. Because (in the worst case) use of any character beyond the  
828 portable file name character set would render the program or data not portable to all possible  
829 systems, no extensions are permitted in this context.

**830 Regular File**

831 POSIX.1 does not intend to preclude the addition of structuring data (for example, record  
832 lengths) in the file, as long as such data is not visible to an application that uses the features  
833 described in POSIX.1.

**834 Root Directory**

835 This definition permits the operation of *chroot()*, even though that function is not in POSIX.1; see  
836 also *file hierarchy*.

**837 Root File System\***

838 Implementation-defined.

**839 Root of a File System\***

840 Implementation-defined; see *mount point*.

**841 Seconds Since the Epoch**

842 Coordinated Universal Time uses the concept of leap seconds; at the time POSIX.1 was  
843 published, 14 leap seconds had been added since January 1, 1970. These 14 seconds are ignored  
844 to provide an easy and compatible method of computing time differences.

845 Most systems’ notion of “time” is that of a continuously increasing value, so this value should  
846 increase even during leap seconds. However, not only do most systems not keep track of leap  
847 seconds, but most systems are probably not synchronized to any standard time reference.  
848 Therefore, it is inappropriate to require that a time represented as seconds since the Epoch  
849 precisely represent the number of seconds between the referenced time and the Epoch.

850 It is sufficient to require that applications be allowed to treat this time as if it represented the  
851 number of seconds between the referenced time and the Epoch. It is the responsibility of the  
852 vendor of the system, and the administrator of the system, to ensure that this value represents  
853 the number of seconds between the referenced time and the Epoch as closely as necessary for the



854 application being run on that system.

855 It is important that the interpretation of time names and *seconds since the Epoch* values be  
 856 consistent across conforming systems; that is, it is important that all conforming systems  
 857 interpret “536 457 599 seconds since the Epoch” as 59 seconds, 59 minutes, 23 hours 31 December  
 858 1986, regardless of the accuracy of the system’s idea of the current time. The expression is given  
 859 to assure a consistent interpretation, not to attempt to specify the calendar. The relationship  
 860 between *tm\_yday* and the day of week, day of month, and month is presumed to be specified  
 861 elsewhere and is not given in POSIX.1.

862 Consistent interpretation of *seconds since the Epoch* can be critical to certain types of distributed  
 863 applications that rely on such timestamps to synchronize events. The accrual of leap seconds in  
 864 a time standard is not predictable. The number of leap seconds since the Epoch will likely  
 865 increase. POSIX.1 is more concerned about the synchronization of time between applications of  
 866 astronomically short duration. These concerns are expected to become more critical in the future.

867 Note that *tm\_yday* is zero-based, not one-based, so the day number in the example above is 364.  
 868 Note also that the division is an integer division (discarding remainder) as in the C language.

869 Note also that the meaning of *gmtime()*, *localtime()*, and *mktime()* is specified in terms of this  
 870 expression. However, the ISO C standard computes *tm\_yday* from *tm\_mday*, *tm\_mon*, and  
 871 *tm\_year* in *mktime()*. Because it is stated as a (bidirectional) relationship, not a function, and  
 872 because the conversion between month-day-year and day-of-year dates is presumed well known  
 873 and is also a relationship, this is not a problem.

874 Implementations that implement **time\_t** as a 32-bit integer will overflow in 2 038. POSIX.1 does  
 875 not Specify the data size for **time\_t**.

876 See also **Epoch** (on page 3324).

## 877 **Signal**

878 The definition implies a double meaning for the term. Although a signal is an event, common  
 879 usage implies that a signal is an identifier of the class of event.

## 880 **Superuser\***

881 This concept, with great historical significance to UNIX system users, has been replaced with the  
 882 notion of appropriate privileges.

## 883 **Supplementary Group ID**

884 The POSIX.1-1990 standard is inconsistent in its treatment of supplementary groups. The  
 885 definition of supplementary group ID explicitly permits the effective group ID to be included in  
 886 the set, but wording in the description of the *setuid()* and *setgid()* functions states: “Any  
 887 supplementary group IDs of the calling process remain unchanged by these function calls”. In  
 888 the case of *setgid()* this contradicts that definition. In addition, some felt that the unspecified  
 889 behavior in the definition of supplementary group IDs adds unnecessary portability problems.  
 890 The standard developers considered several solutions to this problem:

- 891 1. Reword the description of *setgid()* to permit it to change the supplementary group IDs to  
 892 reflect the new effective group ID. A problem with this is that it adds more “may”s to the  
 893 wording and does not address the portability problems of this optional behavior.
- 894 2. Mandate the inclusion of the effective group ID in the supplementary set (giving  
 895 {NGROUPS\_MAX} a minimum value of 1). This is the behavior of 4.4 BSD. In that system,  
 896 the effective group ID is the first element of the array of supplementary group IDs (there is  
 897 no separate copy stored, and changes to the effective group ID are made only in the

898 supplementary group set). By convention, the initial value of the effective group ID is  
 899 duplicated elsewhere in the array so that the initial value is not lost when executing a set-  
 900 group-ID program.

901 3. Change the definition of supplementary group ID to exclude the effective group ID and  
 902 specify that the effective group ID does not change the set of supplementary group IDs.  
 903 This is the behavior of 4.2 BSD, 4.3 BSD, and System V, Release 4.

904 4. Change the definition of supplementary group ID to exclude the effective group ID, and  
 905 require that *getgroups()* return the union of the effective group ID and the supplementary  
 906 group IDs.

907 5. Change the definition of {NGROUPS\_MAX} to be one more than the number of  
 908 supplementary group IDs, so it continues to be the number of values returned by  
 909 *getgroups()* and existing applications continue to work. This alternative is effectively the  
 910 same as the second (and might actually have the same implementation).

911 The standard developers decided to permit either 2 or 3. The effective group ID is orthogonal to  
 912 the set of supplementary group IDs, and it is implementation-defined whether *getgroups()*  
 913 returns this. If the effective group ID is returned with the set of supplementary group IDs, then  
 914 all changes to the effective group ID affect the supplementary group set returned by *getgroups()*.  
 915 It is permissible to eliminate duplicates from the list returned by *getgroups()*. However, if a  
 916 group ID is contained in the set of supplementary group IDs, setting the group ID to that value  
 917 and then to a different value should not remove that value from the supplementary group IDs.

918 The definition of supplementary group IDs has been changed to not include the effective group  
 919 ID. This simplifies permanent rationale and makes the relevant functions easier to understand.  
 920 The *getgroups()* function has been modified so that it can, on an implementation-defined basis,  
 921 return the effective group ID. By making this change, functions that modify the effective group  
 922 ID do not need to discuss adding to the supplementary group list; the only view into the  
 923 supplementary group list that the application writer has is through the *getgroups()* function.

## 924 **Symbolic Link**

925 Many implementations associate no attributes, including ownership with symbolic links.  
 926 Security experts encouraged consideration for defining these attributes as optional.  
 927 Consideration was given to changing *utime()* to allow modification of the times for a symbolic  
 928 link, or as an alternative adding an *lutime()* interface. Modifications to *chown()* were also  
 929 considered: allow changing symbolic link ownership or alternatively adding *lchown()*. As a  
 930 result of the problems encountered in defining attributes for symbolic links (and interfaces to  
 931 access/modify those attributes) and since implementations exist that do not associate these  
 932 attributes with symbolic links, only the file type bits in the *st\_mode* member and the *st\_size*  
 933 member of the **stat** structure are required to be applicable to symbolic links.

934 Historical implementations were followed when determining which interfaces should apply to  
 935 symbolic links. Interfaces that historically followed symbolic links include *chmod()*, *link()*, and  
 936 *utime()*. Interfaces that historically do not follow symbolic links include *chown()*, *lstat()*,  
 937 *readlink()*, *rename()*, *remove()*, *rmdir()*, and *unlink()*. IEEE Std. 1003.1-200x deviates from  
 938 historical practice only in the case of *chown()*. Because there is no requirement that there be an  
 939 association of ownership with symbolic links, there was no point in requiring an interface to  
 940 change ownership. In addition, other implementations of symbolic links have modified *chown()*  
 941 to follow symbolic links.

942 In the case of symbolic links, IEEE Std. 1003.1-200x states that a trailing slash is considered to be  
 943 the final component of a path name rather than the path name component that preceded it. This  
 944 is the behavior of historical implementations. For example, for **/a/b** and **/a/b/**, if **/a/b** is a symbolic

945 link to a directory, then `/a/b` refers to the symbolic link, and `/a/b/` is the same as `/a/b/.`, which is the  
946 directory to which the symbolic link points.

947 For multi-level security purposes, it is possible to have the link read mode govern permission for  
948 the `readlink()` function. It is also possible that the read permissions of the directory containing  
949 the link be used for this purpose. Implementations may choose to use either of these methods;  
950 however, this is not current practice and neither method is specified.

951 Several reasons were advanced for requiring that when a symbolic link is used as the source  
952 argument to the `link()` function, the resulting link will apply to the file named by the contents of  
953 the symbolic link rather than to the symbolic link itself. This is the case in historical  
954 implementations. This action was preferred, as it supported the traditional idea of persistence  
955 with respect to the target of a hard link. This decision is appropriate in light of a previous  
956 decision not to require association of attributes with symbolic links, thereby allowing  
957 implementations which do not use inodes. Opposition centered on the lack of symmetry on the  
958 part of the `link()` and `unlink()` function pair with respect to symbolic links.

959 Because a symbolic link and its referenced object coexist in the file system name space, confusion  
960 can arise in distinguishing between the link itself and the referenced object. Historically, utilities  
961 and system calls have adopted their own link following conventions in a somewhat *ad hoc*  
962 fashion. Rules for a uniform approach are outlined here, although historical practice has been  
963 adhered to as much as was possible. To promote consistent system use, user-written utilities are  
964 encouraged to follow these same rules.

965 Symbolic links are handled either by operating on the link itself, or by operating on the object  
966 referenced by the link. In the latter case, an application or system call is said to follow the link.  
967 Symbolic links may reference other symbolic links, in which case links are dereferenced until an  
968 object that is not a symbolic link is found, a symbolic link that references a file that does not exist  
969 is found, or a loop is detected. (Current implementations do not detect loops, but have a limit on  
970 the number of symbolic links that they will dereference before declaring it an error.)

971 There are four domains for which default symbolic link policy is established in a system. In  
972 almost all cases, there are utility options that override this default behavior. The four domains  
973 are as follows:

- 974 1. Symbolic links specified to system calls that take file name arguments
- 975 2. Symbolic links specified as command line file name arguments to utilities that are not  
976 performing a traversal of a file hierarchy
- 977 3. Symbolic links referencing files not of type directory, specified to utilities that are  
978 performing a traversal of a file hierarchy
- 979 4. Symbolic links referencing files of type directory, specified to utilities that are performing a  
980 traversal of a file hierarchy

#### 981 *First Domain*

982 The first domain is considered in earlier rationale.

#### 983 *Second Domain*

984 The reason this category is restricted to utilities that are not traversing the file hierarchy is that  
985 some standard utilities take an option that specifies a hierarchical traversal, but by default  
986 operate on the arguments themselves. Generally, users specifying the option for a file hierarchy  
987 traversal wish to operate on a single, physical hierarchy, and therefore symbolic links, which  
988 may reference files outside of the hierarchy, are ignored. For example, *chown owner file* is a  
989 different operation from the same command with the `-R` option specified. In this example, the  
990 behavior of the command *chown owner file* is described here, while the behavior of the command

- 991 *chown* **-R** *owner file* is described in the third and fourth domains.
- 992 The general rule is that the utilities in this category follow symbolic links named as arguments.
- 993 Exceptions in the second domain are:
- 994 • The *mv* and *rm* utilities do not follow symbolic links named as arguments, but respectively  
995 attempt to rename or delete them.
  - 996 • The *ls* utility is also an exception to this rule. For compatibility with historical systems, when  
997 the **-R** option is not specified, the *ls* utility follows symbolic links named as arguments if the  
998 **-L** option is specified or if the **-F**, **-d**, or **-l** options are not specified. (If the **-L** option is  
999 specified, *ls* always follows symbolic links; it is the only utility where the **-L** option affects its  
1000 behavior even though a tree walk is not being performed.)
- 1001 All other standard utilities, when not traversing a file hierarchy, always follow symbolic links  
1002 named as arguments.
- 1003 Historical practice is that the **-h** option is specified if standard utilities are to act upon symbolic  
1004 links instead of upon their targets. Examples of commands that have historically had a **-h** option  
1005 for this purpose are the *chgrp*, *chown*, *file*, and *test* utilities.
- 1006 *Third Domain*
- 1007 The third domain is symbolic links, referencing files not of type directory, specified to utilities  
1008 that are performing a traversal of a file hierarchy. (This includes symbolic links specified as  
1009 command line file name arguments or encountered during the traversal.)
- 1010 The intention of the Shell and Utilities volume of IEEE Std. 1003.1-200x is that the operation that  
1011 the utility is performing is applied to the symbolic link itself, if that operation is applicable to  
1012 symbolic links. The reason that the operation is not required is that symbolic links in some  
1013 systems do not have such attributes as a file owner, and therefore the *chown* operation would be  
1014 meaningless. If symbolic links on the system have an owner, it is the intention that the utility  
1015 *chown* cause the owner of the symbolic link to change. If symbolic links do not have an owner,  
1016 the symbolic link should be ignored. Specifically, by default, no change should be made to the  
1017 file referenced by the symbolic link.
- 1018 *Fourth Domain*
- 1019 The fourth domain is symbolic links referencing files of type directory, specified to utilities that  
1020 are performing a traversal of a file hierarchy. (This includes symbolic links specified as  
1021 command line file name arguments or encountered during the traversal.)
- 1022 All standard utilities do not, by default, indirect into the file hierarchy referenced by the  
1023 symbolic link. (The Shell and Utilities volume of IEEE Std. 1003.1-200x uses the informal term  
1024 *physical walk* to describe this case. The case where the utility does indirect through the symbolic  
1025 link is termed a *logical walk*.)
- 1026 There are three reasons for the default to a physical walk:
- 1027 1. With very few exceptions, a physical walk has been the historical default on UNIX systems  
1028 supporting symbolic links. Because some utilities (that is, *rm*) must default to a physical  
1029 walk, regardless, changing historical practice in this regard would be confusing to users  
1030 and needlessly incompatible.
  - 1031 2. For systems where symbolic links have the historical file attributes (that is, *owner*, *group*,  
1032 *mode*), defaulting to a logical traversal would require the addition of a new option to the  
1033 commands to modify the attributes of the link itself. This is painful and more complex  
1034 than the alternatives.

1035 3. There is a security issue with defaulting to a logical walk. Historically, the command  
1036 *chown -R user file* has been safe for the superuser because *setuid* and *setgid* bits were lost  
1037 when the ownership of the file was changed. If the walk were logical, changing ownership  
1038 would no longer be safe because a user might have inserted a symbolic link pointing to any  
1039 file in the tree. Again, this would necessitate the addition of an option to the commands  
1040 doing hierarchy traversal to not indirect through the symbolic links, and historical scripts  
1041 doing recursive walks would instantly become security problems. While this is mostly an  
1042 issue for system administrators, it is preferable to not have different defaults for different  
1043 classes of users.

1044 As consistently as possible, users may cause standard utilities performing a file hierarchy  
1045 traversal to follow any symbolic links named on the command line, regardless of the type of file  
1046 they reference, by specifying the **-H** (for half logical) option. This option is intended to make the  
1047 command line name space look like the logical name space.

1048 As consistently as possible, users may cause standard utilities performing a file hierarchy  
1049 traversal to follow any symbolic links named on the command line as well as any symbolic links  
1050 encountered during the traversal, regardless of the type of file they reference, by specifying the  
1051 **-L** (for logical) option. This option is intended to make the entire name space look like the  
1052 logical name space.

1053 For consistency, implementors are encouraged to use the **-P** (for physical) flag to specify the  
1054 physical walk in utilities that do logical walks by default for whatever reason. The only standard  
1055 utilities that require the **-P** option are *cd* and *pwd*; see the note below.

1056 When one or more of the **-H**, **-L**, and **-P** flags can be specified, the last one specified determines  
1057 the behavior of the utility. This permits users to alias commands so that the default behavior is a  
1058 logical walk and then override that behavior on the command line.

#### 1059 *Exceptions in the Third and Fourth Domains*

1060 The *ls* and *rm* utilities are exceptions to these rules. The *rm* utility never follows symbolic links  
1061 and does not support the **-H**, **-L**, or **-P** options. Some historical versions of *ls* always followed  
1062 symbolic links given on the command line whether the **-L** option was specified or not. Historical  
1063 versions of *ls* did not support the **-H** option. In IEEE Std. 1003.1-200x, the *ls* utility never follows  
1064 symbolic links unless one of the **-H** or **-L** options is specified. The *ls* utility does not support the  
1065 **-P** option.

1066 The Shell and Utilities volume of IEEE Std. 1003.1-200x requires that the standard utilities *ls*, *find*,  
1067 and *pax* detect infinite loops when doing logical walks; that is, a directory, or more commonly a  
1068 symbolic link, that refers to an ancestor in the current file hierarchy. If the file system itself is  
1069 corrupted, causing the infinite loop, it may be impossible to recover. Because *find* and *ls* are often  
1070 used in system administration and security applications, they should attempt to recover and  
1071 continue as best as they can. The *pax* utility should terminate because the archive it was creating  
1072 is by definition corrupted. Other, less vital, utilities should probably simply terminate as well.  
1073 Implementations are strongly encouraged to detect infinite loops in all utilities.

1074 Historical practice is shown in Table A-1 (on page 3338). The heading **SVID3** stands for the  
1075 Third Edition of the System V Interface Definition.

1076 Historically, several shells have had built-in versions of the *pwd* utility. In some of these shells,  
1077 *pwd* reported the physical path, and in others, the logical path. Implementations of the shell  
1078 corresponding to IEEE Std. 1003.1-200x must report the logical path by default. Earlier versions  
1079 of IEEE Std. 1003.1-200x did not require the *pwd* utility to be a built-in utility. Now that *pwd* is  
1080 required to set an environment variable in the current shell execution environment, it must be a  
1081 built-in utility.

1082 The *cd* command is required, by default, to treat the file name dot-dot logically. Implementors  
 1083 are required to support the **-P** flag in *cd* so that users can have their current environment  
 1084 handled physically. In 4.3 BSD, *chgrp* during tree traversal changed the group of the symbolic  
 1085 link, not the target. Symbolic links in 4.4 BSD do not have *owner*, *group*, *mode*, or other standard  
 1086 UNIX system file attributes.

1087 **Table A-1** Historical Practice for Symbolic Links

| Utility           | SVID3   | 4.3 BSD | 4.4 BSD | POSIX | Comments                                |
|-------------------|---------|---------|---------|-------|-----------------------------------------|
| 1088 <i>cd</i>    |         |         |         | -L    | Treat ". ." logically.                  |
| 1089 <i>cd</i>    |         |         |         | -P    | ". ." physically.                       |
| 1090 <i>chgrp</i> |         |         | -H      | -H    | Follow command line symlinks.           |
| 1091 <i>chgrp</i> |         |         | -h      | -L    | Follow symlinks.                        |
| 1092 <i>chgrp</i> | -h      |         |         | -h    | Affect the symlink.                     |
| 1093 <i>chmod</i> |         |         |         | -h    | Affect the symlink.                     |
| 1094 <i>chmod</i> |         |         | -H      | -H    | Follow command line symlinks.           |
| 1095 <i>chmod</i> |         |         | -h      | -L    | Follow symlinks.                        |
| 1096 <i>chown</i> |         |         | -H      | -H    | Follow command line symlinks.           |
| 1097 <i>chown</i> |         |         | -h      | -L    | Follow symlinks.                        |
| 1098 <i>chown</i> | -h      |         |         | -h    | Affect the symlink.                     |
| 1099 <i>cp</i>    |         |         | -H      | -H    | Follow command line symlinks.           |
| 1100 <i>cp</i>    |         |         | -h      | -L    | Follow symlinks.                        |
| 1101 <i>cpio</i>  | -L      |         | -L      |       | Follow symlinks.                        |
| 1102 <i>du</i>    |         |         | -H      | -H    | Follow command line symlinks.           |
| 1103 <i>du</i>    |         |         | -h      | -L    | Follow symlinks.                        |
| 1104 <i>file</i>  | -h      |         |         | -h    | Affect the symlink.                     |
| 1105 <i>find</i>  |         |         | -H      | -H    | Follow command line symlinks.           |
| 1106 <i>find</i>  |         |         | -h      | -L    | Follow symlinks.                        |
| 1107 <i>find</i>  | -follow |         | -follow |       | Follow symlinks.                        |
| 1108 <i>ln</i>    | -s      | -s      | -s      | -s    | Create a symbolic link.                 |
| 1109 <i>ls</i>    | -L      | -L      | -L      | -L    | Follow symlinks.                        |
| 1110 <i>ls</i>    |         |         |         | -H    | Follow command line symlinks.           |
| 1111 <i>mv</i>    |         |         |         |       | Operates on the symlink.                |
| 1112 <i>pax</i>   |         |         | -H      | -H    | Follow command line symlinks.           |
| 1113 <i>pax</i>   |         |         | -h      | -L    | Follow symlinks.                        |
| 1114 <i>pwd</i>   |         |         |         | -L    | Printed path may contain symlinks.      |
| 1115 <i>pwd</i>   |         |         |         | -P    | Printed path will not contain symlinks. |
| 1116 <i>rm</i>    |         |         |         |       | Operates on the symlink.                |
| 1117 <i>tar</i>   |         |         | -H      |       | Follow command line symlinks.           |
| 1118 <i>tar</i>   |         | -h      | -h      |       | Follow symlinks.                        |
| 1119 <i>test</i>  | -h      |         | -h      | -h    | Affect the symlink.                     |

### 1121 Synchronously-Generated Signal

1122 Those signals that may be generated synchronously include SIGABRT, SIGBUS, SIGILL, SIGFPE,  
 1123 SIGPIPE, and SIGSEGV.

**1124 System Call\***

1125 The distinction between a *system call* and a *library routine* is an implementation detail that may  
1126 differ between implementations and has thus been excluded from POSIX.1.

1127 See “Interface, Not Implementation” in the Introduction.

**1128 System Reboot**

1129 A *system reboot* is an event initiated by an unspecified circumstance that causes all processes  
1130 (other than special system processes) to be terminated in an implementation-defined manner,  
1131 after which any changes to the state and contents of files created or written to by a Conforming  
1132 POSIX.1 Application prior to the event are implementation-defined.

**1133 Synchronized I/O Data (and File) Integrity Completion**

1134 These terms specify that for synchronized read operations, pending writes must be successfully  
1135 completed before the read operation can complete. This is motivated by two circumstances.  
1136 Firstly, when synchronizing processes can access the same file, but not share common buffers  
1137 (such as for a remote file system), this requirement permits the reading process to guarantee that  
1138 it can read data written remotely. Secondly, having data written synchronously is insufficient to  
1139 guarantee the order with respect to a subsequent write by a reading process, and thus this extra  
1140 read semantic is necessary.

**1141 Text File**

1142 The term *text file* does not prevent the inclusion of control or other non-printable characters  
1143 (other than NUL). Therefore, standard utilities that list text files as inputs or outputs are either  
1144 able to process the special characters or they explicitly describe their limitations within their  
1145 individual descriptions. The definition of *text file* has caused controversy. The only difference  
1146 between text and binary files is that text files have lines of less than {LINE\_MAX} bytes, with no  
1147 NUL characters, each terminated by a <newline> character. The definition allows a file with a  
1148 single <newline> character, but not a totally empty file, to be called a text file. If a file ends with  
1149 an incomplete line it is not strictly a text file by this definition. The <newline> character referred  
1150 to in IEEE Std. 1003.1-200x is not some generic line separator, but a single character; files created  
1151 on systems where they use multiple characters for ends of lines are not portable to all  
1152 conforming systems without some translation process unspecified by IEEE Std. 1003.1-200x.

**1153 Thread**

1154 IEEE Std. 1003.1-200x defines a thread to be a flow of control within a process. Each thread has a  
1155 minimal amount of private state; most of the state associated with a process is shared among all  
1156 of the threads in the process. While most multi-thread extensions to POSIX have taken this  
1157 approach, others have made different decisions.

1158 **Note:** The choice to put threads within a process does not constrain implementations to  
1159 implement threads in that manner. However, all functions have to behave as though  
1160 threads share the indicated state information with the process from which they were  
1161 created.

1162 Threads need to share resources in order to cooperate. Memory has to be widely shared between  
1163 threads in order for the threads to cooperate at a fine level of granularity. Threads keep data  
1164 structures and the locks protecting those data structures in shared memory. For a data structure  
1165 to be usefully shared between threads, such structures should not refer to any data that can only  
1166 be interpreted meaningfully by a single thread. Thus, any system resources that might be  
1167 referred to in data structures need to be shared between all threads. File descriptors, path names,

1168 and pointers to stack variables are all things that programmers want to share between their  
1169 threads. Thus, the file descriptor table, the root directory, the current working directory, and the  
1170 address space have to be shared.

1171 Library implementations are possible as long as the effective behavior is as if system services  
1172 invoked by one thread do not suspend other threads. This may be difficult for some library  
1173 implementations on systems that do not provide asynchronous facilities.

1174 See Section B.2.9 (on page 3447) for additional rationale.

#### 1175 **Thread ID**

1176 See Section B.2.9.2 (on page 3463) for additional rationale.

#### 1177 **Thread-Safe Function**

1178 All functions required by IEEE Std. 1003.1-200x need to be thread-safe; see Section A.4.14 (on  
1179 page 3347) and Section B.2.9.1 (on page 3460) for additional rationale.

#### 1180 **User Database**

1181 There are no references in IEEE Std. 1003.1-200x to a *passwd file* or a *group file*, and there is no  
1182 requirement that the *group* or *passwd* databases be kept in files containing editable text. Many  
1183 large timesharing systems use *passwd* databases that are hashed for speed. Certain security  
1184 classifications prohibit certain information in the *passwd* database from being publicly readable.

1185 The term *encoded* is used instead of *encrypted* in order to avoid the implementation connotations  
1186 (such as reversibility or use of a particular algorithm) of the latter term.

1187 The *getgrent()*, *setgrent()*, *endgrent()*, *getpwent()*, *setpwent()*, and *endpwent()* functions are not  
1188 included as part of the base standard because they provide a linear database search capability  
1189 that is not generally useful (the *getpwuid()*, *getpwnam()*, *getgrgid()*, and *getgrnam()* functions are  
1190 provided for keyed lookup) and because in certain distributed systems, especially those with  
1191 different authentication domains, it may not be possible or desirable to provide an application  
1192 with the ability to browse the system databases indiscriminately. They are provided on XSI-  
1193 conformant systems due to their historical usage by many existing applications.

1194 A change from historical implementations is that the structures used by these functions have  
1195 fields of the types **gid\_t** and **uid\_t**, which are required to be defined in the **<sys/types.h>** header.  
1196 IEEE Std. 1003.1-200x requires implementations to ensure that these types are defined by  
1197 inclusion of **<grp.h>** and **<pwd.h>**, respectively, without imposing any name space pollution or  
1198 errors from redefinition of types.

1199 IEEE Std. 1003.1-200x is silent about the content of the strings containing user or group names.  
1200 These could be digit strings. IEEE Std. 1003.1-200x is also silent as to whether such digit strings  
1201 bear any relationship to the corresponding (numeric) user or group ID.

#### 1202 *Database Access*

1203 The thread-safe versions of the user and group database access functions return values in user-  
1204 supplied buffers instead of possibly using static data areas that may be overwritten by each call.



1205 **Virtual Processor\***

1206 The term *virtual processor* was chosen as a neutral term describing all kernel-level schedulable  
1207 entities, such as processes, Mach tasks, or lightweight processes. Implementing threads using  
1208 multiple processes as virtual processors, or implementing multiplexed threads above a virtual  
1209 processor layer, should be possible, provided some mechanism has also been implemented for  
1210 sharing state between processes or virtual processors. Many systems may also wish to provide  
1211 implementations of threads on systems providing “shared processes” or “variable-weight  
1212 processes”. It was felt that exposing such implementation details would severely limit the type  
1213 of systems upon which the threads interface could be supported and prevent certain types of  
1214 valid implementations. It was also determined that a virtual processor interface was out of the  
1215 scope of the Rationale (Informative) volume of IEEE Std. 1003.1-200x.

1216 **XSI**

1217 This is introduced to allow IEEE Std. 1003.1-200x to be adopted as an IEEE standard and an  
1218 Open Group Technical Standard, serving both the POSIX and the Single UNIX Specification in a  
1219 core set of volumes.

1220 The term *XSI* has been used for 10 years in connection with the XPG series and the first and  
1221 second versions of the base volumes of the Single UNIX Specification. The XSI margin code was  
1222 introduced to denote the extended or more restrictive semantics beyond POSIX that are  
1223 applicable to UNIX systems.

## 1224 **A.4 General Concepts**

### 1225 **A.4.1 Concurrent Execution**

1226 There is no additional rationale provided for this section.

### 1227 **A.4.2 Extended Security Controls**

1228 Allowing an implementation to define extended security controls enables the use of  
1229 IEEE Std. 1003.1-200x in environments that require different or more rigorous security than that  
1230 provided in POSIX.1. Extensions are allowed in two areas: privilege and file access permissions.  
1231 The semantics of these areas have been defined to permit extensions with reasonable, but not  
1232 exact, compatibility with all existing practices. For example, the elimination of the superuser  
1233 definition precludes identifying a process as privileged or not by virtue of its effective user ID.

### 1234 **A.4.3 File Access Permissions**

1235 A process should not try to anticipate the result of an attempt to access data by *a priori* use of  
1236 these rules. Rather, it should make the attempt to access data and examine the return value (and  
1237 possibly *errno* as well), or use *access()*. An implementation may include other security  
1238 mechanisms in addition to those specified in POSIX.1, and an access attempt may fail because of  
1239 those additional mechanisms, even though it would succeed according to the rules given in this  
1240 section. (For example, the user's security level might be lower than that of the object of the access  
1241 attempt.) The supplementary group IDs provide another reason for a process to not attempt to  
1242 anticipate the result of an access attempt.

### 1243 **A.4.4 File Hierarchy**

1244 Though the file hierarchy is commonly regarded to be a tree, POSIX.1 does not define it as such  
1245 for three reasons:

- 1246 1. Links may join branches.
- 1247 2. In some network implementations, there may be no single absolute root directory; see *path*  
1248 *name resolution*.
- 1249 3. With symbolic links, the file system need not be a tree or even a directed acyclic graph.

### 1250 **A.4.5 File Names**

1251 Historically, certain file names have been reserved. This list includes *core*, */etc/passwd*, and so  
1252 on. Portable applications should avoid these.

1253 Most historical implementations prohibit case folding in file names; that is, treating uppercase  
1254 and lowercase alphabetic characters as identical. However, some consider case folding desirable:

- 1255 • For user convenience
- 1256 • For ease-of-implementation of the POSIX.1 interface as a hosted system on some popular  
1257 operating systems

1258 Variants, such as maintaining case distinctions in file names, but ignoring them in comparisons,  
1259 have been suggested. Methods of allowing escaped characters of the case opposite the default  
1260 have been proposed.

1261 Many reasons have been expressed for not allowing case folding, including:

- 1262 • No solid evidence has been produced as to whether case-sensitivity or case-insensitivity is  
1263 more convenient for users.
- 1264 • Making case-insensitivity a POSIX.1 implementation option would be worse than either  
1265 having it or not having it, because:
- 1266 — More confusion would be caused among users.
- 1267 — Application developers would have to account for both cases in their code.
- 1268 — POSIX.1 implementors would still have other problems with native file systems, such as  
1269 short or otherwise constrained file names or path names, and the lack of hierarchical  
1270 directory structure.
- 1271 • Case folding is not easily defined in many European languages, both because many of them  
1272 use characters outside the USASCII alphabetic set, and because:
- 1273 — In Spanish, the digraph "ll" is considered to be a single letter, the capitalized form of  
1274 which may be either "Ll" or "LL", depending on context.
- 1275 — In French, the capitalized form of a letter with an accent may or may not retain the accent,  
1276 depending on the country in which it is written.
- 1277 — In German, the sharp ess may be represented as a single character resembling a Greek  
1278 beta (β) in lowercase, but as the digraph "SS" in uppercase.
- 1279 — In Greek, there are several lowercase forms of some letters; the one to use depends on its  
1280 position in the word. Arabic has similar rules.
- 1281 • Many East Asian languages, including Japanese, Chinese, and Korean, do not distinguish  
1282 case and are sometimes encoded in character sets that use more than one byte per character.
- 1283 • Multiple character codes may be used on the same machine simultaneously. There are  
1284 several ISO character sets for European alphabets. In Japan, several Japanese character codes  
1285 are commonly used together, sometimes even in file names; this is evidently also the case in  
1286 China. To handle case insensitivity, the kernel would have to at least be able to distinguish  
1287 for which character sets the concept made sense.
- 1288 • The file system implementation historically deals only with bytes, not with characters, except  
1289 for slash and the null byte.
- 1290 • The purpose of POSIX.1 is to standardize the common, existing definition, not to change it.  
1291 Mandating case-insensitivity would make all historical implementations non-standard.
- 1292 • Not only the interface, but also application programs would need to change, counter to the  
1293 purpose of having minimal changes to existing application code.
- 1294 • At least one of the original developers of the UNIX system has expressed objection in the  
1295 strongest terms to either requiring case-insensitivity or making it an option, mostly on the  
1296 basis that POSIX.1 should not hinder portability of application programs across related  
1297 implementations in order to allow compatibility with unrelated operating systems.
- 1298 Two proposals were entertained regarding case folding in file names:
- 1299 1. Remove all wording that previously permitted case folding.
- 1300 Rationale Case folding is inconsistent with portable file name character set definition  
1301 and file name definition (all characters except slash and null). No known  
1302 implementations allowing all characters except slash and null also do case  
1303 folding.

1304           2. Change “though this practice is not recommended:” to “although this practice is strongly  
1305 discouraged.”

1306           Rationale    If case folding must be included in POSIX.1, the wording should be stronger  
1307 to discourage the practice.

1308           The consensus selected the first proposal. Otherwise, a portable application would have to  
1309 assume that case folding would occur when it was not wanted, but that it would not occur when  
1310 it was wanted.

#### 1311 **A.4.6 File Times Update**

1312           This section reflects the actions of historical implementations. The times are not updated  
1313 immediately, but are only marked for update by the functions. An implementation may update  
1314 these times immediately.

1315           The accuracy of the time update values is intentionally left unspecified so that systems can  
1316 control the bandwidth of a possible covert channel.

1317           The wording was carefully chosen to make it clear that there is no requirement that the  
1318 conformance document contain information that might incidentally affect file update times. Any  
1319 function that performs path name resolution might update several *st\_atime* fields. Functions  
1320 such as *getpwnam()* and *getgrnam()* might update the *st\_atime* field of some specific file or files. It  
1321 is intended that these are not required to be documented in the conformance document, but they  
1322 should appear in the system documentation.

#### 1323 **A.4.7 Measurement of Execution Time**

1324           The methods used to measure the execution time of processes and threads, and the precision of  
1325 these measurements, may vary considerably depending on the software architecture of the  
1326 implementation, and on the underlying hardware. Implementations can also make tradeoffs  
1327 between the scheduling overhead and the precision of the execution time measurements.  
1328 IEEE Std. 1003.1-200x does not impose any requirement on the accuracy of the execution time; it  
1329 instead specifies that the measurement mechanism and its precision are implementation-  
1330 defined.

#### 1331 **A.4.8 Memory Synchronization**

1332           In older multi-processors, access to memory by the processors was strictly multiplexed. This  
1333 meant that a processor executing program code interrogates or modifies memory in the order  
1334 specified by the code and that all the memory operation of all the processors in the system  
1335 appear to happen in some global order, though the operation histories of different processors are  
1336 interleaved arbitrarily. The memory operations of such machines are said to be sequentially  
1337 consistent. In this environment, threads can synchronize using ordinary memory operations. For  
1338 example, a producer thread and a consumer thread can synchronize access to a circular data  
1339 buffer as follows:

```

1340 int rdptr = 0;
1341 int wrptr = 0;
1342 data_t buf[BUFSIZE];

1343 Thread 1:
1344 while (work_to_do) {
1345 int next;

1346 buf[wrptr] = produce();
1347 next = (wrptr + 1) % BUFSIZE;
1348 while (rdptr == next)
1349 ;
1350 wrptr = next;
1351 }

1352 Thread 2:
1353 while (work_to_do) {
1354 while (rdptr == wrptr)
1355 ;
1356 consume(buf[rdptr]);
1357 rdptr = (rdptr + 1) % BUFSIZE;
1358 }

```

1359 In modern multi-processors, these conditions are relaxed to achieve greater performance. If one  
1360 processor stores values in location A and then location B, then other processors loading data  
1361 from location B and then location A may see the new value of B but the old value of A. The  
1362 memory operations of such machines are said to be weakly ordered. On these machines, the  
1363 circular buffer technique shown in the example will fail because the consumer may see the new  
1364 value of *wrptr* but the old value of the data in the buffer. In such machines, synchronization can  
1365 only be achieved through the use of special instructions that enforce an order on memory  
1366 operations. Most high-level language compilers only generate ordinary memory operations to  
1367 take advantage of the increased performance. They usually cannot determine when memory  
1368 operation order is important and generate the special ordering instructions. Instead, they rely on  
1369 the programmer to use synchronization primitives correctly to ensure that modifications to a  
1370 location in memory are ordered with respect to modifications and/or access to the same location  
1371 in other threads. Access to read-only data need not be synchronized. The resulting program is  
1372 said to be data race-free.

1373 Synchronization is still important even when accessing a single primitive variable (for example,  
1374 an integer). On machines where the integer may not be aligned to the bus data width or be larger  
1375 than the data width, a single memory load may require multiple memory cycles. This means  
1376 that it may be possible for some parts of the integer to have an old value while other parts have a  
1377 newer value. On some processor architectures this cannot happen, but portable programs cannot  
1378 rely on this.

1379 In summary, a portable multi-threaded program, or a multi-process program that shares  
1380 writable memory between processes, has to use the synchronization primitives to synchronize  
1381 data access. It cannot rely on modifications to memory being observed by other threads in the  
1382 order written in the program or even on modification of a single variable being seen atomically.

1383 Conforming applications may only use the functions listed to synchronize threads of control  
1384 with respect to memory access. There are many other candidates for functions that might also be  
1385 used. Examples are: signal sending and reception, or pipe writing and reading. In general, any  
1386 function that allows one thread of control to wait for an action caused by another thread of  
1387 control is a candidate. IEEE Std. 1003.1-200x does not require these additional functions to  
1388 synchronize memory access since this would imply the following:

- 1389           • All these functions would have to be recognized by advanced compilation systems so that
- 1390           memory operations and calls to these functions are not reordered by optimization.
- 1391           • All these functions would potentially have to have memory synchronization instructions
- 1392           added, depending on the particular machine.
- 1393           • The additional functions complicate the model of how memory is synchronized and make
- 1394           automatic data race detection techniques impractical.

1395           Formal definitions of the memory model were rejected as unreadable by the vast majority of

1396           programmers. In addition, most of the formal work in the literature has concentrated on the

1397           memory as provided by the hardware as opposed to the application programmer through the

1398           compiler and runtime system. It was believed that a simple statement intuitive to most

1399           programmers would be most effective. IEEE Std. 1003.1-200x defines functions that can be used

1400           to synchronize access to memory, but it leaves open exactly how one relates those functions to

1401           the semantics of each function as specified elsewhere in IEEE Std. 1003.1-200x.

1402           IEEE Std. 1003.1-200x also does not make a formal specification of the partial ordering in time

1403           that the functions can impose, as that is implied in the description of the semantics of each

1404           function. It simply states that the programmer has to ensure that modifications do not occur

1405           “simultaneously” with other access to a memory location.

#### 1406 **A.4.9 Path Name Resolution**

1407           It is necessary to differentiate between the definition of path name and the concept of path name

1408           resolution with respect to the handling of trailing slashes. By specifying the behavior here, it is

1409           not possible to provide an implementation that is conforming but extends all interfaces that

1410           handle path names to also handle strings that are not legal path names (because they have

1411           trailing slashes).

1412           Path names that end with one or more trailing slash characters must refer to directory paths.

1413           Previous versions of IEEE Std. 1003.1-200x were not specific about the distinction between

1414           trailing slashes on files and directories, and both were permitted.

1415           Two types of implementation have been prevalent; those that ignored trailing slash characters

1416           on all path names regardless, and those that only permitted them only on existing directories.

1417           IEEE Std. 1003.1-200x requires that a path name with a trailing slash character be treated as if it

1418           had a trailing " / . " everywhere.

1419           Note that this change does not break any portable applications; since there were two different

1420           types of implementation, no application could have portably depended on either behavior. This

1421           change does however require some implementations to be altered to remain compliant.

1422           Substantial discussion over a three-year period has shown that the benefits to application

1423           developers outweighs the disadvantages for some vendors.

1424           On a historical note, some early applications automatically appended a ' / ' to every path.

1425           Rather than fix the applications, the system implementation was modified to accept this

1426           behavior by ignoring any trailing slash.

1427           Each directory has exactly one parent directory which is represented by the name **dot-dot** in the

1428           first directory. No other directory, regardless of linkages established by symbolic links, is

1429           considered the parent directory by IEEE Std. 1003.1-200x.

1430           There are two general categories of interfaces involving path name resolution: those that follow

1431           the symbolic link, and those that do not. There are several exceptions to this rule; for example,

1432           *open(path, O\_CREAT | O\_EXCL)* will fail when *path* names a symbolic link. However, in all other

1433           situations, the *open()* function will follow the link.

1434 What the file name **dot-dot** refers to relative to the root directory is implementation-defined. In  
 1435 Version 7 it refers to the root directory itself; this is the behavior mentioned in  
 1436 IEEE Std. 1003.1-200x. In some networked systems the construction `././hostname/` is used to  
 1437 refer to the root directory of another host, and POSIX.1 permits this behavior.

1438 Other networked systems use the construct `//hostname` for the same purpose; that is, a double  
 1439 initial slash is used. There is a potential problem with existing applications that create full path  
 1440 names by taking a trunk and a relative path name and making them into a single string  
 1441 separated by `'/'`, because they can accidentally create networked path names when the trunk is  
 1442 `'/'`. This practice is not prohibited because such applications can be made to conform by  
 1443 simply changing to use `"/"` as a separator instead of `'/'`:

- 1444 • If the trunk is `'/'`, the full path name will begin with `"/"` (the initial `'/'` and the  
 1445 separator `"/"`). This is the same as `'/'`, which is what is desired. (This is the general case  
 1446 of making a relative path name into an absolute one by prefixing with `"/"` instead of `'/'`.)
- 1447 • If the trunk is `"/A"`, the result is `"/A// . . ."`; since non-leading sequences of two or more  
 1448 slashes are treated as a single slash, this is equivalent to the desired `"/A/ . . ."`.
- 1449 • If the trunk is `"//A"`, the implementation-defined semantics will apply. (The multiple slash  
 1450 rule would apply.)

1451 Application developers should avoid generating path names that start with `"/"`.  
 1452 Implementations are strongly encouraged to avoid using this special interpretation since a  
 1453 number of applications currently do not follow this practice and may inadvertently generate  
 1454 `"// . . ."`.

1455 The term *root directory* is only defined in POSIX.1 relative to the process. In some  
 1456 implementations, there may be no absolute root directory. The initialization of the root directory  
 1457 of a process is implementation-defined.

#### 1458 **A.4.10 Process ID Reuse**

1459 There is no additional rationale provided for this section.

#### 1460 **A.4.11 Scheduling Policy**

1461 There is no additional rationale provided for this section.

#### 1462 **A.4.12 Seconds Since the Epoch**

1463 There is no additional rationale provided for this section.

#### 1464 **A.4.13 Semaphore**

1465 There is no additional rationale provided for this section.

#### 1466 **A.4.14 Thread-Safety**

1467 Where the interface of a function required by IEEE Std. 1003.1-200x precludes thread-safety, an  
 1468 alternate form that shall be thread-safe is provided. The names of these thread-safe forms are the  
 1469 same as the non-thread-safe forms with the addition of the suffix `"_r"`. The suffix `"_r"` is  
 1470 historical, where the `'r'` stood for "reentrant".

1471 In some cases, thread-safety is provided by restricting the arguments to an existing function.

1472 See also Section B.2.9.1 (on page 3460).

1473 **A.4.15 Utility**

1474           There is no additional rationale provided for this section.

1475 **A.4.16 Variable Assignment**

1476           There is no additional rationale provided for this section.



1477 **A.5 File Format Notation**

1478 The notation for spaces allows some flexibility for application output. Note that an empty  
1479 character position in *format* represents one or more <blank> characters on the output (not *white*  
1480 *space*, which can include <newline> characters). Therefore, another utility that reads that output  
1481 as its input must be prepared to parse the data using *scanf()*, *awk*, and so on. The ' $\Delta$ ' character  
1482 is used when exactly one <space> character is output.

1483 The treatment of integers and spaces is different from the *printf()* function in that they can be  
1484 surrounded with <blank> characters. This was done so that, given a format such as:

```
1485 "%d\n", <foo>
```

1486 the implementation could use a *printf()* call such as:

```
1487 printf("%6d\n", foo);
```

1488 and still conform. This notation is thus somewhat like *scanf()* in addition to *printf()*.

1489 The *printf()* function was chosen as a model because most of the standard developers were  
1490 familiar with it. One difference from the C function *printf()* is that the *l* and *h* conversion  
1491 characters are not used. As expressed by the Shell and Utilities volume of IEEE Std. 1003.1-200x,  
1492 there is no differentiation between decimal values for type **int**, type **long**, or type **short**. The  
1493 specifications *%d* or *%i* should be interpreted as an arbitrary length sequence of digits. Also, no  
1494 distinction is made between single precision and double precision numbers (**float** or **double** in  
1495 C). These are simply referred to as floating point numbers.

1496 Many of the output descriptions in the Shell and Utilities volume of IEEE Std. 1003.1-200x use  
1497 the term *line*, such as:

```
1498 "%s", <input line>
```

1499 Since the definition of *line* includes the trailing <newline> character already, there is no need to  
1500 include a ' $\backslash n$ ' in the format; a double <newline> character would otherwise result.

**1501 A.6 Character Set****1502 A.6.1 Portable Character Set**

1503 The portable character set is listed in full so there is no dependency on the ISO/IEC 646:1991  
1504 standard (or historically ASCII) encoded character set, although the set is identical to the  
1505 characters defined in the International Reference version of the ISO/IEC 646:1991 standard.

1506 IEEE Std. 1003.1-200x poses no requirement that multiple character sets or codesets be  
1507 supported, leaving this as a marketing differentiation for implementors. Although multiple  
1508 charmap files are supported, it is the responsibility of the implementation to provide the file(s);  
1509 if only one is provided, only that one will be accessible using the *localedef -f* option.

1510 The statement about invariance in codesets for the portable character set is worded to avoid  
1511 precluding implementations where multiple incompatible codesets are available (for instance,  
1512 ASCII and EBCDIC). The standard utilities cannot be expected to produce predictable results if  
1513 they access portable characters that vary on the same implementation.

1514 Not all character sets need include the portable character set, but each locale must include it. For  
1515 example, a Japanese-based locale might be supported by a mixture of character sets: JIS X 0201  
1516 Roman (a Japanese version of the ISO/IEC 646:1991 standard), JIS X 0208, and JIS X 0201  
1517 Katakana. Not all of these character sets include the portable characters, but at least one does  
1518 (JIS X 0201 Roman).

**1519 A.6.2 Character Encoding**

1520 Encoding mechanisms based on single shifts, such as the EUC encoding used in some Asian and  
1521 other countries, can be supported via the current charmap mechanism. With single-shift  
1522 encoding, each character is preceded by a shift code (SS2 or SS3). A complete EUC code,  
1523 consisting of the portable character set (G0) and up to three additional character sets (G1, G2,  
1524 G3), can be described using the current charmap mechanism; the encoding for each character in  
1525 additional character sets G2 and G3 must then include their single-shift code. Other mechanisms  
1526 to support locales based on encoding mechanisms such as locking shift are not addressed by this  
1527 volume of IEEE Std. 1003.1-200x.

**1528 A.6.3 C Language Wide-Character Codes**

1529 There is no additional rationale for this section.

**1530 A.6.4 Character Set Description File****1531 A.6.4.1 State-Dependent Character Encodings**

1532 A requirement was considered that would force utilities to eliminate any redundant locking  
1533 shifts, but this was left as a quality of implementation issue.

1534 This change satisfies the following requirement from the ISO POSIX-2:1993 standard, Annex  
1535 H.1:

1536            The support of state-dependent (shift encoding) character sets should be addressed fully. See  
 1537            descriptions of these in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 6.2, Character  
 1538            Encoding. If such character encodings are supported, it is expected that this will impact the Base  
 1539            Definitions volume of IEEE Std. 1003.1-200x, Section 6.2, Character Encoding, the Base Definitions  
 1540            volume of IEEE Std. 1003.1-200x, Chapter 7, Locale, the Base Definitions volume of  
 1541            IEEE Std. 1003.1-200x, Chapter 9, Regular Expressions, and the comm, cut, diff, grep, head, join,  
 1542            paste, and tail utilities.

1543            The character set description file provides:

- 1544            • The capability to describe character set attributes (such as collation order or character  
 1545            classes) independent of character set encoding, and using only the characters in the portable  
 1546            character set. This makes it possible to create generic *localedef* source files for all codesets that  
 1547            share the portable character set (such as the ISO 8859 family or IBM Extended ASCII).
- 1548            • Standardized symbolic names for all characters in the portable character set, making it  
 1549            possible to refer to any such character regardless of encoding.

1550            Implementations are free to choose their own symbolic names, as long as the names identified  
 1551            by this volume of IEEE Std. 1003.1-200x are also defined; this provides support for already  
 1552            existing “character names”.

1553            The names selected for the members of the portable character set follow the  
 1554            ISO/IEC 8859-1:1998 standard and the ISO/IEC 10646-1:1993 standard. However, several  
 1555            commonly used UNIX system names occur as synonyms in the list:

- 1556            • The historical UNIX system names are used for control characters.
- 1557            • The word “slash” is given in addition to “solidus”.
- 1558            • The word “backslash” is given in addition to “reverse-solidus”.
- 1559            • The word “hyphen” is given in addition to “hyphen-minus”.
- 1560            • The word “period” is given in addition to “full-stop”.
- 1561            • For digits, the word “digit” is eliminated.
- 1562            • For letters, the words “Latin Capital Letter” and “Latin Small Letter” are eliminated.
- 1563            • The words “left brace” and “right brace” are given in addition to “left-curly-bracket” and  
 1564            “right-curly-bracket”.
- 1565            • The names of the digits are preferred over the numbers to avoid possible confusion between  
 1566            ‘0’ and ‘o’, and between ‘1’ and ‘l’ (one and the letter ell).

1567            The names for the control characters in the Base Definitions volume of IEEE Std. 1003.1-200x,  
 1568            Chapter 6, Character Set were taken from the ISO/IEC 4873:1991 standard.

1569            The charmap file was introduced to resolve problems with the portability of, especially, *localedef*  
 1570            sources. IEEE Std. 1003.1-200x assumes that the portable character set is constant across all  
 1571            locales, but does not prohibit implementations from supporting two incompatible codings, such  
 1572            as both ASCII and EBCDIC. Such dual-support implementations should have all charmaps and  
 1573            *localedef* sources encoded using one portable character set, in effect cross-compiling for the other  
 1574            environment. Naturally, charmaps (and *localedef* sources) are only portable without  
 1575            transformation between systems using the same encodings for the portable character set. They  
 1576            can, however, be transformed between two sets using only a subset of the actual characters (the  
 1577            portable character set). However, the particular coded character set used for an application or an  
 1578            implementation does not necessarily imply different characteristics or collation; on the contrary,  
 1579            these attributes should in many cases be identical, regardless of codeset. The charmap provides

1580 the capability to define a common locale definition for multiple codesets (the same *localedef*  
1581 source can be used for codesets with different extended characters; the ability in the charmap to  
1582 define empty names allows for characters missing in certain codesets).

1583 The `<escape_char>` declaration was added at the request of the international community to ease  
1584 the creation of portable charmap files on terminals not implementing the default backslash  
1585 escape. The `<comment_char>` declaration was added at the request of the international  
1586 community to eliminate the potential confusion between the number sign and the pound sign.

1587 The octal number notation with no leading zero required was selected to match those of *awk* and  
1588 *tr* and is consistent with that used by *localedef*. To avoid confusion between an octal constant  
1589 and the back-references used in *localedef* source, the octal, hexadecimal, and decimal constants  
1590 shall contain at least two digits. As single-digit constants are relatively rare, this should not  
1591 impose any significant hardship. Provision is made for more digits to account for systems in  
1592 which the byte size is larger than 8 bits. For example, a Unicode (ISO/IEC 10646-1:1993  
1593 standard) system that has defined 16-bit bytes may require six octal, four hexadecimal, and five  
1594 decimal digits.

1595 The decimal notation is supported because some newer international standards define character  
1596 values in decimal, rather than in the old column/row notation.

1597 The charmap identifies the coded character sets supported by an implementation. At least one  
1598 charmap shall be provided, but no implementation is required to provide more than one.  
1599 Likewise, implementations can allow users to generate new charmaps (for instance, for a new  
1600 version of the ISO 8859 family of coded character sets), but does not have to do so. If users are  
1601 allowed to create new charmaps, the system documentation describes the rules that apply (for  
1602 instance, “only coded character sets that are supersets of the ISO/IEC 646:1991 standard IRV, no  
1603 multi-byte characters”).

1604 This addition of the **WIDTH** specification satisfies the following requirement from the  
1605 ISO POSIX-2:1993 standard, Annex H.1:

1606 (9) *The definition of column position relies on the implementation’s knowledge of the integral width*  
1607 *of the characters. The charmap or LC\_CTYPE locale definitions should be enhanced to allow*  
1608 *application specification of these widths.*

1609 The character “width” information was first considered for inclusion under *LC\_CTYPE* but was  
1610 moved because it is more closely associated with the information in the *charmap* than  
1611 information in the locale source (cultural conventions information). Concerns were raised that  
1612 formalizing this type of information is moving the locale source definition from the codeset-  
1613 independent entity that it was designed to be to a repository of codeset-specific information. A  
1614 similar issue occurred with the `<code_set_name>`, `<mb_cur_max>`, and `<mb_cur_min>`  
1615 information, which was resolved to reside in the *charmap* definition.

1616 The width definition was added to the IEEE P1003.2b draft standard with the intent that the  
1617 *wcswidth()* and/or *wcwidth()* functions (currently specified in the System Interfaces volume of  
1618 IEEE Std. 1003.1-200x) be the mechanism to retrieve the character width information.

## 1619 A.7 Locale

### 1620 A.7.1 General

1621 The description of locales is based on work performed in the UniForum Technical Committee  
1622 Subcommittee on Internationalization. Wherever appropriate, keywords are taken from the  
1623 ISO C standard or the X/Open Portability Guide.

1624 The value used to specify a locale with environment variables is the name specified as the *name*  
1625 operand to the *localedef* utility when the locale was created. This provides a verifiable method to  
1626 create and invoke a locale.

1627 The “object” definitions need not be portable, as long as “source” definitions are. Strictly  
1628 speaking, source definitions are portable only between implementations using the same  
1629 character set(s). Such source definitions, if they use symbolic names only, easily can be ported  
1630 between systems using different codesets, as long as the characters in the portable character set  
1631 (see the Base Definitions volume of IEEE Std. 1003.1-200x, Section 6.1, Portable Character Set )  
1632 have common values between the codesets; this is frequently the case in historical  
1633 implementations. Of source, this requires that the symbolic names used for characters outside  
1634 the portable character set be identical between character sets. The definition of symbolic names  
1635 for characters is outside the scope of IEEE Std. 1003.1-200x, but is certainly within the scope of  
1636 other standards organizations.

1637 Applications can select the desired locale by invoking the *setlocale()* function (or equivalent)  
1638 with the appropriate value. If the function is invoked with an empty string, the value of the  
1639 corresponding environment variable is used. If the environment variable is not set or is set to the  
1640 empty string, the implementation sets the appropriate environment as defined in the Base  
1641 Definitions volume of IEEE Std. 1003.1-200x, Chapter 8, Environment Variables.

### 1642 A.7.2 POSIX Locale

1643 The POSIX locale is equal to the C locale. To avoid being classified as a C-language function, the  
1644 name has been changed to the POSIX locale; the environment variable value can be either  
1645 "POSIX" or, for historical reasons, "C".

1646 The POSIX definitions mirror the historical UNIX system behavior.

1647 The use of symbolic names for characters in the tables does not imply that the POSIX locale must  
1648 be described using symbolic character names, but merely that it may be advantageous to do so.

### 1649 A.7.3 Locale Definition

1650 The decision to separate the file format from the *localedef* utility description was only partially  
1651 editorial. Implementations may provide other interfaces than *localedef*. Requirements on “the  
1652 utility”, mostly concerning error messages, are described in this way because they are meant to  
1653 affect the other interfaces implementations may provide as well as *localedef*.

1654 The text about POSIX2\_LOCALEDEF does not mean that internationalization is optional; only  
1655 that the functionality of the *localedef* utility is. REs, for instance, must still be able to recognize,  
1656 for example, character class expressions such as "[[:alpha:]]". A possible analogy is with  
1657 an applications development environment; while all conforming implementations must be  
1658 capable of executing applications, not all need to have the development environment installed.  
1659 The assumption is that the capability to modify the behavior of utilities (and applications) via  
1660 locale settings must be supported. If the *localedef* utility is not present, then the only choice is to  
1661 select an existing (presumably implementation-documented) locale. An implementation could,  
1662 for example, choose to support only the POSIX locale, which would in effect limit the amount of

1663 changes from historical implementations quite drastically. The *localedef* utility is still required,  
1664 but would always terminate with an exit code indicating that no locale could be created.  
1665 Supported locales must be documented using the syntax defined in this chapter. (This ensures  
1666 that users can accurately determine what capabilities are provided. If the implementation  
1667 decides to provide additional capabilities to the ones in this chapter, that is already provided  
1668 for.)

1669 If the option is present (that is, locales can be created), then the *localedef* utility must be capable  
1670 of creating locales based on the syntax and rules defined in this chapter. This does not mean that  
1671 the implementation cannot also provide alternate means for creating locales.

1672 The octal, decimal, and hexadecimal notations are the same employed by the charmap facility  
1673 (see the Base Definitions volume of IEEE Std. 1003.1-200x, Section 6.4, Character Set Description  
1674 File). To avoid confusion between an octal constant and a back-reference, the octal, hexadecimal,  
1675 and decimal constants must contain at least two digits. As single-digit constants are relatively  
1676 rare, this should not impose any significant hardship. Provision is made for more digits to  
1677 account for systems in which the byte size is larger than 8 bits. For example, a Unicode (see the  
1678 ISO/IEC 10646-1:1993 standard) system that has defined 16-bit bytes may require six octal, four  
1679 hexadecimal, and five decimal digits. As with the charmap file, multi-byte characters are  
1680 described in the locale definition file using “big-endian” notation for reasons of portability.  
1681 There is no requirement that the internal representation in the computer memory be in this same  
1682 order.

1683 One of the guidelines used for the development of this volume of IEEE Std. 1003.1-200x is that  
1684 characters outside the invariant part of the ISO/IEC 646:1991 standard should not be used in  
1685 portable specifications. The backslash character is not in the invariant part; the number sign is,  
1686 but with multiple representations: as a number sign, and as a pound sign. As far as general  
1687 usage of these symbols, they are covered by the “grandfather clause”, but for newly defined  
1688 interfaces, the WG15 POSIX working group has requested that POSIX provide alternate  
1689 representations. Consequently, while the default escape character remains the backslash and the  
1690 default comment character is the number sign, implementations are required to recognize  
1691 alternative representations, identified in the applicable source file via the `<escape_char>` and  
1692 `<comment_char>` keywords.

### 1693 A.7.3.1 *LC\_CTYPE*

1694 The *LC\_CTYPE* category is primarily used to define the encoding-independent aspects of a  
1695 character set, such as character classification. In addition, certain encoding-dependent  
1696 characteristics are also defined for an application via the *LC\_CTYPE* category.  
1697 IEEE Std. 1003.1-200x does not mandate that the encoding used in the locale is the same as the  
1698 one used by the application because an implementation may decide that it is advantageous to  
1699 define locales in a system-wide encoding rather than having multiple, logically identical locales  
1700 in different encodings, and to convert from the application encoding to the system-wide  
1701 encoding on usage. Other implementations could require encoding-dependent locales.

1702 In either case, the *LC\_CTYPE* attributes that are directly dependent on the encoding, such as  
1703 `<mb_cur_max>` and the display width of characters, are not user-specifiable in a locale source  
1704 and are consequently not defined as keywords.

1705 Implementations may define additional keywords or extend the *LC\_CTYPE* mechanism to allow  
1706 application-defined keywords.

1707 The text “The ellipsis specification shall only be valid within a single encoded character set” is  
1708 present because it is possible to have a locale supported by multiple character encodings, as  
1709 explained in the rationale for the Base Definitions volume of IEEE Std. 1003.1-200x, Section 6.1,  
1710 Portable Character Set. An example given there is of a possible Japanese-based locale supported

1711 by a mixture of the character sets JIS X 0201 Roman, JIS X 0208, and JIS X 0201 Katakana.  
 1712 Attempting to express a range of characters across these sets is not logical and the  
 1713 implementation is free to reject such attempts.

1714 As the `LC_CTYPE` character classes are based on the ISO C standard character class definition,  
 1715 the category does not support multi-character elements. For instance, the German character  
 1716 <sharp-s> is traditionally classified as a lowercase letter. There is no corresponding uppercase  
 1717 letter; in proper capitalization of German text, the <sharp-s> will be replaced by "SS"; that is, by  
 1718 two characters. This kind of conversion is outside the scope of the **toupper** and **tolower**  
 1719 keywords.

1720 Where IEEE Std. 1003.1-200x specifies that only certain characters can be specified, as for the  
 1721 keywords **digit** and **xdigit**, the specified characters shall be from the portable character set, as  
 1722 shown. As an example, only the Arabic digits 0 through 9 are acceptable as digits.

1723 The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included  
 1724 characters. These only need to be specified if the character values (that is, encoding) differs from  
 1725 the implementation default values. It is not possible to define a locale without these  
 1726 automatically included characters unless some implementation extension is used to prevent  
 1727 their inclusion. Such a definition would not be a proper superset of the C locale, and thus, it  
 1728 might not be possible for the standard utilities to be implemented as programs conforming to  
 1729 the ISO C standard.

1730 The definition of character class **digit** requires that only ten characters—the ones defining  
 1731 digits—can be specified; alternate digits (for example, Hindi or Kanji) cannot be specified here.  
 1732 However, the encoding may vary if an implementation supports more than one encoding.

1733 The definition of character class **xdigit** requires that the characters included in character class  
 1734 **digit** are included here also and allows for different symbols for the hexadecimal digits 10  
 1735 through 15.

1736 The inclusion of the **charclass** keyword satisfies the following requirement from the  
 1737 ISO POSIX-2: 1993 standard, Annex H.1:

1738 (3) *The `LC_CTYPE` (2.5.2.1) locale definition should be enhanced to allow user-specified additional*  
 1739 *character classes, similar in concept to the ISO C standard Multibyte Support Extension (MSE)*  
 1740 *is\_wctype() function.*

1741 This keyword was previously included in The Open Group specifications and is now mandated  
 1742 in the Shell and Utilities volume of IEEE Std. 1003.1-200x.

1743 The symbolic constant `{CHARCLASS_NAME_MAX}` was also adopted from The Open Group  
 1744 specifications. Application portability is enhanced by the use of symbolic constants.

#### 1745 A.7.3.2 `LC_COLLATE`

1746 The rules governing collation depend to some extent on the use. At least five different levels of  
 1747 increasingly complex collation rules can be distinguished:

- 1748 1. *Byte/machine code order*: This is the historical collation order in the UNIX system and many  
 1749 proprietary operating systems. Collation is here performed character by character, without  
 1750 any regard to context. The primary virtue is that it usually is quite fast and also  
 1751 completely deterministic; it works well when the native machine collation sequence  
 1752 matches the user expectations.
- 1753 2. *Character order*: On this level, collation is also performed character by character, without  
 1754 regard to context. The order between characters is, however, not determined by the code  
 1755 values, but on the expectations by the user of the “correct” order between characters. In

1756 addition, such a (simple) collation order can specify that certain characters collate equally  
1757 (for example, uppercase and lowercase letters).

1758 3. *String ordering*: On this level, entire strings are compared based on relatively  
1759 straightforward rules. Several “passes” may be required to determine the order between  
1760 two strings. Characters may be ignored in some passes, but not in others; the strings may  
1761 be compared in different directions; and simple string substitutions may be performed  
1762 before strings are compared. This level is best described as “dictionary” ordering; it is  
1763 based on the spelling, not the pronunciation, or meaning, of the words.

1764 4. *Text search ordering*: This is a further refinement of the previous level, best described as  
1765 “telephone book ordering”; some common homonyms (words spelled differently but with  
1766 the same pronunciation) are collated together; numbers are collated as if they were spelled  
1767 out, and so on.

1768 5. *Semantic-level ordering*: Words and strings are collated based on their meaning; entire words  
1769 (such as “the”) are eliminated; the ordering is not deterministic. This usually requires  
1770 special software and is highly dependent on the intended use.

1771 While the historical collation order formally is at level 1, for the English language it corresponds  
1772 roughly to elements at level 2. The user expects to see the output from the *ls* utility sorted very  
1773 much as it would be in a dictionary. While telephone book ordering would be an optimal goal  
1774 for standard collation, this was ruled out as the order would be language-dependent.  
1775 Furthermore, a requirement was that the order must be determined solely from the text string  
1776 and the collation rules; no external information (for example, “pronunciation dictionaries”)   
1777 could be required.

1778 As a result, the goal for the collation support is at level 3. This also matches the requirements for  
1779 the Canadian collation order, as well as other, known collation requirements for alphabetic  
1780 scripts. It specifically rules out collation based on pronunciation rules or based on semantic  
1781 analysis of the text.

1782 The syntax for the *LC\_COLLATE* category source meets the requirements for level 3 and has  
1783 been verified to produce the correct result with examples based on French, Canadian, and  
1784 Danish collation order. Because it supports multi-character collating elements, it is also capable  
1785 of supporting collation in codesets where a character is expressed using non-spacing characters  
1786 followed by the base character (such as the ISO/IEC 6937: 1994 standard).

1787 The directives that can be specified in an operand to the **order\_start** keyword are based on the  
1788 requirements specified in several proposed standards and in customary use. The following is a  
1789 rephrasing of rules defined for “lexical ordering in English and French” by the Canadian  
1790 Standards Association (the text in square brackets is rephrased):

- 1791 • Once special characters [punctuation] have been removed from original strings, the ordering  
1792 is determined by scanning forwards (left to right) [disregarding case and diacriticals].
- 1793 • In case of equivalence, special characters are once again removed from original strings and  
1794 the ordering is determined by scanning backwards (starting from the rightmost character of  
1795 the string and back), character by character [disregarding case but considering diacriticals].
- 1796 • In case of repeated equivalence, special characters are removed again from original strings  
1797 and the ordering is determined by scanning forwards, character by character [considering  
1798 both case and diacriticals].
- 1799 • If there is still an ordering equivalence after the first three rules have been applied, then only  
1800 special characters and the position they occupy in the string are considered to determine  
1801 ordering. The string that has a special character in the lowest position comes first. If two  
1802 strings have a special character in the same position, the character [with the lowest collation



1803 value] comes first. In case of equality, the other special characters are considered until there  
1804 is a difference or until all special characters have been exhausted.

1805 It is estimated that this part of IEEE Std. 1003.1-200x covers the requirements for all European  
1806 languages, and no particular problems are anticipated with Slavic or Middle East character sets.

1807 The Far East (particularly Japanese/Chinese) collations are often based on contextual  
1808 information and pronunciation rules (the same ideogram can have different meanings and  
1809 different pronunciations). Such collation, in general, falls outside the desired goal of  
1810 IEEE Std. 1003.1-200x. There are, however, several other collation rules (stroke/radical or “most  
1811 common pronunciation”) that can be supported with the mechanism described here.

1812 The character (and collating element) order is defined by the order in which characters and  
1813 elements are specified between the **order\_start** and **order\_end** keywords. This character order is  
1814 used in range expressions in REs (see the Base Definitions volume of IEEE Std. 1003.1-200x,  
1815 Chapter 9, Regular Expressions). Weights assigned to the characters and elements define the  
1816 collation sequence; in the absence of weights, the character order is also the collation sequence.

#### 1817 A.7.3.3 *LC\_MONETARY*

1818 The currency symbol does not appear in *LC\_MONETARY* because it is not defined in the C locale  
1819 of the ISO C standard.

1820 The ISO C standard limits the size of decimal points and thousands delimiters to single-byte  
1821 values. In locales based on multi-byte coded character sets, this cannot be enforced;  
1822 IEEE Std. 1003.1-200x does not prohibit such characters, but makes the behavior unspecified (in  
1823 the text “In contexts where other standards ...”).

1824 The grouping specification is based on, but not identical to, the ISO C standard. The -1 signals  
1825 that no further grouping shall be performed; the equivalent of {CHAR\_MAX} in the ISO C  
1826 standard.

1827 The text “the value is not available in the locale” is taken from the ISO C standard and is used  
1828 instead of the “unspecified” text in early proposals. There is no implication that omitting these  
1829 keywords or assigning them values of " " or -1 produces unspecified results; such omissions or  
1830 assignments eliminate the effects described for the keyword or produce zero-length strings, as  
1831 appropriate.

1832 The locale definition is an extension of the ISO C standard *localeconv()* specification. In  
1833 particular, rules on how **currency\_symbol** is treated are extended to also cover **int\_curr\_symbol**,  
1834 and **p\_set\_by\_space** and **n\_sep\_by\_space** have been augmented with the value 2, which places  
1835 a <space> between the sign and the symbol (if they are adjacent; otherwise, it should be treated  
1836 as a 0).

#### 1837 A.7.3.4 *LC\_NUMERIC*

1838 See the rationale for *LC\_MONETARY* for a description of the behavior of grouping.

#### 1839 A.7.3.5 *LC\_TIME*

1840 Although certain of the field descriptors in the POSIX locale (such as the name of the month) are  
1841 shown with initial capital letters, this need not be the case in other locales. Programs using these  
1842 fields may need to adjust the capitalization if the output is going to be used at the beginning of a  
1843 sentence.

1844 The *LC\_TIME* descriptions of **abday**, **day**, **mon**, and **abmon** imply a Gregorian style calendar (7-  
1845 day weeks, 12-month years, leap years, and so on). Formatting time strings for other types of  
1846 calendars is outside the scope of IEEE Std. 1003.1-200x.

1847 While the ISO 8601:1988 standard numbers the weekdays starting with Monday, historical  
 1848 practice is to use the Sunday as the first day. Rather than change the order and introduce  
 1849 potential confusion, the days must be specified beginning with Sunday; previous references to  
 1850 “first day” have been removed. Note also that the Shell and Utilities volume of  
 1851 IEEE Std. 1003.1-200x *date* utility supports numbering compliant with the ISO 8601:1988  
 1852 standard.

1853 As specified under *date* in the Shell and Utilities volume of IEEE Std. 1003.1-200x and *strftime()*  
 1854 in the System Interfaces volume of IEEE Std. 1003.1-200x, the field descriptors corresponding to  
 1855 the optional keywords consist of a modifier followed by a traditional field descriptor (for  
 1856 instance %Ex). If the optional keywords are not supported by the implementation or are  
 1857 unspecified for the current locale, these field descriptors are treated as the traditional field  
 1858 descriptor. For example, assume the following keywords:

```
1859 alt_digits "0th";"1st";"2nd";"3rd";"4th";"5th";\
1860 "6th";"7th";"8th";"9th";"10th"
```

```
1861 d_fmt "The %Od day of %B in %Y"
```

1862 On 7/4/1776, the %x field descriptor would result in "The 4th day of July in 1776",  
 1863 while on 7/14/1789 would result in "The 14 day of July in 1789". It can be noted that  
 1864 the above example is for illustrative purposes only; the %O modifier is primarily intended to  
 1865 provide for Kanji or Hindi digits in *date* formats.

1866 A.7.3.6 *LC\_MESSAGES*

## 1867 A.7.4 Locale Definition Grammar

1868 There is no additional rationale for this section.

### 1869 A.7.4.1 *Locale Lexical Conventions*

1870 There is no additional rationale for this section.

### 1871 A.7.4.2 *Locale Grammar*

1872 There is no additional rationale for this section.

## 1873 A.7.5 Locale Definition Example

1874 There is no additional rationale for this section.

1875 **A.8 Environment Variables**1876 **A.8.1 Environment Variable Definition**

1877 The variable *environ* is not intended to be declared in any header, but rather to be declared by the  
 1878 user for accessing the array of strings that is the environment. This is the traditional usage of the  
 1879 symbol. Putting it into a header could break some programs that use the symbol for their own  
 1880 purposes.

1881 The decision to restrict conforming systems to the use of digits, uppercase letters, and  
 1882 underscores for environment variable names allows applications to use lowercase letters in their  
 1883 environment variable names without conflicting with any conforming system.

1884 **A.8.2 Internationalization Variables**

1885 The text about locale implies that any utilities written in standard C and conforming to  
 1886 IEEE Std. 1003.1-200x must issue the following call:

```
1887 setlocale(LC_ALL, "")
```

1888 If this were omitted, the ISO C standard specifies that the C locale would be used.

1889 If any of the environment variables are invalid, it makes sense to default to an implementation-  
 1890 defined, consistent locale environment. It is more confusing for a user to have partial settings  
 1891 occur in case of a mistake. All utilities would then behave in one language/cultural  
 1892 environment. Furthermore, it provides a way of forcing the whole environment to be the  
 1893 implementation-defined default. Disastrous results could occur if a pipeline of utilities partially  
 1894 uses the environment variables in different ways. In this case, it would be appropriate for  
 1895 utilities that use *LANG* and related variables to exit with an error if any of the variables are  
 1896 invalid. For example, users typing individual commands at a terminal might want *date* to work if  
 1897 *LC\_MONETARY* is invalid as long as *LC\_TIME* is valid. Since these are conflicting reasonable  
 1898 alternatives, IEEE Std. 1003.1-200x leaves the results unspecified if the locale environment  
 1899 variables would not produce a complete locale matching the specification of the user.

1900 The locale settings of individual categories cannot be truly independent and still guarantee  
 1901 correct results. For example, when collating two strings, characters must first be extracted from  
 1902 each string (governed by *LC\_CTYPE*) before being mapped to collating elements (governed by  
 1903 *LC\_COLLATE*) for comparison. That is, if *LC\_CTYPE* is causing parsing according to the rules of  
 1904 a large, multi-byte code set (potentially returning 20 000 or more distinct character codeset  
 1905 values), but *LC\_COLLATE* is set to handle only an 8-bit codeset with 256 distinct characters,  
 1906 meaningful results are obviously impossible.

1907 The *LC\_MESSAGES* variable affects the language of messages generated by the standard  
 1908 utilities.

1909 The description of the environment variable names starting with the characters “LC\_”  
 1910 acknowledges the fact that the interfaces presented may be extended as new international  
 1911 functionality is required. In the ISO C standard, names preceded by “LC\_” are reserved in the  
 1912 name space for future categories.

1913 To avoid name clashes, new categories and environment variables are divided into two  
 1914 classifications: *implementation-independent* and *implementation-defined*.

1915 Implementation-independent names will have the following format:

```
1916 LC_NAME
```

1917 where *NAME* is the name of the new category and environment variable. Capital letters must be  
 1918 used for implementation-independent names.

1919 Implementation-defined names must be in lowercase letters, as below:

1920 `LC_name`

### 1921 **A.8.3 Other Environment Variables**

1922 The quoted form of the timezone variable allows timezone names of the form UTC+1 (or any  
 1923 name that contains the character plus ('+'), the character minus ('-'), or digits), which may be  
 1924 appropriate for countries that do not have an official timezone name. It would be coded as  
 1925 <UTC+1>+1<UTC+2>, which would cause *std* to have a value of UTC+1 and *dst* a value of  
 1926 UTC+2, each with a length of 6 characters. This does not appear to conflict with any existing  
 1927 usage. The characters '<' and '>' were chosen for quoting because they are easier to parse  
 1928 visually than a quoting character that does not provide some sense of bracketing (and in a string  
 1929 like this, such bracketing is helpful). They were also chosen because they do not need special  
 1930 treatment when assigning to the *TZ* variable. Users are often confused by embedding quotes in a  
 1931 string. Because '<' and '>' are meaningful to the shell, the whole string would have to be  
 1932 quoted, but that is easily explained. (Parentheses would have presented the same problems.)  
 1933 Although the '>' symbol could have been permitted in the string by either escaping it or  
 1934 doubling it, it seemed of little value to require that. This could be provided as an extension if  
 1935 there was a need. Timezone names of this new form lead to a requirement that the value of  
 1936 `{_POSIX_TZNAME_MAX}` change from 3 to 6.

### 1937 **COLUMNS, LINES**

1938 The default value for the number of column positions, *COLUMNS*, and screen height, *LINES*, are  
 1939 unspecified because historical implementations use different methods to determine values  
 1940 corresponding to the size of the screen in which the utility is run. This size is typically known to  
 1941 the implementation through the value of *TERM*, or by more elaborate methods such as  
 1942 extensions to the *stty* utility or knowledge of how the user is dynamically resizing windows on a  
 1943 bit-mapped display terminal. Users should not need to set these variables in the environment  
 1944 unless there is a specific reason to override the default behavior of the implementation, such as  
 1945 to display data in an area arbitrarily smaller than the terminal or window. Values for these  
 1946 variables that are not decimal integers greater than zero are implicitly undefined values; it is  
 1947 unnecessary to enumerate all of the possible values outside of the acceptable set.

### 1948 **PATH**

1949 Many historical implementations of the Bourne shell do not interpret a trailing colon to represent  
 1950 the current working directory and are thus non-conforming. The C Shell and the KornShell  
 1951 conform to IEEE Std. 1003.1-200x on this point. The usual name of dot may also be used to refer  
 1952 to the current working directory.

1953 Many implementations historically have used a default value of `/bin` and `/usr/bin` for the *PATH*  
 1954 variable. IEEE Std. 1003.1-200x does not mandate this default path be identical to that retrieved  
 1955 from `getconf _CS_PATH` because it is likely that the standardized utilities may be provided in  
 1956 another directory separate from the directories used by some historical applications.

1957 **LOGNAME**

1958 In most implementations, the value of such a variable is easily forged, so security-critical  
1959 applications should rely on other means of determining user identity. *LOGNAME* is required to  
1960 be constructed from the portable file name character set for reasons of interchange. No  
1961 diagnostic condition is specified for violating this rule, and no requirement for enforcement  
1962 exists. The intent of the requirement is that if extended characters are used, the “guarantee” of  
1963 portability implied by a standard is void.

1964 **SHELL**

1965 The *SHELL* variable names the preferred shell of the user; it is a guide to applications. There is  
1966 no direct requirement that that shell conform to IEEE Std. 1003.1-200x; that decision should rest  
1967 with the user. It is the intention of the standard developers that alternative shells be permitted, if  
1968 the user chooses to develop or acquire one. An operating system that builds its shell into the  
1969 “kernel” in such a manner that alternative shells would be impossible does not conform to the  
1970 spirit of IEEE Std. 1003.1-200x.

1971 **CHANGE HISTORY**1972 **Issue 6**

1973 Changed format of *TZ* field to allow for the quoted form as defined in previous  
1974 versions of the ISO POSIX-1 standard.

1975 **A.9 Regular Expressions**

1976 Rather than repeating the description of REs for each utility supporting REs, the standard  
 1977 developers preferred a common, comprehensive description of regular expressions in one place.  
 1978 The most common behavior is described here, and exceptions or extensions to this are  
 1979 documented for the respective utilities, as appropriate.

1980 The BRE corresponds to the *ed* or historical *grep* type, and the ERE corresponds to the historical  
 1981 *egrep* type (now *grep -E*).

1982 The text is based on the *ed* description and substantially modified, primarily to aid developers  
 1983 and others in the understanding of the capabilities and limitations of REs. Much of this was  
 1984 influenced by internationalization requirements.

1985 It should be noted that the definitions in this section do not cover the *tr* utility; the *tr* syntax does  
 1986 not employ REs.

1987 The specification of REs is particularly important to internationalization because pattern  
 1988 matching operations are very basic operations in business and other operations. The syntax and  
 1989 rules of REs are intended to be as intuitive as possible to make them easy to understand and use.  
 1990 The historical rules and behavior do not provide that capability to non-English language users,  
 1991 and do not provide the necessary support for commonly used characters and language  
 1992 constructs. It was necessary to provide extensions to the historical RE syntax and rules to  
 1993 accommodate other languages.

1994 As they are limited to bracket expressions, the rationale for these modifications is in the Base  
 1995 Definitions volume of IEEE Std. 1003.1-200x, Section 9.3.5, RE Bracket Expression.

1996 **A.9.1 Regular Expression Definitions**

1997 It is possible to determine what strings correspond to subexpressions by recursively applying  
 1998 the leftmost longest rule to each subexpression, but only with the proviso that the overall match  
 1999 is leftmost longest. For example, matching "`\(ac*\\)c*d[ac]*\1`" against *acdacaaa* matches  
 2000 *acdacaaa* (with `\1=a`); simply matching the longest match for "`\(ac*\\)`" would yield `\1=ac`, but  
 2001 the overall match would be smaller (*acdac*). Conceptually, the implementation must examine  
 2002 every possible match and among those that yield the leftmost longest total matches, pick the one  
 2003 that does the longest match for the leftmost subexpression, and so on. Note that this means that  
 2004 matching by subexpressions is context-dependent: a subexpression within a larger RE may  
 2005 match a different string from the one it would match as an independent RE, and two instances of  
 2006 the same subexpression within the same larger RE may match different lengths even in similar  
 2007 sequences of characters. For example, in the ERE "`(a.*b)(a.*b)`", the two identical  
 2008 subexpressions would match four and six characters, respectively, of *accbacccb*.

2009 The definition of *single character* has been expanded to include also collating elements consisting  
 2010 of two or more characters; this expansion is applicable only when a bracket expression is  
 2011 included in the BRE or ERE. An example of such a collating element may be the Dutch *ij*, which  
 2012 collates as a 'y'. In some encodings, a ligature "i with j" exists as a character and would  
 2013 represent a single-character collating element. In another encoding, no such ligature exists, and  
 2014 the two-character sequence *ij* is defined as a multi-character collating element. Outside brackets,  
 2015 the *ij* is treated as a two-character RE and matches the same characters in a string. Historically, a  
 2016 bracket expression only matched a single character. If, however, the bracket expression defines,  
 2017 for example, a range that includes *ij*, then this particular bracket expression also matches a  
 2018 sequence of the two characters 'i' and 'j' in the string.

**2019 A.9.2 Regular Expression General Requirements**

2020 The definition of which sequence is matched when several are possible is based on the leftmost-  
2021 longest rule historically used by deterministic recognizers. This rule is easier to define and  
2022 describe, and arguably more useful, than the first-match rule historically used by non-  
2023 deterministic recognizers. It is thought that dependencies on the choice of rule are rare; carefully  
2024 contrived examples are needed to demonstrate the difference.

2025 A formal expression of the leftmost-longest rule is:

2026       The search is performed as if all possible suffixes of the string were tested for a prefix  
2027       matching the pattern; the longest suffix containing a matching prefix is chosen, and the  
2028       longest possible matching prefix of the chosen suffix is identified as the matching sequence.

2029 Historically, most RE implementations only match lines, not strings. However, that is more an  
2030 effect of the usage than of an inherent feature of REs themselves. Consequently, IEEE Std. 1003.1-200x  
2031 does not regard <newline>s as special; they are ordinary characters, and both a period and a non-matching  
2032 list can match them. Those utilities (like *grep*) that do not allow <newline>s to match are responsible  
2033 for eliminating any <newline> from strings before matching against the RE. The *regcomp()* function,  
2034 however, can provide support for such processing without violating the rules of this section.  
2035

2036 The definition of case-insensitive processing is intended to allow matching of multi-character  
2037 collating elements as well as characters. For instance, as each character in the string is matched  
2038 using both its cases, the RE "[ [ . Ch . ] ]", when matched against "char", is in reality matched  
2039 against "ch", "Ch", "cH", and "CH".

2040 Some implementations of *egrep* have had very limited flexibility in handling complex EREs. IEEE Std.  
2041 1003.1-200x does not attempt to define the complexity of a BRE or ERE, but does place a lower limit  
2042 on it—any RE must be handled, as long as it can be expressed in 256 bytes or less. (Of course, this  
2043 does not place an upper limit on the implementation.) There are historical programs using a non-  
2044 deterministic-recognizer implementation that should have no difficulty with this limit. It is possible  
2045 that a good approach would be to attempt to use the faster, but more limited, deterministic  
2046 recognizer for simple expressions and to fall back on the non-deterministic recognizer for those  
2047 expressions requiring it. Non-deterministic implementations must be careful to observe the rules on  
2048 which match is chosen; the longest match, not the first match, starting at a given character is used.  
2049

2050 The term *invalid* highlights a difference between this section and some others: IEEE Std. 1003.1-200x  
2051 frequently avoids mandating of errors for syntax violations because they can be used by implementors  
2052 to trigger extensions. However, the authors of the internationalization features of REs wanted to  
2053 mandate errors for certain conditions to identify usage problems or non-portable constructs. These  
2054 are identified within this rationale as appropriate. The remaining syntax violations have been left  
2055 implicitly or explicitly undefined. For example, the BRE construct "\{1,2,3\}" does not comply with  
2056 the grammar. A conforming application cannot rely on it producing an error nor matching the literal  
2057 characters "\{1,2,3\}". The term “undefined” was used in favor of “unspecified” because many of  
2058 the situations are considered errors on some implementations, and the standard developers  
2059 considered that consistency throughout the section was preferable to mixing undefined and  
2060 unspecified.  
2061

2062 **A.9.3 Basic Regular Expressions**

2063 There is no additional rationale for this section.

2064 **A.9.3.1 BREs Matching a Single Character or Collating Element**

2065 There is no additional rationale for this section.

2066 **A.9.3.2 BRE Ordinary Characters**

2067 There is no additional rationale for this section.

2068 **A.9.3.3 BRE Special Characters**

2069 There is no additional rationale for this section.

2070 **A.9.3.4 Periods in BREs**

2071 There is no additional rationale for this section.

2072 **A.9.3.5 RE Bracket Expression**

2073 Range expressions are, historically, an integral part of REs. However, the requirements of  
 2074 “natural language behavior” and portability do conflict: ranges must be treated according to the  
 2075 current collating sequence and include such characters that fall within the range based on that  
 2076 collating sequence, regardless of character values. This means, however, that the interpretation  
 2077 will differ depending on collating sequence. If, for instance, one collating sequence defines ‘a’ as  
 2078 a variant of ‘a’, while another defines it as a letter following ‘z’, then the expression “[a-z]”  
 2079 is valid in the first language and invalid in the second. This kind of ambiguity should be avoided  
 2080 in portable applications, and therefore the standard developers elected to state that ranges must  
 2081 not be used in strictly conforming applications; however, implementations must support them.

2082 Some historical implementations allow range expressions where the ending range point of one  
 2083 range is also the starting point of the next (for instance, “[a-m-o]”). This behavior should not  
 2084 be permitted, but to avoid breaking historical implementations, it is now *undefined* whether it is a  
 2085 valid expression and how it should be interpreted.

2086 Current practice in *awk* and *lex* is to accept escape sequences in bracket expressions as per the  
 2087 Base Definitions volume of IEEE Std. 1003.1-200x, Table 5-1, Escape Sequences and Associated  
 2088 Actions, while the normal ERE behavior is to regard such a sequence as consisting of two  
 2089 characters. Allowing the *awk/lex* behavior in EREs would change the normal behavior in an  
 2090 unacceptable way; it is expected that *awk* and *lex* will decode escape sequences in EREs before  
 2091 passing them to *regcomp()* or comparable routines. Each utility describes the escape sequences it  
 2092 accepts as an exception to the rules in this section; the list is not the same, for historical reasons.

2093 As noted previously, the new syntax and rules have been added to accommodate other  
 2094 languages than English. The remainder of this section describes the rationale for these  
 2095 modifications.

2096 **A.9.3.6 BREs Matching Multiple Characters**

2097 The limit of nine back-references to subexpressions in the RE is based on the use of a single-digit  
 2098 identifier; increasing this to multiple digits would break historical applications. This does not  
 2099 imply that only nine subexpressions are allowed in REs. The following is a valid BRE with ten  
 2100 subexpressions:

2101 `\\(\\(\\(ab\\)*c\\)*d\\)\\(ef\\)*\\(gh\\){2}\\(ij\\)*\\(kl\\)*\\(mn\\)*\\(op\\)*\\(qr\\)*`



2102 The standard developers regarded the common historical behavior, which supported "\n\*", but  
 2103 not "\n\{min,max\}", "\(...\)\*", or "\(...\)\{min,max\}", as a non-intentional  
 2104 result of a specific implementation, and they supported both duplication and interval  
 2105 expressions following subexpressions and back-references.

2106 The changes to the processing of the back-reference expression remove an unspecified or  
 2107 ambiguous behavior in the Shell and Utilities volume of IEEE Std. 1003.1-200x, aligning it with  
 2108 the requirements specified for the *regcomp()* expression, and is the result of PASC Interpretation  
 2109 1003.2-92 #43 submitted for the ISO POSIX-2: 1993 standard.

#### 2110 A.9.3.7 BRE Precedence

2111 There is no additional rationale for this section.

#### 2112 A.9.3.8 BRE Expression Anchoring

2113 Often, the dollar sign is viewed as matching the ending <newline> in text files. This is not  
 2114 strictly true; the <newline> is typically eliminated from the strings to be matched, and the dollar  
 2115 sign matches the terminating null character.

2116 The ability of '^', '\$', and '\*' to be non-special in certain circumstances may be confusing to  
 2117 some programmers, but this situation was changed only in a minor way from historical practice  
 2118 to avoid breaking many historical scripts. Some consideration was given to making the use of  
 2119 the anchoring characters undefined if not escaped and not at the beginning or end of strings.  
 2120 This would cause a number of historical BREs, such as "2^10", "\$HOME", and "\$1.35", that  
 2121 relied on the characters being treated literally, to become invalid.

2122 However, one relatively uncommon case was changed to allow an extension used on some  
 2123 implementations. Historically, the BREs "^foo" and "\(^foo\)" did not match the same  
 2124 string, despite the general rule that subexpressions and entire BREs match the same strings. To  
 2125 increase consensus, IEEE Std. 1003.1-200x has allowed an extension on some systems to treat  
 2126 these two cases in the same way by declaring that anchoring *may* occur at the beginning or end  
 2127 of a subexpression. Therefore, portable BREs that require a literal circumflex at the beginning or  
 2128 a dollar sign at the end of a subexpression must escape them. Note that a BRE such as  
 2129 "a\(^bc\)" will either match "a^bc" or nothing on different systems under the rules.

2130 ERE anchoring has been different from BRE anchoring in all historical systems. An unescaped  
 2131 anchor character has never matched its literal counterpart outside a bracket expression. Some  
 2132 systems treated "foo\$bar" as a valid expression that never matched anything; others treated it  
 2133 as invalid. IEEE Std. 1003.1-200x mandates the former, valid unmatched behavior.

2134 Some systems have extended the BRE syntax to add alternation. For example, the subexpression  
 2135 "\(foo\$|bar\)" would match either "foo" at the end of the string or "bar" anywhere. The  
 2136 extension is triggered by the use of the undefined "\|" sequence. Because the BRE is undefined  
 2137 for portable scripts, the extending system is free to make other assumptions, such that the '\$'  
 2138 represents the end-of-line anchor in the middle of a subexpression. If it were not for the  
 2139 extension, the '\$' would match a literal dollar sign under the rules.

**2140 A.9.4 Extended Regular Expressions**

2141 As with BREs, the standard developers decided to make the interpretation of escaped ordinary  
2142 characters undefined.

2143 The right parenthesis is not listed as an ERE special character because it is only special in the  
2144 context of a preceding left parenthesis. If found without a preceding left parenthesis, the right  
2145 parenthesis has no special meaning.

2146 The *interval expression*, " $\{m,n\}$ ", has been added to EREs. Historically, the interval expression  
2147 has only been supported in some ERE implementations. The standard developers estimated that  
2148 the addition of interval expressions to EREs would not decrease consensus and would also make  
2149 BREs more of a subset of EREs than in many historical implementations.

2150 It was suggested that, in addition to interval expressions, back-references ( $\backslash n$ ) should also be  
2151 added to EREs. This was rejected by the standard developers as likely to decrease consensus.

2152 In historical implementations, multiple duplication symbols are usually interpreted from left to  
2153 right and treated as additive. As an example, " $a^*b$ " matches zero or more instances of 'a'  
2154 followed by a 'b'. In IEEE Std. 1003.1-200x, multiple duplication symbols are undefined; that is,  
2155 they cannot be relied upon for portable applications. One reason for this is to provide some  
2156 scope for future enhancements.

2157 The precedence of operations differs between EREs and those in *lex*; in *lex*, for historical reasons,  
2158 interval expressions have a lower precedence than concatenation.

**2159 A.9.4.1 EREs Matching a Single Character or Collating Element**

2160 There is no additional rationale for this section.

**2161 A.9.4.2 ERE Ordinary Characters**

2162 There is no additional rationale for this section.

**2163 A.9.4.3 ERE Special Characters**

2164 There is no additional rationale for this section.

**2165 A.9.4.4 Periods in EREs**

2166 There is no additional rationale for this section.

**2167 A.9.4.5 ERE Bracket Expression**

2168 There is no additional rationale for this section.

**2169 A.9.4.6 EREs Matching Multiple Characters**

2170 There is no additional rationale for this section.

**2171 A.9.4.7 ERE Alternation**

2172 There is no additional rationale for this section.

2173 A.9.4.8 *ERE Precedence*

2174 There is no additional rationale for this section.

2175 A.9.4.9 *ERE Expression Anchoring*

2176 There is no additional rationale for this section.

2177 **A.9.5 Regular Expression Grammar**

2178 The grammars are intended to represent the range of acceptable syntaxes available to portable  
 2179 applications. There are instances in the text where undefined constructs are described; as  
 2180 explained previously, these allow implementation extensions. There is no intended requirement  
 2181 that an implementation extension must somehow fit into the grammars shown here.

2182 The BRE grammar does not permit L\_ANCHOR or R\_ANCHOR inside "\(" and "\)" (which  
 2183 implies that '^' and '\$' are ordinary characters). This reflects the semantic limits on the  
 2184 application, as noted in the Base Definitions volume of IEEE Std. 1003.1-200x, Section 9.3.8, BRE  
 2185 Expression Anchoring. Implementations are permitted to extend the language to interpret '^'  
 2186 and '\$' as anchors in these locations, and as such, portable applications cannot use unescaped  
 2187 '^' and '\$' in positions inside "\(" and "\)" that might be interpreted as anchors.

2188 The ERE grammar does not permit several constructs that the Base Definitions volume of  
 2189 IEEE Std. 1003.1-200x, Section 9.4.2, ERE Ordinary Characters and the Base Definitions volume  
 2190 of IEEE Std. 1003.1-200x, Section 9.4.3, ERE Special Characters specify as having undefined  
 2191 results:

- 2192 • ORD\_CHAR preceded by '\'
- 2193 • *ERE\_dupl\_symbol*(s) appearing first in an ERE, or immediately following '|', '^', or '('
- 2194 • '{' not part of a valid *ERE\_dupl\_symbol*
- 2195 • '|' appearing first or last in an ERE, or immediately following '|' or '(', or immediately  
 2196 preceding ')'

2197 Implementations are permitted to extend the language to allow these. Portable applications  
 2198 cannot use such constructs.

2199 A.9.5.1 *BRE/ERE Grammar Lexical Conventions*

2200 There is no additional rationale for this section.

2201 A.9.5.2 *RE and Bracket Expression Grammar*

2202 The removal of the *Back\_open\_paren Back\_close\_paren* option from the *nondupl\_RE* specification is  
 2203 the result of PASC Interpretation 1003.2-92 #43 submitted for the ISO POSIX-2: 1993 standard.  
 2204 Although the grammar required support for null subexpressions, this section does not describe  
 2205 the meaning of, and historical practice did not support, this construct.

2206 A.9.5.3 *ERE Grammar*

2207 There is no additional rationale for this section.

**2208 A.10 Directory Structure and Devices****2209 A.10.1 Directory Structure and Files**

2210 A description of the historical **/usr/tmp** was omitted, removing any concept of differences in  
2211 emphasis between the **/** and **/usr** directories. The descriptions of **/bin**, **/usr/bin**, **/lib**, and **/usr/lib**  
2212 were omitted because they are not useful for applications. In an early draft, a distinction was  
2213 made between system and application directory usage, but this was not found to be useful.

2214 The directories **/** and **/dev** are included because the notion of a hierarchical directory structure is  
2215 key to other information presented elsewhere in IEEE Std. 1003.1-200x. In early drafts, it was  
2216 argued that special devices and temporary files could conceivably be handled without a  
2217 directory structure on some implementations. For example, the system could treat the characters  
2218 `" /tmp "` as a special token that would store files using some non-POSIX file system structure.  
2219 This notion was rejected by the standard developers, who required that all the files in this  
2220 section be implemented via POSIX file systems.

2221 The **/tmp** directory is retained in IEEE Std. 1003.1-200x to accommodate historical applications  
2222 that assume its availability. Implementations are encouraged to provide suitable directory  
2223 names in the environment variable *TMPDIR* and applications are encouraged to use the contents  
2224 of *TMPDIR* for creating temporary files.

2225 The standard files **/dev/null** and **/dev/tty** are required to be both readable and writable to allow  
2226 applications to have the intended historical access to these files.

2227 The standard file **/dev/console** has been added for alignment with the Single UNIX Specification.

**2228 A.10.2 Output Devices and Terminal Types**

2229 There is no additional rationale for this section.

## 2230 A.11 General Terminal Interface

2231 If the implementation does not support this interface on any device types, it should behave as if  
2232 it were being used on a device that is not a terminal device (in most cases *errno* will be set to  
2233 [ENOTTY] on return from functions defined by this interface). This is based on the fact that  
2234 many applications are written to run both interactively and in some non-interactive mode, and  
2235 they adapt themselves at runtime. Requiring that they all be modified to test an environment  
2236 variable to determine whether they should try to adapt is unnecessary. On a system that  
2237 provides no general terminal interface, providing all the entry points as stubs that return  
2238 [ENOTTY] (or an equivalent, as appropriate) has the same effect and requires no changes to the  
2239 application.

2240 Although the needs of both interface implementors and application developers were addressed  
2241 throughout IEEE Std. 1003.1-200x, this section pays more attention to the needs of the latter. This  
2242 is because, while many aspects of the programming interface can be hidden from the user by the  
2243 application developer, the terminal interface is usually a large part of the user interface.  
2244 Although to some extent the application developer can build missing features or work around  
2245 inappropriate ones, the difficulties of doing that are greater in the terminal interface than  
2246 elsewhere. For example, efficiency prohibits the average program from interpreting every  
2247 character passing through it in order to simulate character erase, line kill, and so on. These  
2248 functions should usually be done by the operating system, possibly at the interrupt level.

2249 The *tc\**() functions were introduced as a way of avoiding the problems inherent in the  
2250 traditional *ioctl*() function and in variants of it that were proposed. For example, *tcsetattr*() is  
2251 specified in place of the use of the TCSETA *ioctl*() command function. This allows specification  
2252 of all the arguments in a manner consistent with the ISO C standard unlike the varying third  
2253 argument of *ioctl*(), which is sometimes a pointer (to any of many different types) and  
2254 sometimes an **int**.

2255 The advantages of this new method include:

- 2256 • It allows strict type checking.
- 2257 • The direction of transfer of control data is explicit.
- 2258 • Portable capabilities are clearly identified.
- 2259 • The need for a general interface routine is avoided.
- 2260 • Size of the argument is well-defined (there is only one type).

2261 The disadvantages include:

- 2262 • No historical implementation uses the new method.
- 2263 • There are many small routines instead of one general-purpose one.
- 2264 • The historical parallel with *fcntl*() is broken.

2265 The issue of modem control was excluded from IEEE Std. 1003.1-200x on the grounds that:

- 2266 • It was concerned with setting and control of hardware timers.
- 2267 • The appropriate timers and settings vary widely internationally.
- 2268 • Feedback from European computer manufacturers indicated that this facility was not  
2269 consistent with European needs and that specification of such a facility was not a  
2270 requirement for portability.

2271 **A.11.1 Interface Characteristics**2272 *A.11.1.1 Opening a Terminal Device File*

2273 There is no additional rationale provided for this section.

2274 *A.11.1.2 Process Groups*

2275 There is a potential race when the members of the foreground process group on a terminal leave  
2276 that process group, either by exit or by changing process groups. After the last process exits the  
2277 process group, but before the foreground process group ID of the terminal is changed (usually  
2278 by a job-control shell), it would be possible for a new process to be created with its process ID  
2279 equal to the terminal's foreground process group ID. That process might then become the  
2280 process group leader and accidentally be placed into the foreground on a terminal that was not  
2281 necessarily its controlling terminal. As a result of this problem, the controlling terminal is  
2282 defined to not have a foreground process group during this time.

2283 The cases where a controlling terminal has no foreground process group occur when all  
2284 processes in the foreground process group either terminate and are waited for or join other  
2285 process groups via *setpgid()* or *setsid()*. If the process group leader terminates, this is the first  
2286 case described; if it leaves the process group via *setpgid()*, this is the second case described (a  
2287 process group leader cannot successfully call *setsid()*). When one of those cases causes a  
2288 controlling terminal to have no foreground process group, it has two visible effects on  
2289 applications. The first is the value returned by *tcgetpgrp()*. The second (which occurs only in the  
2290 case where the process group leader terminates) is the sending of signals in response to special  
2291 input characters. The intent of IEEE Std. 1003.1-200x is that no process group be wrongly  
2292 identified as the foreground process group by *tcgetpgrp()* or unintentionally receive signals  
2293 because of placement into the foreground.

2294 In 4.3 BSD, the old process group ID continues to be used to identify the foreground process  
2295 group and is returned by the function equivalent to *tcgetpgrp()*. In that implementation it is  
2296 possible for a newly created process to be assigned the same value as a process ID and then form  
2297 a new process group with the same value as a process group ID. The result is that the new  
2298 process group would receive signals from this terminal for no apparent reason, and  
2299 IEEE Std. 1003.1-200x precludes this by forbidding a process group from entering the foreground  
2300 in this way. It would be more direct to place part of the requirement made by the last sentence  
2301 under *fork()*, but there is no convenient way for that section to refer to the value that *tcgetpgrp()*  
2302 returns, since in this case there is no process group and thus no process group ID.

2303 One possibility for a conforming implementation is to behave similarly to 4.3 BSD, but to  
2304 prevent this reuse of the ID, probably in the implementation of *fork()*, as long as it is in use by  
2305 the terminal.

2306 Another possibility is to recognize when the last process stops using the terminal's foreground  
2307 process group ID, which is when the process group lifetime ends, and to change the terminal's  
2308 foreground process group ID to a reserved value that is never used as a process ID or process  
2309 group ID. (See the definition of *process group lifetime* in the definitions section.) The process ID  
2310 can then be reserved until the terminal has another foreground process group.

2311 The 4.3 BSD implementation permits the leader (and only member) of the foreground process  
2312 group to leave the process group by calling the equivalent of *setpgid()* and to later return,  
2313 expecting to return to the foreground. There are no known application needs for this behavior,  
2314 and IEEE Std. 1003.1-200x neither requires nor forbids it (except that it is forbidden for session  
2315 leaders) by leaving it unspecified.

2316 A.11.1.3 *The Controlling Terminal*

2317 IEEE Std. 1003.1-200x does not specify a mechanism by which to allocate a controlling terminal.  
2318 This is normally done by a system utility (such as *getty*) and is considered an administrative  
2319 feature outside the scope of IEEE Std. 1003.1-200x.

2320 Historical implementations allocate controlling terminals on certain *open()* calls. Since *open()* is  
2321 part of POSIX.1, its behavior had to be dealt with. The traditional behavior is not required  
2322 because it is not very straightforward or flexible for either implementations or applications.  
2323 However, because of its prevalence, it was not practical to disallow this behavior either. Thus, a  
2324 mechanism was standardized to ensure portable, predictable behavior in *open()*.

2325 Some historical implementations deallocate a controlling terminal on the last system-wide close.  
2326 This behavior is neither required nor prohibited. Even on implementations that do provide this  
2327 behavior, applications generally cannot depend on it due to its system-wide nature.

2328 A.11.1.4 *Terminal Access Control*

2329 The access controls described in this section apply only to a process that is accessing its  
2330 controlling terminal. A process accessing a terminal that is not its controlling terminal is  
2331 effectively treated the same as a member of the foreground process group. While this may seem  
2332 unintuitive, note that these controls are for the purpose of job control, not security, and job  
2333 control relates only to a process' controlling terminal. Normal file access permissions handle  
2334 security.

2335 If the process calling *read()* or *write()* is in a background process group that is orphaned, it is not  
2336 desirable to stop the process group, as it is no longer under the control of a job control shell that  
2337 could put it into foreground again. Accordingly, calls to *read()* or *write()* functions by such  
2338 processes receive an immediate error return. This is different than in 4.2 BSD, which kills  
2339 orphaned processes that receive terminal stop signals.

2340 The foreground/background/orphaned process group check performed by the terminal driver  
2341 must be repeatedly performed until the calling process moves into the foreground or until the  
2342 process group of the calling process becomes orphaned. That is, when the terminal driver  
2343 determines that the calling process is in the background and should receive a job control signal,  
2344 it sends the appropriate signal (SIGTTIN or SIGTTOU) to every process in the process group of  
2345 the calling process and then it allows the calling process to immediately receive the signal. The  
2346 latter is typically performed by blocking the process so that the signal is immediately noticed.  
2347 Note, however, that after the process finishes receiving the signal and control is returned to the  
2348 driver, the terminal driver must reexecute the foreground/background/orphaned process group  
2349 check. The process may still be in the background, either because it was continued in the  
2350 background by a job-control shell, or because it caught the signal and did nothing.

2351 The terminal driver repeatedly performs the foreground/background/orphaned process group  
2352 checks whenever a process is about to access the terminal. In the case of *write()* or the control  
2353 *tc\*()* functions, the check is performed at the entry of the function. In the case of *read()*, the check  
2354 is performed not only at the entry of the function, but also after blocking the process to wait for  
2355 input characters (if necessary). That is, once the driver has determined that the process calling  
2356 the *read()* function is in the foreground, it attempts to retrieve characters from the input queue. If  
2357 the queue is empty, it blocks the process waiting for characters. When characters are available  
2358 and control is returned to the driver, the terminal driver must return to the repeated  
2359 foreground/background/orphaned process group check again. The process may have moved  
2360 from the foreground to the background while it was blocked waiting for input characters.

2361 A.11.1.5 *Input Processing and Reading Data*

2362 There is no additional rationale provided for this section.

2363 A.11.1.6 *Canonical Mode Input Processing*

2364 The term *character* is intended here. ERASE should erase the last character, not the last byte. In  
2365 the case of multi-byte characters, these two may be different.

2366 4.3 BSD has a WERASE character that erases the last “word” typed (but not any preceding  
2367 <blank>s or <tab>s). A word is defined as a sequence of non-<blank> characters, with <tab>s  
2368 counted as <blank>s. Like ERASE, WERASE does not erase beyond the beginning of the line.  
2369 This WERASE feature has not been specified in POSIX.1 because it is difficult to define in the  
2370 international environment. It is only useful for languages where words are delimited by  
2371 <blank>s. In some ideographic languages, such as Japanese and Chinese, words are not  
2372 delimited at all. The WERASE character should presumably take one back to the beginning of a  
2373 sentence in those cases; practically, this means it would not get much use for those languages.

2374 It should be noted that there is a possible inherent deadlock if the application and  
2375 implementation conflict on the value of MAX\_CANON. With ICANON set (if IXOFF is  
2376 enabled) and more than MAX\_CANON characters transmitted without a <linefeed>,  
2377 transmission will be stopped, the <linefeed> (or <carriage-return> when ICRLF is set) will never  
2378 arrive, and the *read()* will never be satisfied.

2379 An application should not set IXOFF if it is using canonical mode unless it knows that (even in  
2380 the face of a transmission error) the conditions described previously cannot be met or unless it is  
2381 prepared to deal with the possible deadlock in some other way, such as timeouts.

2382 It should also be noted that this can be made to happen in non-canonical mode if the trigger  
2383 value for sending IXOFF is less than VMIN and VTIME is zero.

2384 A.11.1.7 *Non-Canonical Mode Input Processing*

2385 Some points to note about MIN and TIME:

- 2386 1. The interactions of MIN and TIME are not symmetric. For example, when MIN>0 and  
2387 TIME=0, TIME has no effect. However, in the opposite case where MIN=0 and TIME>0,  
2388 both MIN and TIME play a role in that MIN is satisfied with the receipt of a single  
2389 character.
- 2390 2. Also note that in case A (MIN>0, TIME>0), TIME represents an inter-character timer, while  
2391 in case C (MIN=0, TIME>0), TIME represents a read timer.

2392 These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B, where  
2393 MIN>0, exist to handle burst-mode activity (for example, file transfer programs) where a  
2394 program would like to process at least MIN characters at a time. In case A, the inter-character  
2395 timer is activated by a user as a safety measure; in case B, it is turned off.

2396 Cases C and D exist to handle single-character timed transfers. These cases are readily adaptable  
2397 to screen-based applications that need to know if a character is present in the input queue before  
2398 refreshing the screen. In case C, the read is timed; in case D, it is not.

2399 Another important note is that MIN is always just a minimum. It does not denote a record  
2400 length. That is, if a program does a read of 20 bytes, MIN is 10, and 25 characters are present, 20  
2401 characters shall be returned to the user. In the special case of MIN=0, this still applies: if more  
2402 than one character is available, they all will be returned immediately.



2403 **A.11.1.8 Writing Data and Output Processing**

2404 There is no additional rationale for this section.

2405 **A.11.1.9 Special Characters**

2406 There is no additional rationale for this section.

2407 **A.11.1.10 Modem Disconnect**

2408 There is no additional rationale for this section.

2409 **A.11.1.11 Closing a Terminal Device File**

2410 IEEE Std. 1003.1-200x does not specify that a *close()* on a terminal device file include the  
2411 equivalent of a call to *tcfow(fd,TCOON)*.

2412 An implementation that discards output at the time *close()* is called after reporting the return  
2413 value to the *write()* call that data was written does not conform with IEEE Std. 1003.1-200x. An  
2414 application has functions such as *tcdrain()*, *tcfush()*, and *tcfow()* available to obtain the detailed  
2415 behavior it requires with respect to flushing of output.

2416 At the time of the last close on a terminal device, an application relinquishes any ability to exert  
2417 flow control via *tcfow()*.

2418 **A.11.2 Parameters that Can be Set**2419 **A.11.2.1 The termios Structure**

2420 This structure is part of an interface that, in general, retains the historic grouping of flags.  
2421 Although a more optimal structure for implementations may be possible, the degree of change  
2422 to applications would be significantly larger.

2423 **A.11.2.2 Input Modes**

2424 Some historical implementations treated a long break as multiple events, as many as one per  
2425 character time. The wording in POSIX.1 explicitly prohibits this.

2426 Although the ISTRIP flag is normally superfluous with today's terminal hardware and software,  
2427 it is historically supported. Therefore, applications may be using ISTRIP, and there is no  
2428 technical problem with supporting this flag. Also, applications may wish to receive only 7-bit  
2429 input bytes and may not be connected directly to the hardware terminal device (for example,  
2430 when a connection traverses a network).

2431 Also, there is no requirement in general that the terminal device ensures that high-order bits  
2432 beyond the specified character size are cleared. ISTRIP provides this function for 7-bit  
2433 characters, which are common.

2434 In dealing with multi-byte characters, the consequences of a parity error in such a character, or in  
2435 an escape sequence affecting the current character set, are beyond the scope of POSIX.1 and are  
2436 best dealt with by the application processing the multi-byte characters.

2437 *A.11.2.3 Output Modes*

2438 POSIX.1 does not describe postprocessing of output to a terminal or detailed control of that from  
2439 a portable application. (That is, translation of <newline> to <carriage-return> followed by  
2440 <linefeed> or <tab> processing.) There is nothing that a portable application should do to its  
2441 output for a terminal because that would require knowledge of the operation of the terminal. It  
2442 is the responsibility of the operating system to provide postprocessing appropriate to the output  
2443 device, whether it is a terminal or some other type of device.

2444 Extensions to POSIX.1 to control the type of postprocessing already exist and are expected to  
2445 continue into the future. The control of these features is primarily to adjust the interface between  
2446 the system and the terminal device so the output appears on the display correctly. This should  
2447 be set up before use by any application.

2448 In general, both the input and output modes should not be set absolutely, but rather modified  
2449 from the inherited state.

2450 *A.11.2.4 Control Modes*

2451 This section could be misread that the symbol “CSIZE” is a title in the **termios** *c\_cflag* field .  
2452 Although it does serve that function, it is also a required symbol, as a literal reading of POSIX.1  
2453 (and the caveats about typography) would indicate.

2454 *A.11.2.5 Local Modes*

2455 Non-canonical mode is provided to allow fast bursts of input to be read efficiently while still  
2456 allowing single-character input.

2457 The ECHONL function historically has been in many implementations. Since there seems to be  
2458 no technical problem with supporting ECHONL, it is included in POSIX.1 to increase consensus.

2459 The alternate behavior possible when ECHOK or ECHOE are specified with ICANON is  
2460 permitted as a compromise depending on what the actual terminal hardware can do. Erasing  
2461 characters and lines is preferred, but is not always possible.

2462 *A.11.2.6 Special Control Characters*

2463 Permitting VMIN and VTIME to overlap with VEOF and VEOL was a compromise for historical  
2464 implementations. Only when backwards-compatibility of object code is a serious concern to an  
2465 implementor should an implementation continue this practice. Correct applications that work  
2466 with the overlap (at the source level) should also work if it is not present, but not the reverse.

2467 **A.12 Utility Conventions**2468 **A.12.1 Utility Argument Syntax**

2469 The standard developers considered that recent trends toward diluting the SYNOPSIS sections  
2470 of historical reference pages to the equivalent of:

2471 `command [options][operands]`

2472 were a disservice to the reader. Therefore, considerable effort was placed into rigorous  
2473 definitions of all the command line arguments and their interrelationships. The relationships  
2474 depicted in the synopses are normative parts of IEEE Std. 1003.1-200x; this information is  
2475 sometimes repeated in textual form, but that is only for clarity within context.

2476 The use of “undefined” for conflicting argument usage and for repeated usage of the same  
2477 option is meant to prevent portable applications from using conflicting arguments or repeated  
2478 options unless specifically allowed (as is the case with *ls*, which allows simultaneous, repeated  
2479 use of the *-C*, *-l*, and *-1* options). Many historical implementations will tolerate this usage,  
2480 choosing either the first or the last applicable argument. This tolerance can continue, but  
2481 portable applications cannot rely upon it. (Other implementations may choose to print usage  
2482 messages instead.)

2483 The use of “undefined” for conflicting argument usage also allows an implementation to make  
2484 reasonable extensions to utilities where the implementor considers mutually-exclusive options  
2485 according to IEEE Std. 1003.1-200x to have a sensible meaning and result.

2486 IEEE Std. 1003.1-200x does not define the result of a command when an option-argument or  
2487 operand is not followed by ellipses and the application specifies more than one of that option-  
2488 argument or operand. This allows an implementation to define valid (although non-standard)  
2489 behavior for the utility when more than one such option or operand is specified.

2490 Allowing <blank> characters after an option (that is, placing an option and its option-argument  
2491 into separate argument strings) when IEEE Std. 1003.1-200x does not require it encourages  
2492 portability of users, while still preserving backwards-compatibility of scripts. Inserting <blank>  
2493 characters between the option and the option-argument is preferred; however, historical usage  
2494 has not been consistent in this area; therefore, <blank>s are required to be handled by all  
2495 implementations, but implementations are also allowed to handle the historical syntax. Another  
2496 justification for selecting the multiple-argument method was that the single-argument case is  
2497 inherently ambiguous when the option-argument can legitimately be a null string.

2498 IEEE Std. 1003.1-200x explicitly states that digits are permitted as operands and option-  
2499 arguments. The lower and upper bounds for the values of the numbers used for operands and  
2500 option-arguments were derived from the ISO C standard values for {LONG\_MIN} and  
2501 {LONG\_MAX}. The requirement on the standard utilities is that numbers in the specified range  
2502 do not cause a syntax error, although the specification of a number need not be semantically  
2503 correct for a particular operand or option-argument of a utility. For example, the specification of:

2504 `dd obs=3000000000`

2505 would yield undefined behavior for the application and would be a syntax error because the  
2506 number 3 000 000 000 is outside of the range *-2 147 483 647* to *+2 147 483 647*. On the other hand:

2507 `dd obs=2000000000`

2508 may cause some error, such as “blocksize too large”, rather than a syntax error.

2509 **A.12.2 Utility Syntax Guidelines**

2510 This section is based on the rules listed in the SVID. It was included for two reasons:

- 2511 1. The individual utility descriptions in the Shell and Utilities volume of  
2512 IEEE Std. 1003.1-200x, Chapter 4, Utilities needed a set of common (although not universal)  
2513 actions on which they could anchor their descriptions of option and operand syntax. Most  
2514 of the standard utilities actually do use these guidelines, and many of their historical  
2515 implementations use the *getopt()* function for their parsing. Therefore, it was simpler to  
2516 cite the rules and merely identify exceptions.
- 2517 2. Writers of portable applications need suggested guidelines if the POSIX community is to  
2518 avoid the chaos of historical UNIX system command syntax.

2519 It is recommended that all *future* utilities and applications use these guidelines to enhance “user  
2520 portability”. The fact that some historical utilities could not be changed (to avoid breaking  
2521 historical applications) should not deter this future goal.

2522 The voluntary nature of the guidelines is highlighted by repeated uses of the word *should*  
2523 throughout. This usage should not be misinterpreted to imply that utilities that claim  
2524 conformance in their OPTIONS sections do not always conform.

2525 Guidelines 1 and 2 are offered as guidance for locales using Latin alphabets. No  
2526 recommendations are made by IEEE Std. 1003.1-200x concerning utility naming in other locales.

2527 In the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.9.1, Simple Commands, it is  
2528 further stated that a command used in the Shell Command Language cannot be named with a  
2529 trailing colon.

2530 Guideline 3 was changed to allow alphanumeric characters (letters and digits) from the character  
2531 set to allow compatibility with historical usage. Historical practice allows the use of digits  
2532 wherever practical, and there are no portability issues that would prohibit the use of digits. In  
2533 fact, from an internationalization viewpoint, digits (being non-language-dependent) are  
2534 preferable over letters (a *-2* is intuitively self-explanatory to any user, while in the *-f filename* the  
2535 letter ‘f’ is a mnemonic aid only to speakers of Latin-based languages where “file name”  
2536 happens to translate to a word that begins with ‘f’). Since guideline 3 still retains the word  
2537 “single”, multi-digit options are not allowed. Instances of historical utilities that used them have  
2538 been marked obsolescent, with the numbers being changed from option names to option-  
2539 arguments.

2540 It was difficult to achieve a satisfactory solution to the problem of name space in option  
2541 characters. When the standard developers desired to extend the historical *cc* utility to accept  
2542 ISO C standard programs, they found that all of the portable alphabet was already in use by  
2543 various vendors. Thus, they had to devise a new name, *c89*, rather than something like *cc -X*.  
2544 There were suggestions that implementors be restricted to providing extensions through various  
2545 means (such as using a plus sign as the option delimiter or using option characters outside the  
2546 alphanumeric set) that would reserve all of the remaining alphanumeric characters for future  
2547 POSIX standards. These approaches were resisted because they lacked the historical style of  
2548 UNIX systems. Furthermore, if a vendor-provided option should become commonly used in the  
2549 industry, it would be a candidate for standardization. It would be desirable to standardize such a  
2550 feature using historical practice for the syntax (the semantics can be standardized with any  
2551 syntax). This would not be possible if the syntax was one reserved for the vendor. However,  
2552 since the standardization process may lead to minor changes in the semantics, it may prove to be  
2553 better for a vendor to use a syntax that will not be affected by standardization.

2554 Guideline 8 includes the concept of comma-separated lists in a single argument. It is up to the  
2555 utility to parse such a list itself because *getopt()* just returns the single string. This situation was

2556 retained so that certain historical utilities would not violate the guidelines. Applications  
2557 preparing for international use should be aware of an occasional problem with comma-  
2558 separated lists: in some locales, the comma is used as the radix character. Thus, if an application  
2559 is preparing operands for a utility that expects a comma-separated lists, it should avoid  
2560 generating non-integer values through one of the means that is influenced by setting the  
2561 *LC\_NUMERIC* variable (such as *awk*, *bc*, *printf*, or *printf()*).

2562 Applications calling any utility with a first operand starting with '-' should usually specify --,  
2563 as indicated by Guideline 10, to mark the end of the options. This is true even if the SYNOPSIS in  
2564 the Shell and Utilities volume of IEEE Std. 1003.1-200x does not specify any options;  
2565 implementations may provide options as extensions to the Shell and Utilities volume of  
2566 IEEE Std. 1003.1-200x. The standard utilities that do not support Guideline 10 indicate that fact  
2567 in the OPTIONS section of the utility description.

2568 Guideline 11 was modified to clarify that the order of different options should not matter  
2569 relative to one another. However, the order of repeated options that also have option-arguments  
2570 may be significant; therefore, such options are required to be interpreted in the order that they  
2571 are specified. The *make* utility is an instance of a historical utility that uses repeated options  
2572 in which the order is significant. Multiple files are specified by giving multiple instances of the -f  
2573 option; for example:

```
2574 make -f common_header -f specific_rules target
```

2575 Guideline 13 does not imply that all of the standard utilities automatically accept the operand  
2576 '-' to mean standard input or output, nor does it specify the actions of the utility upon  
2577 encountering multiple '-' operands. It simply says that, by default, '-' operands are not used  
2578 for other purposes in the file reading or writing (but not when using *stat*, *unlink*, *touch*, and so on)  
2579 utilities. All information concerning actual treatment of the '-' operand is found in the  
2580 individual utility sections.

2581 An area of concern was that as implementations mature, implementation-defined utilities and  
2582 implementation-defined utility options will result. The idea was expressed that there needed to  
2583 be a standard way, say an environment variable or some such mechanism, to identify  
2584 implementation-defined utilities separately from standard utilities that may have the same  
2585 name. It was decided that there already exist several ways of dealing with this situation and that  
2586 it is outside of the POSIX.2 scope to attempt to standardize in the area of non-standard items. A  
2587 method that exists on some historical implementations is the use of the so-called */local/bin* or  
2588 */usr/local/bin* directory to separate local or additional copies or versions of utilities. Another  
2589 method that is also used is to isolate utilities into completely separate domains. Still another  
2590 method to ensure that the desired utility is being used is to request the utility by its full path  
2591 name. There are many approaches to this situation; the examples given above serve to illustrate  
2592 that there is more than one.

**2593 A.13 Headers****2594 A.13.1 Format of Entries**

2595 Each header reference page has a common layout of sections describing the interface. This layout  
2596 is similar to the manual page or “man” page format shipped with most UNIX systems, and each  
2597 header has sections describing the SYNOPSIS and DESCRIPTION. These are the two sections  
2598 that relate to conformance.

2599 Additional sections are informative, and add considerable information for the application  
2600 developer. APPLICATION USAGE sections provide additional caveats, issues, and  
2601 recommendations to the developer. RATIONALE sections give additional information on the  
2602 decisions made in defining the interface.

2603 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
2604 the future, and often cautions the developer to architect the code to account for a change in this  
2605 area. Note that a future directions statement should not be taken as a commitment to adopt a  
2606 feature or interface in the future.

2607 The CHANGE HISTORY section describes when the interface was introduced, and how it has  
2608 changed.

2609 Option labels and margin markings in the page can be useful in guiding the application  
2610 developer.

2611 / *Rationale (Informative)*

2612 **Part B:**

2613 **System Interfaces**

2614 *The Open Group*





# Rationale for System Interfaces

2615

## 2616 **B.1 Introduction**

### 2617 **B.1.1 Scope**

2618 Refer to Section A.1.1 (on page 3311).

### 2619 **B.1.2 Conformance**

2620 Refer to Section A.2 (on page 3317).

### 2621 **B.1.3 Normative References**

2622 There is no additional rationale for this section.

### 2623 **B.1.4 Changes from Issue 4**

2624 The change history is provided as an informative section, to track changes from previous issues  
2625 of IEEE Std. 1003.1-200x that comprised earlier versions of the Single UNIX Specification.

#### 2626 *B.1.4.1 Changes from Issue 4 to Issue 4, Version 2*

2627 There is no additional rationale for this section.

#### 2628 *B.1.4.2 Changes from Issue 4, Version 2 to Issue 5*

2629 There is no additional rationale for this section.

#### 2630 *B.1.4.3 Changes from Issue 5 to Issue 6 (IEEE Std. 1003.1-200x)*

2631 There is no additional rationale for this section.

### 2632 **B.1.5 New Features**

2633 There is no additional rationale for this section.

#### 2634 *B.1.5.1 New Features in Issue 4, Version 2*

2635 There is no additional rationale for this section.

#### 2636 *B.1.5.2 New Features in Issue 5*

2637 There is no additional rationale for this section.

#### 2638 *B.1.5.3 New Features in Issue 6*

2639 There is no additional rationale for this section.

**2640 B.1.6 Terminology**

2641 Refer to Section A.1.4 (on page 3313).

**2642 B.1.7 Definitions**

2643 Refer to Section A.3 (on page 3321).

**2644 B.1.8 Relationship to Other Formal Standards**

2645 There is no additional rationale for this section.

**2646 B.1.9 Portability**

2647 Refer to Section A.1.5 (on page 3315).

**2648 B.1.9.1 Codes**

2649 Refer to Section A.1.5.1 (on page 3315).

**2650 B.1.10 Format of Entries**

2651 Each system interface reference page has a common layout of sections describing the interface.  
2652 This layout is similar to the manual page or “man” page format shipped with most UNIX  
2653 systems, and each header has sections describing the SYNOPSIS, DESCRIPTION, RETURN  
2654 VALUE, and ERRORS. These are the four sections that relate to conformance.

2655 Additional sections are informative, and add considerable information for the application  
2656 developer. EXAMPLES sections provide example usage. APPLICATION USAGE sections  
2657 provide additional caveats, issues, and recommendations to the developer. RATIONALE  
2658 sections give additional information on the decisions made in defining the interface.

2659 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
2660 the future, and often cautions the developer to architect the code to account for a change in this  
2661 area. Note that a future directions statement should not be taken as a commitment to adopt a  
2662 feature or interface in the future.

2663 The CHANGE HISTORY section describes when the interface was introduced, and how it has  
2664 changed.

2665 Option labels and margin markings in the page can be useful in guiding the application  
2666 developer.

2667 **B.2 General Information**2668 **B.2.1 Use and Implementation of Functions**

2669 The information concerning the use of functions was adapted from a description in the ISO C  
 2670 standard. Here is an example of how an application program can protect itself from library  
 2671 functions that may or may not be macros, rather than true functions:

2672 The *atoi()* function may be used in any of several ways:

- 2673 • By use of its associated header (possibly generating a macro expansion):

```
2674 #include <stdlib.h>
2675 /* ... */
2676 i = atoi(str);
```

- 2677 • By use of its associated header (assuredly generating a true function call):

```
2678 #include <stdlib.h>
2679 #undef atoi
2680 /* ... */
2681 i = atoi(str);
```

2682 or:

```
2683 #include <stdlib.h>
2684 /* ... */
2685 i = (atoi) (str);
```

- 2686 • By explicit declaration:

```
2687 extern int atoi (const char *);
2688 /* ... */
2689 i = atoi(str);
```

- 2690 • By implicit declaration:

```
2691 /* ... */
2692 i = atoi(str);
```

2693 (Assuming no function prototype is in scope. This is not allowed by the ISO C standard for  
 2694 functions with variable arguments; furthermore, parameter type conversion “widening” is  
 2695 subject to different rules in this case.)

2696 Note that the ISO C standard reserves names starting with ‘\_’ for the compiler. Therefore, the  
 2697 compiler could, for example, implement an intrinsic, built-in function *\_asm\_builtin\_atoi()*, which  
 2698 it recognized and expanded into inline assembly code. Then, in *<stdlib.h>*, there could be the  
 2699 following:

```
2700 #define atoi(X) _asm_builtin_atoi(X)
```

2701 The user’s “normal” call to *atoi()* would then be expanded inline, but the implementor would  
 2702 also be required to provide a callable function named *atoi()* for use when the application  
 2703 requires it; for example, if its address is to be stored in a function pointer variable.

## 2704 **B.2.2 The Compilation Environment**

### 2705 *B.2.2.1 POSIX.1 Symbols*

2706 This and the following section address the issue of “name space pollution”. The ISO C standard  
2707 requires that the name space beyond what it reserves not be altered except by explicit action of  
2708 the application writer. This section defines the actions to add the POSIX.1 symbols for those  
2709 headers where both the ISO C standard and POSIX.1 need to define symbols, and also where the  
2710 XSI Extension extends the base standard.

2711 When headers are used to provide symbols, there is a potential for introducing symbols that the  
2712 application writer cannot predict. Ideally, each header should only contain one set of symbols,  
2713 but this is not practical for historical reasons. Thus, the concept of feature test macros is  
2714 included. Two feature test macros are explicitly defined by IEEE Std. 1003.1-200x; it is expected  
2715 that future revisions may add to this.

2716 It is further intended that these feature test macros apply only to the headers specified by  
2717 IEEE Std. 1003.1-200x. Implementations are expressly permitted to make visible symbols not  
2718 specified by IEEE Std. 1003.1-200x, within both POSIX.1 and other headers, under the control of  
2719 feature test macros that are not defined by IEEE Std. 1003.1-200x.

### 2720 **The `_POSIX_C_SOURCE` Feature Test Macro**

2721 Since `_POSIX_SOURCE` specified by the POSIX.1-1990 standard did not have a value associated  
2722 with it, the `_POSIX_C_SOURCE` macro replaces it, allowing an application to inform the system  
2723 of the revision of the standard to which it conforms. This symbol will allow implementations to  
2724 support various revisions of IEEE Std. 1003.1-200x simultaneously. For instance, when either  
2725 `_POSIX_SOURCE` is defined or `_POSIX_C_SOURCE` is defined as 1, the system should make  
2726 visible the same name space as permitted and required by the POSIX.1-1990 standard. When  
2727 `_POSIX_C_SOURCE` is defined, the state of `_POSIX_SOURCE` is completely irrelevant.

2728 It is expected that C bindings to future POSIX standards will define new values for  
2729 `_POSIX_C_SOURCE`, with each new value reserving the name space for that new standard, plus  
2730 all earlier POSIX standards. Using a single feature test macro for all standards rather than a  
2731 separate macro for each standard furthers the goal of eventually combining all of the C bindings  
2732 into one standard.

2733 It is further intended that these feature test macros apply only to the headers specified by  
2734 IEEE Std. 1003.1-200x. Implementations are expressly permitted to make visible symbols not  
2735 specified by IEEE Std. 1003.1-200x, within both IEEE Std. 1003.1-200x and other headers, under  
2736 the control of feature test macros that are not defined by IEEE Std. 1003.1-200x.

### 2737 *B.2.2.2 The Name Space*

2738 The reservation of identifiers is paraphrased from the ISO C standard. The text is included  
2739 because it needs to be part of IEEE Std. 1003.1-200x, regardless of possible changes in future  
2740 versions of the ISO C standard.

2741 These identifiers may be used by implementations, particularly for feature test macros.  
2742 Implementations should not use feature test macro names that might be reasonably used by a  
2743 standard.

2744 Including headers more than once is a reasonably common practice, and it should be carried  
2745 forward from the ISO C standard. More significantly, having definitions in more than one  
2746 header is explicitly permitted. Where the potential declaration is “benign” (the same definition  
2747 twice) the declaration can be repeated, if that is permitted by the compiler. (This is usually true  
2748 of macros, for example.) In those situations where a repetition is not benign (for example,

2749 **typedefs**), conditional compilation must be used. The situation actually occurs both within the  
 2750 ISO C standard and within POSIX.1: **time\_t** should be in `<sys/types.h>`, and the ISO C standard  
 2751 mandates that it be in `<time.h>`.

2752 The area of name space pollution *versus* additions to structures is difficult because of the macro  
 2753 structure of C. The following discussion summarizes all the various problems with and  
 2754 objections to the issue.

2755 Note the phrase “user-defined macro”. Users are not permitted to define macro names (or any  
 2756 other name) beginning with “\_`[A-Z_]`”. Thus, the conflict cannot occur for symbols reserved  
 2757 to the vendor’s name space, and the permission to add fields automatically applies, without  
 2758 qualification, to those symbols.

2759 1. Data structures (and unions) need to be defined in headers by implementations to meet  
 2760 certain requirements of POSIX.1 and the ISO C standard.

2761 2. The structures defined by POSIX.1 are typically minimal, and any practical  
 2762 implementation would wish to add fields to these structures either to hold additional  
 2763 related information or for backwards-compatibility (or both). Future standards (and *de*  
 2764 *facto* standards) would also wish to add to these structures. Issues of field alignment make  
 2765 it impractical (at least in the general case) to simply omit fields when they are not defined  
 2766 by the particular standard involved.

2767 Struct **dirent** is an example of such a minimal structure (although one could argue about  
 2768 whether the other fields need visible names). The `st_rdev` field of most implementations’  
 2769 **stat** structure is a common example where extension is needed and where a conflict could  
 2770 occur.

2771 3. Fields in structures are in an independent name space, so the addition of such fields  
 2772 presents no problem to the C language itself in that such names cannot interact with  
 2773 identically named user symbols because access is qualified by the specific structure name.

2774 4. There is an exception to this: macro processing is done at a lexical level. Thus, symbols  
 2775 added to a structure might be recognized as user-provided macro names at the location  
 2776 where the structure is declared. This only can occur if the user-provided name is declared  
 2777 as a macro before the header declaring the structure is included. The user’s use of the name  
 2778 after the declaration cannot interfere with the structure because the symbol is hidden and  
 2779 only accessible through access to the structure. Presumably, the user would not declare  
 2780 such a macro if there was an intention to use that field name.

2781 5. Macros from the same or a related header might use the additional fields in the structure,  
 2782 and those field names might also collide with user macros. Although this is a less frequent  
 2783 occurrence, since macros are expanded at the point of use, no constraint on the order of use  
 2784 of names can apply.

2785 6. An “obvious” solution of using names in the reserved name space and then redefining  
 2786 them as macros when they should be visible does not work because this has the effect of  
 2787 exporting the symbol into the general name space. For example, given a (hypothetical)  
 2788 system-provided header `<h.h>`, and two parts of a C program in `a.c` and `b.c`, in header  
 2789 `<h.h>`:

```

2790 struct foo {
2791 int __i;
2792 }
2793
2794 #ifdef _FEATURE_TEST
2795 #define i __i;
2796 #endif

```

2796 **In file a.c:**

```

2797 #include h.h
2798 extern int i;
2799 ...

```

2800 **In file b.c:**

```

2801 extern int i;
2802 ...

```

2803 The symbol that the user thinks of as *i* in both files has an external name of `__i` in **a.c**; the  
 2804 same symbol *i* in **b.c** has an external name *i* (ignoring any hidden manipulations the  
 2805 compiler might perform on the names). This would cause a mysterious name resolution  
 2806 problem when **a.o** and **b.o** are linked.

2807 Simply avoiding definition then causes alignment problems in the structure.

2808 A structure of the form:

```

2809 struct foo {
2810 union {
2811 int __i;
2812 #ifdef _FEATURE_TEST
2813 int i;
2814 #endif
2815 } __ii;
2816 }

```

2817 does not work because the name of the logical field *i* is `__ii.i`, and introduction of a macro  
 2818 to restore the logical name immediately reintroduces the problem discussed previously  
 2819 (although its manifestation might be more immediate because a syntax error would result  
 2820 if a recursive macro did not cause it to fail first).

2821 7. A more workable solution would be to declare the structure:

```

2822 struct foo {
2823 #ifdef _FEATURE_TEST
2824 int i;
2825 #else
2826 int __i;
2827 #endif
2828 }

```

2829 However, if a macro (particularly one required by a standard) is to be defined that uses  
 2830 this field, two must be defined: one that uses *i*, the other that uses `__i`. If more than one  
 2831 additional field is used in a macro and they are conditional on distinct combinations of  
 2832 features, the complexity goes up as  $2^n$ .

2833 All this leaves a difficult situation: vendors must provide very complex headers to deal with  
 2834 what is conceptually simple and safe—adding a field to a structure. It is the possibility of user-

2835 provided macros with the same name that makes this difficult.

2836 Several alternatives were proposed that involved constraining the user's access to part of the  
2837 name space available to the user (as specified by the ISO C standard). In some cases, this was  
2838 only until all the headers had been included. There were two proposals discussed that failed to  
2839 achieve consensus:

- 2840 1. Limiting it for the whole program.
- 2841 2. Restricting the use of identifiers containing only uppercase letters until after all system  
2842 headers had been included. It was also pointed out that because macros might wish to  
2843 access fields of a structure (and macro expansion occurs totally at point of use) restricting  
2844 names in this way would not protect the macro expansion, and thus the solution was  
2845 inadequate.

2846 It was finally decided that reservation of symbols would occur, but as constrained.

2847 The current wording also allows the addition of fields to a structure, but requires that user  
2848 macros of the same name not interfere. This allows vendors to do one of the following:

- 2849 • Not create the situation (do not extend the structures with user-accessible names or use the  
2850 solution in (7) above)
- 2851 • Extend their compilers to allow some way of adding names to structures and macros safely

2852 There are at least two ways that the compiler might be extended: add new preprocessor  
2853 directives that turn off and on macro expansion for certain symbols (without changing the value  
2854 of the macro) and a function or lexical operation that suppresses expansion of a word. The latter  
2855 seems more flexible, particularly because it addresses the problem in macros as well as in  
2856 declarations.

2857 The following seems to be a possible implementation extension to the C language that will do  
2858 this: any token that during macro expansion is found to be preceded by three '#' symbols shall  
2859 not be further expanded in exactly the same way as described for macros that expand to their  
2860 own name as in Section 3.8.3.4 of the ISO C standard. A vendor may also wish to implement this  
2861 as an operation that is lexically a function, which might be implemented as:

```
2862 #define __safe_name(x) ###x
```

2863 Using a function notation would insulate vendors from changes in standards until such a  
2864 functionality is standardized (if ever). Standardization of such a function would be valuable  
2865 because it would then permit third parties to take advantage of it portably in software they may  
2866 supply.

2867 The symbols that are "explicitly permitted, but not required by IEEE Std. 1003.1-200x" include  
2868 those classified below. (That is, the symbols classified below might, but are not required to, be  
2869 present when `_POSIX_C_SOURCE` is defined to have the value 20010xL.)

- 2870 • Symbols in `<limits.h>` and `<unistd.h>` that are defined to indicate support for options or  
2871 limits that are constant at compile-time.
- 2872 • Symbols in the name space reserved for the implementation by the ISO C standard.
- 2873 • Symbols in a name space reserved for a particular type of extension (for example, type names  
2874 ending with `_t` in `<sys/types.h>`).
- 2875 • Additional members of structures or unions whose names do not reduce the name space  
2876 reserved for applications.

2877 Since both implementations and future revisions of IEEE Std. 1003.1-200x and other POSIX  
2878 standards may use symbols in the reserved spaces described in these tables, there is a potential

2879 for name space clashes. To avoid future name space clashes when adding symbols,  
2880 implementations should not use the `posix_`, `POSIX_`, or `_POSIX_` prefixes.

### 2881 **B.2.3 Error Numbers**

2882 It was the consensus of the standard developers that to allow the conformance document to  
2883 state that an error occurs and under what conditions, but to disallow a statement that it never  
2884 occurs, does not make sense. It could be implied by the current wording that this is allowed, but  
2885 to reduce the possibility of future interpretation requests, it is better to make an explicit  
2886 statement.

2887 The ISO C standard requires that `errno` be an assignable *lvalue*. Originally, the definition in  
2888 POSIX.1 was stricter than that in the ISO C standard, `extern int errno`, in order to support  
2889 historical usage. In a multi-threaded environment, implementing `errno` as a global variable  
2890 results in non-deterministic results when accessed. It is required, however, that `errno` work as a  
2891 per-thread error reporting mechanism. In order to do this, a separate `errno` value has to be  
2892 maintained for each thread. The following section discusses the various alternative solutions  
2893 that were considered.

2894 In order to avoid this problem altogether for new functions, these functions avoid using `errno`  
2895 and, instead, return the error number directly as the function return value; a return value of zero  
2896 indicates that no error was detected.

2897 For any function that can return errors, the function return value is not used for any purpose  
2898 other than for reporting errors. Even when the output of the function is scalar, it is passed  
2899 through a function argument. While it might have been possible to allow some scalar outputs to  
2900 be coded as negative function return values and mixed in with positive error status returns, this  
2901 was rejected—using the return value for a mixed purpose was judged to be of limited use and  
2902 error prone.

2903 Checking the value of `errno` alone is not sufficient to determine the existence or type of an error,  
2904 since it is not required that a successful function call clear `errno`. The variable `errno` should only  
2905 be examined when the return value of a function indicates that the value of `errno` is meaningful.  
2906 In that case, the function is required to set the variable to something other than zero.

2907 The variable `errno` shall never be set to zero by any function call; to do so would contradict the  
2908 ISO C standard.

2909 POSIX.1 requires (in the ERRORS sections of function descriptions) certain error values to be set  
2910 in certain conditions because many existing applications depend on them. Some error numbers,  
2911 such as [EFAULT], are entirely implementation-defined and are noted as such in their  
2912 description in the ERRORS section. This section otherwise allows wide latitude to the  
2913 implementation in handling error reporting.

2914 Some of the ERRORS sections in IEEE Std. 1003.1-200x have two subsections. The first:

2915        “The function shall fail if:”

2916 could be called the “mandatory” section.

2917 The second:

2918        “The function may fail if:”

2919 could be informally known as the “optional” section.

2920 Attempting to infer the quality of an implementation based on whether it detects optional error  
2921 conditions is not useful.



|      |                |                                                                                                    |
|------|----------------|----------------------------------------------------------------------------------------------------|
| 2922 |                | Following each one-word symbolic name for an error, there is a description of the error. The       |
| 2923 |                | rationale for some of the symbolic names follows:                                                  |
| 2924 | [ECANCELED]    | This spelling was chosen as being more common.                                                     |
| 2925 | [EFAULT]       | Most historical implementations do not catch an error and set <i>errno</i> when an                 |
| 2926 |                | invalid address is given to the functions <i>wait()</i> , <i>time()</i> , or <i>times()</i> . Some |
| 2927 |                | implementations cannot reliably detect an invalid address. And most systems                        |
| 2928 |                | that detect invalid addresses will do so only for a system call, not for a library                 |
| 2929 |                | routine.                                                                                           |
| 2930 | [EFTYPE]       | This error code was proposed in earlier proposals as “Inappropriate operation                      |
| 2931 |                | for file type”, meaning that the operation requested is not appropriate for the                    |
| 2932 |                | file specified in the function call. This code was proposed, although the same                     |
| 2933 |                | idea was covered by [ENOTTY], because the connotations of the name would                           |
| 2934 |                | be misleading. It was pointed out that the <i>fcntl()</i> function uses the error code             |
| 2935 |                | [EINVAL] for this notion, and hence all instances of [EFTYPE] were changed                         |
| 2936 |                | to this code.                                                                                      |
| 2937 | [EINTR]        | POSIX.1 prohibits conforming implementations from restarting interrupted                           |
| 2938 |                | system calls. However, it does not require that [EINTR] be returned when                           |
| 2939 |                | another legitimate value may be substituted; for example, a partial transfer                       |
| 2940 |                | count when <i>read()</i> or <i>write()</i> are interrupted. This is only given when the            |
| 2941 |                | signal catching function returns normally as opposed to returns by                                 |
| 2942 |                | mechanisms like <i>longjmp()</i> or <i>siglongjmp()</i> .                                          |
| 2943 | [ELOOP]        | In specifying conditions under which implementations would generate this                           |
| 2944 |                | error, the following goals were considered:                                                        |
| 2945 |                | • To ensure that actual loops are detected, including loops that result from                       |
| 2946 |                | symbolic links across distributed file systems.                                                    |
| 2947 |                | • To ensure that during path name resolution an application can rely on the                        |
| 2948 |                | ability to follow at least {SYMLOOP_MAX} symbolic links in the absence                             |
| 2949 |                | of a loop.                                                                                         |
| 2950 |                | • To allow implementations to provide the capability of traversing more                            |
| 2951 |                | than {SYMLOOP_MAX} symbolic links in the absence of a loop.                                        |
| 2952 |                | • To allow implementations to detect loops and generate the error prior to                         |
| 2953 |                | encountering {SYMLOOP_MAX} symbolic links.                                                         |
| 2954 | [ENAMETOOLONG] |                                                                                                    |
| 2955 |                | When a symbolic link is encountered during path name resolution, the                               |
| 2956 |                | contents of that symbolic link are used to create a new path name. The                             |
| 2957 |                | standard developers intended to allow, but not require, that implementations                       |
| 2958 |                | enforce the restriction of {PATH_MAX} on the result of this path name                              |
| 2959 |                | substitution.                                                                                      |
| 2960 | [ENOMEM]       | The term <i>main memory</i> is not used in POSIX.1 because it is implementation-                   |
| 2961 |                | defined.                                                                                           |
| 2962 | [ENOTSUP]      | This error code is to be used when an implementation chooses to implement                          |
| 2963 |                | the required functionality of IEEE Std. 1003.1-200x but does not support                           |
| 2964 |                | optional facilities defined by IEEE Std. 1003.1-200x. The return of [ENOSYS] is                    |
| 2965 |                | to be taken to indicate that the function of the interface is not supported at all;                |
| 2966 |                | the function will always fail with this error code.                                                |

2967 [ENOTTY] The symbolic name for this error is derived from a time when device control  
 2968 was done by *ioctl()* and that operation was only permitted on a terminal  
 2969 interface. The term *TTY* is derived from *teletypewriter*, the devices to which  
 2970 this error originally applied.

2971 [EPIPE] This condition normally generates the signal SIGPIPE; the error is returned if  
 2972 the signal does not terminate the process.

2973 [EROFS] In historical implementations, attempting to *unlink()* or *rmdir()* a mount point  
 2974 would generate an [EBUSY] error. An implementation could be envisioned  
 2975 where such an operation could be performed without error. In this case, if  
 2976 *either* the directory entry or the actual data structures reside on a read-only file  
 2977 system, [EROFS] is the appropriate error to generate. (For example, changing  
 2978 the link count of a file on a read-only file system could not be done, as is  
 2979 required by *unlink()*, and thus an error should be reported.)

2980 Three error numbers, [EDOM], [EILSEQ], and [ERANGE], were added to this section primarily  
 2981 for consistency with the ISO C standard.

### 2982 **Alternative Solutions for Per-Thread *errno***

2983 The usual implementation of *errno* as a single global variable does not work in a multi-threaded  
 2984 environment. In such an environment, a thread may make a POSIX.1 call and get a -1 error  
 2985 return, but before that thread can check the value of *errno*, another thread might have made a  
 2986 second POSIX.1 call that also set *errno*. This behavior is unacceptable in robust programs. There  
 2987 were a number of alternatives that were considered for handling the *errno* problem:

- 2988 • Implement *errno* as a per-thread integer variable.
- 2989 • Implement *errno* as a service that can access the per-thread error number.
- 2990 • Change all POSIX.1 calls to accept an extra status argument and avoid setting *errno*.
- 2991 • Change all POSIX.1 calls to raise a language exception.

2992 The first option offers the highest level of compatibility with existing practice but requires  
 2993 special support in the linker, compiler, and/or virtual memory system to support the new  
 2994 concept of thread private variables. When compared with current practice, the third and fourth  
 2995 options are much cleaner, more efficient, and encourage a more robust programming style, but  
 2996 they require new versions of all of the POSIX.1 functions that might detect an error. The second  
 2997 option offers compatibility with existing code that uses the `<errno.h>` header to define the  
 2998 symbol *errno*. In this option, *errno* may be a macro defined:

```
2999 #define errno (*__errno())
3000 extern int *__errno();
```

3001 This option may be implemented as a per-thread variable whereby an *errno* field is allocated in  
 3002 the user space object representing a thread, and whereby the function *\_\_errno()* makes a system  
 3003 call to determine the location of its user space object and returns the address of the *errno* field of  
 3004 that object. Another implementation, one that avoids calling the kernel, involves allocating  
 3005 stacks in chunks. The stack allocator keeps a side table indexed by chunk number containing a  
 3006 pointer to the thread object that uses that chunk. The *\_\_errno()* function then looks at the stack  
 3007 pointer, determines the chunk number, and uses that as an index into the chunk table to find its  
 3008 thread object and thus its private value of *errno*. On most architectures, this can be done in four  
 3009 to five instructions. Some compilers may wish to implement *\_\_errno()* inline to improve  
 3010 performance.

**3011 Disallowing Return of the [EINTR] Error Code**

3012 Many blocking interfaces defined by IEEE Std. 1003.1-200x may return [EINTR] if interrupted  
3013 during their execution by a signal handler. Blocking interfaces introduced under the Threads  
3014 option do not have this property. Instead, they require that the interface appear to be atomic  
3015 with respect to interruption. In particular, clients of block interfaces need not handle any  
3016 possible [EINTR] return as a special case since it will never occur. If it is necessary to restart  
3017 operations or complete incomplete operations following the execution of a signal handler, this is  
3018 handled by the implementation, rather than by the application.

3019 Requiring applications to handle [EINTR] errors on blocking interfaces has been shown to be a  
3020 frequent source of often unreproducible bugs, and it adds no compelling value to the available  
3021 functionality. Thus, blocking interfaces introduced for use by multi-threaded programs do not  
3022 use this paradigm. In particular, in none of the functions *flockfile()*, *pthread\_cond\_timedwait()*,  
3023 *pthread\_cond\_wait()*, *pthread\_join()*, *pthread\_mutex\_lock()*, and *sigwait()* did providing [EINTR]  
3024 returns add value, or even particularly make sense. Thus, these functions do not provide for an  
3025 [EINTR] return, even when interrupted by a signal handler. The same arguments can be applied  
3026 to *sem\_wait()*, *sem\_trywait()*, *sigwaitinfo()*, and *sigtimedwait()*, but implementations are  
3027 permitted to return [EINTR] error codes for these functions for compatibility with earlier  
3028 versions of IEEE Std. 1003.1-200x. Applications cannot rely on calls to these functions returning  
3029 [EINTR] error codes when signals are delivered to the calling thread, but they should allow for  
3030 the possibility.

**3031 B.2.3.1 Additional Error Numbers**

3032 The ISO C standard defines the name space for implementations to add additional error  
3033 numbers.

**3034 B.2.4 Signal Concepts**

3035 Historical implementations of signals, using the *signal()* function, have shortcomings that make  
3036 them unreliable for many application uses. Because of this, a new signal mechanism, based very  
3037 closely on the one of 4.2 BSD and 4.3 BSD, was added to POSIX.1.

**3038 Signal Names**

3039 The restriction on the actual type used for **sigset\_t** is intended to guarantee that these objects can  
3040 always be assigned, have their address taken, and be passed as parameters by value. It is not  
3041 intended that this type be a structure including pointers to other data structures, as that could  
3042 impact the portability of applications performing such operations. A reasonable implementation  
3043 could be a structure containing an array of some integer type.

3044 The signals described in IEEE Std. 1003.1-200x must have unique values so that they may be  
3045 named as parameters of **case** statements in the body of a C language **switch** clause. However,  
3046 implementation-defined signals may have values that overlap with each other or with signals  
3047 specified in IEEE Std. 1003.1-200x. An example of this is SIGABRT, which traditionally overlaps  
3048 some other signal, such as SIGIOT.

3049 SIGKILL, SIGTERM, SIGUSR1, and SIGUSR2 are ordinarily generated only through the explicit  
3050 use of the *kill()* function, although some implementations generate SIGKILL under  
3051 extraordinary circumstances. SIGTERM is traditionally the default signal sent by the *kill*  
3052 command.

3053 The signals SIGBUS, SIGEMT, SIGIOT, SIGTRAP, and SIGSYS were omitted from POSIX.1  
3054 because their behavior is implementation-defined and could not be adequately categorized.  
3055 Conforming implementations may deliver these signals, but must document the circumstances

3056 under which they are delivered and note any restrictions concerning their delivery. The signals  
3057 SIGFPE, SIGILL, and SIGSEGV are similar in that they also generally result only from  
3058 programming errors. They were included in POSIX.1 because they do indicate three relatively  
3059 well-categorized conditions. They are all defined by the ISO C standard and thus would have to  
3060 be defined by any system with a ISO C standard binding, even if not explicitly included in  
3061 POSIX.1.

3062 There is very little that a Conforming POSIX.1 Application can do by catching, ignoring, or  
3063 masking any of the signals SIGILL, SIGTRAP, SIGIOT, SIGEMT, SIGBUS, SIGSEGV, SIGSYS, or  
3064 SIGFPE. They will generally be generated by the system only in cases of programming errors.  
3065 While it may be desirable for some robust code (for example, a library routine) to be able to  
3066 detect and recover from programming errors in other code, these signals are not nearly sufficient  
3067 for that purpose. One portable use that does exist for these signals is that a command interpreter  
3068 can recognize them as the cause of a process' termination (with *wait()*) and print an appropriate  
3069 message. The mnemonic tags for these signals are derived from their PDP-11 origin.

3070 The signals SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, and SIGCONT are provided for job control  
3071 and are unchanged from 4.2 BSD. The signal SIGCHLD is also typically used by job control  
3072 shells to detect children that have terminated or, as in 4.2 BSD, stopped.

3073 Some implementations, including System V, have a signal named SIGCLD, which is similar to  
3074 SIGCHLD in 4.2 BSD. POSIX.1 permits implementations to have a single signal with both  
3075 names. POSIX.1 carefully specifies ways in which portable applications can avoid the semantic  
3076 differences between the two different implementations. The name SIGCHLD was chosen for  
3077 POSIX.1 because most current application usages of it can remain unchanged in conforming  
3078 applications. SIGCLD in System V has more cases of semantics that POSIX.1 does not specify,  
3079 and thus applications using it are more likely to require changes in addition to the name change.

3080 The signals SIGUSR1 and SIGUSR2 are commonly used by applications for notification of  
3081 exceptional behavior and are described as "reserved as application-defined" so that such use is  
3082 not prohibited. Implementations should not generate SIGUSR1 or SIGUSR2, except when  
3083 explicitly requested by *kill()*. It is recommended that libraries not use these two signals, as such  
3084 use in libraries could interfere with their use by applications calling the libraries. If such use is  
3085 unavoidable, it should be documented. It is prudent for non-portable libraries to use non-  
3086 standard signals to avoid conflicts with use of standard signals by portable libraries.

3087 There is no portable way for an application to catch or ignore non-standard signals. Some  
3088 implementations define the range of signal numbers, so applications can install signal-catching  
3089 functions for all of them. Unfortunately, implementation-defined signals often cause problems  
3090 when caught or ignored by applications that do not understand the reason for the signal. While  
3091 the desire exists for an application to be more robust by handling all possible signals (even those  
3092 only generated by *kill()*), no existing mechanism was found to be sufficiently portable to include  
3093 in POSIX.1. The value of such a mechanism, if included, would be diminished given that  
3094 SIGKILL would still not be catchable.

3095 A number of new signal numbers are reserved for applications because the two user signals  
3096 defined by POSIX.1 are insufficient for many realtime applications. A range of signal numbers is  
3097 specified, rather than an enumeration of additional reserved signal names, because different  
3098 applications and application profiles will require a different number of application signals. It is  
3099 not desirable to burden all application domains and therefore all implementations with the  
3100 maximum number of signals required by all possible applications. Note that in this context,  
3101 signal numbers are essentially different signal priorities.

3102 The relatively small number of required additional signals, `{_POSIX_RTSIG_MAX}`, was chosen  
3103 so as not to require an unreasonably large signal mask/set. While this number of signals defined  
3104 in POSIX.1 will fit in a single 32-bit word signal mask, it is recognized that most existing

3105 implementations define many more signals than are specified in POSIX.1 and, in fact, many  
3106 implementations have already exceeded 32 signals (including the “null signal”). Support of  
3107 `{_POSIX_RTSIG_MAX}` additional signals may push some implementation over the single 32-bit  
3108 word line, but is unlikely to push any implementations that are already over that line beyond the  
3109 64-signal line.

#### 3110 *B.2.4.1 Signal Generation and Delivery*

3111 The terms defined in this section are not used consistently in documentation of historical  
3112 systems. Each signal can be considered to have a lifetime beginning with *generation* and ending  
3113 with *delivery* or *acceptance*. The POSIX.1 definition of *delivery* does not exclude ignored signals;  
3114 this is considered a more consistent definition. This revised text in several parts of  
3115 IEEE Std. 1003.1-200x clarifies the distinct semantics of asynchronous signal *delivery* and  
3116 synchronous signal *acceptance*. The previous wording attempted to categorize both under the  
3117 term *delivery*, which led to conflicts over whether the effects of asynchronous signal delivery  
3118 applied to synchronous signal acceptance.

3119 Signals generated for a process are delivered to only one thread. Thus, if more than one thread is  
3120 eligible to receive a signal, one has to be chosen. The choice of threads is left entirely up to the  
3121 implementation both to allow the widest possible range of conforming implementations and to  
3122 give implementations the freedom to deliver the signal to the “easiest possible” thread should  
3123 there be differences in ease of delivery between different threads.

3124 Note that should multiple delivery among cooperating threads be required by an application,  
3125 this can be trivially constructed out of the provided single-delivery semantics. The construction  
3126 of a `sigwait_multiple()` function that accomplishes this goal is presented with the rationale for  
3127 `sigwaitinfo()`.

3128 Implementations should deliver unblocked signals as soon after they are generated as possible.  
3129 However, it is difficult for POSIX.1 to make specific requirements about this, beyond those in  
3130 `kill()` and `sigprocmask()`. Even on systems with prompt delivery, scheduling of higher priority  
3131 processes is always likely to cause delays.

3132 In general, the interval between the generation and delivery of unblocked signals cannot be  
3133 detected by an application. Thus, references to pending signals generally apply to blocked,  
3134 pending signals. An implementation registers a signal as pending on the process when no thread  
3135 has the signal unblocked and there are no threads blocked in a `sigwait()` function for that signal.  
3136 Thereafter, the implementation delivers the signal to the first thread that unblocks the signal or  
3137 calls a `sigwait()` function on a signal set containing this signal rather than choosing the recipient  
3138 thread at the time the signal is sent.

3139 In the 4.3 BSD system, signals that are blocked and set to `SIG_IGN` are discarded immediately  
3140 upon generation. For a signal that is ignored as its default action, if the action is `SIG_DFL` and  
3141 the signal is blocked, a generated signal remains pending. In the 4.1 BSD system and in  
3142 System V, Release 3, two other implementations that support a somewhat similar signal  
3143 mechanism, all ignored, blocked signals remain pending if generated. Because it is not normally  
3144 useful for an application to simultaneously ignore and block the same signal, it was unnecessary  
3145 for POSIX.1 to specify behavior that would invalidate any of the historical implementations.

3146 There is one case in some historical implementations where an unblocked, pending signal does  
3147 not remain pending until it is delivered. In the System V implementation of `signal()`, pending  
3148 signals are discarded when the action is set to `SIG_DFL` or a signal-catching routine (as well as to  
3149 `SIG_IGN`). Except in the case of setting `SIGCHLD` to `SIG_DFL`, implementations that do this do  
3150 not conform completely to POSIX.1. Some earlier proposals for POSIX.1 explicitly stated this,  
3151 but these statements were redundant due to the requirement that functions defined by POSIX.1  
3152 not change attributes of processes defined by POSIX.1 except as explicitly stated.

3153 POSIX.1 specifically states that the order in which multiple, simultaneously pending signals are  
 3154 delivered is unspecified. This order has not been explicitly specified in historical  
 3155 implementations, but has remained quite consistent and been known to those familiar with the  
 3156 implementations. Thus, there have been cases where applications (usually system utilities) have  
 3157 been written with explicit or implicit dependencies on this order. Implementors and others  
 3158 porting existing applications may need to be aware of such dependencies.

3159 When there are multiple pending signals that are not blocked, implementations should arrange  
 3160 for the delivery of all signals at once, if possible. Some implementations stack calls to all pending  
 3161 signal-catching routines, making it appear that each signal-catcher was interrupted by the next  
 3162 signal. In this case, the implementation should ensure that this stacking of signals does not  
 3163 violate the semantics of the signal masks established by *sigaction()*. Other implementations  
 3164 process at most one signal when the operating system is entered, with remaining signals saved  
 3165 for later delivery. Although this practice is widespread, this behavior is neither standardized  
 3166 nor endorsed. In either case, implementations should attempt to deliver signals associated with  
 3167 the current state of the process (for example, SIGFPE) before other signals, if possible.

3168 In 4.2 BSD and 4.3 BSD, it is not permissible to ignore or explicitly block SIGCONT, because if  
 3169 blocking or ignoring this signal prevented it from continuing a stopped process, such a process  
 3170 could never be continued (only killed by SIGKILL). However, 4.2 BSD and 4.3 BSD do block  
 3171 SIGCONT during execution of its signal-catching function when it is caught, creating exactly  
 3172 this problem. A proposal was considered to disallow catching SIGCONT in addition to ignoring  
 3173 and blocking it, but this limitation led to objections. The consensus was to require that  
 3174 SIGCONT always continue a stopped process when generated. This removed the need to  
 3175 disallow ignoring or explicit blocking of the signal; note that SIG\_IGN and SIG\_DFL are  
 3176 equivalent for SIGCONT .

#### 3177 B.2.4.2 Realtime Signal Generation and Delivery

3178 The Realtime Signals Extension option to POSIX.1 signal generation and delivery behavior is  
 3179 required for the following reasons:

- 3180 • The **sigevent** structure is used by other POSIX.1 functions that result in asynchronous event  
 3181 notifications to specify the notification mechanism to use and other information needed by  
 3182 the notification mechanism. IEEE Std. 1003.1-200x defines only three symbolic values for the  
 3183 notification mechanism. SIGEV\_NONE is used to indicate that no notification is required  
 3184 when the event occurs. This is useful for applications that use asynchronous I/O with polling  
 3185 for completion. SIGEV\_SIGNAL indicates that a signal shall be generated when the event  
 3186 occurs. SIGEV\_NOTIFY provides for “callback functions” for asynchronous notifications  
 3187 done by a function call within the context of a new thread. This provides a multi-threaded  
 3188 process a more natural means of notification than signals. The primary difficulty with  
 3189 previous notification approaches has been to specify the environment of the notification  
 3190 routine.

- 3191 — One approach is to limit the notification routine to call only functions permitted in a  
 3192 signal handler. While the list of permissible functions is clearly stated, this is overly  
 3193 restrictive.

- 3194 — A second approach is to define a new list of functions or classes of functions that are  
 3195 explicitly permitted or not permitted. This would give a programmer more lists to deal  
 3196 with, which would be awkward.

- 3197 — The third approach is to define completely the environment for execution of the  
 3198 notification function. A clear definition of an execution environment for notification is  
 3199 provided by executing the notification function in the environment of a newly created  
 3200 thread.

- 3201 Implementations may support additional notification mechanisms by defining new values  
3202 for *sigev\_notify*.
- 3203 For a notification type of SIGEV\_SIGNAL, the other members of the **sigevent** structure  
3204 defined by IEEE Std. 1003.1-200x specify the realtime signal—that is, the signal number and  
3205 application-defined value that differentiates between occurrences of signals with the same  
3206 number—that will be generated when the event occurs. The structure is defined in  
3207 <**signal.h**>, even though the structure is not directly used by any of the signal functions,  
3208 because it is part of the signals interface used by the POSIX.1b “client functions”. When the  
3209 client functions include <**signal.h**> to define the signal names, the **sigevent** structure will  
3210 also be defined.
- 3211 An application-defined value passed to the signal handler is used to differentiate between  
3212 different “events” instead of requiring that the application use different signal numbers for  
3213 several reasons:
- 3214 — Realtime applications potentially handle a very large number of different events.  
3215 Requiring that implementations support a correspondingly large number of distinct  
3216 signal numbers will adversely impact the performance of signal delivery because the  
3217 signal masks to be manipulated on entry and exit to the handlers will become large.
  - 3218 — Event notifications are prioritized by signal number (the rationale for this is explained in  
3219 the following paragraphs) and the use of different signal numbers to differentiate  
3220 between the different event notifications overloads the signal number more than has  
3221 already been done. It also requires that the application writer make arbitrary assignments  
3222 of priority to events that are logically of equal priority.
- 3223 A union is defined for the application-defined value so that either an integer constant or a  
3224 pointer can be portably passed to the signal-catching function. On some architectures a  
3225 pointer cannot be cast to an **int** and *vice versa*.
- 3226 Use of a structure here with an explicit notification type discriminant rather than explicit  
3227 parameters to realtime functions, or embedded in other realtime structures, provides for  
3228 future extensions to IEEE Std. 1003.1-200x. Additional, perhaps more efficient, notification  
3229 mechanisms can be supported for existing realtime function interfaces, such as timers and  
3230 asynchronous I/O, by extending the **sigevent** structure appropriately. The existing realtime  
3231 function interfaces will not have to be modified to use any such new notification mechanism.  
3232 The revised text concerning the SIGEV\_SIGNAL value makes consistent the semantics of the  
3233 members of the **sigevent** structure, particularly in the definitions of *lio\_listio()* and  
3234 *aio\_fsync()*. For uniformity, other revisions cause this specification to be referred to rather  
3235 than inaccurately duplicated in the descriptions of functions and structures using the  
3236 **sigevent** structure. The revised wording does not relax the requirement that the signal  
3237 number be in the range SIGRTMIN to SIGRTMAX to guarantee queuing and passing of the  
3238 application value, since that requirement is still implied by the signal names.
- 3239 • IEEE Std. 1003.1-200x is intentionally vague on whether “non-realtime” signal-generating  
3240 mechanisms can result in a **siginfo\_t** being supplied to the handler on delivery. In one  
3241 existing implementation, a **siginfo\_t** is posted on signal generation, even though the  
3242 implementation does not support queuing of multiple occurrences of a signal. It is not the  
3243 intent of IEEE Std. 1003.1-200x to preclude this, independent of the mandate to define signals  
3244 that do support queuing. Any interpretation that appears to preclude this is a mistake in the  
3245 reading or writing of the standard.
  - 3246 • Signals handled by realtime signal handlers might be generated by functions or conditions  
3247 that do not allow the specification of an application-defined value and do not queue.  
3248 IEEE Std. 1003.1-200x specifies the *si\_code* member of the **siginfo\_t** structure used in existing

3249 practice and defines additional codes so that applications can detect whether an application-  
 3250 defined value is present or not. The code SI\_USER for *kill()*-generated signals is adopted  
 3251 from existing practice.

3252 • The *sigaction()* *sa\_flags* value SA\_SIGINFO tells the implementation that the signal-catching  
 3253 function expects two additional arguments. When the flag is not set, a single argument, the  
 3254 signal number, is passed as specified by IEEE Std. 1003.1-200x. Although  
 3255 IEEE Std. 1003.1-200x does not explicitly allow the *info* argument to the handler function to  
 3256 be NULL, this is existing practice. This provides for compatibility with programs whose  
 3257 signal-catching functions are not prepared to accept the additional arguments.  
 3258 IEEE Std. 1003.1-200x is explicitly unspecified as to whether signals actually queue when  
 3259 SA\_SIGINFO is not set for a signal, as there appear to be no benefits to applications in  
 3260 specifying one behavior or another. One existing implementation queues a **siginfo\_t** on each  
 3261 signal generation, unless the signal is already pending, in which case the implementation  
 3262 discards the new **siginfo\_t**; that is, the queue length is never greater than one. This  
 3263 implementation only examines SA\_SIGINFO on signal delivery, discarding the queued  
 3264 **siginfo\_t** if its delivery was not requested.

3265 IEEE Std. 1003.1-200x specifies several new values for the *si\_code* member of the **siginfo\_t**  
 3266 structure. In existing practice, a *si\_code* value of less than or equal to zero indicates that  
 3267 the signal was generated by a process via the *kill()* function. In existing practice, values of *si\_code*  
 3268 that provide additional information for implementation-generated signals, such as SIGFPE or  
 3269 SIGSEGV, are all positive. Thus, if implementations define the new constants specified in  
 3270 IEEE Std. 1003.1-200x to be negative numbers, programs written to use existing practice will  
 3271 not break. IEEE Std. 1003.1-200x chose not to attempt to specify existing practice values of  
 3272 *si\_code* other than SI\_USER both because it was deemed beyond the scope of  
 3273 IEEE Std. 1003.1-200x and because many of the values in existing practice appear to be  
 3274 platform and implementation-defined. But, IEEE Std. 1003.1-200x does specify that if an  
 3275 implementation—for example, one that does not have existing practice in this area—chooses  
 3276 to define additional values for *si\_code*, these values have to be different from the values of the  
 3277 symbols specified by IEEE Std. 1003.1-200x. This will allow portable applications to  
 3278 differentiate between signals generated by one of the POSIX.1b asynchronous events and  
 3279 those generated by other implementation events in a manner compatible with existing  
 3280 practice.

3281 The unique values of *si\_code* for the POSIX.1b asynchronous events have implications for  
 3282 implementations of, for example, asynchronous I/O or message passing in user space library  
 3283 code. Such an implementation will be required to provide a hidden interface to the signal  
 3284 generation mechanism that allows the library to specify the standard values of *si\_code*.

3285 Existing practice also defines additional members of **siginfo\_t**, such as the process ID and  
 3286 user ID of the sending process for *kill()*-generated signals. These members were deemed not  
 3287 necessary to meet the requirements of realtime applications and are not specified by  
 3288 IEEE Std. 1003.1-200x. Neither are they precluded.

3289 The third argument to the signal-catching function, *context*, is left undefined by  
 3290 IEEE Std. 1003.1-200x, but is specified in the interface because it matches existing practice for  
 3291 the SA\_SIGINFO flag. It was considered undesirable to require a separate implementation  
 3292 for SA\_SIGINFO for POSIX conformance on implementations that already support the two  
 3293 additional parameters.

3294 • The requirement to deliver lower numbered signals in the range SIGRTMIN to SIGRTMAX  
 3295 first, when multiple unblocked signals are pending, results from several considerations:

3296 — A method is required to prioritize event notifications. The signal number was chosen  
 3297 instead of, for instance, associating a separate priority with each request, because an



3298 implementation has to check pending signals at various points and select one for delivery  
 3299 when more than one is pending. Specifying a selection order is the minimal additional  
 3300 semantic that will achieve prioritized delivery. If a separate priority were to be associated  
 3301 with queued signals, it would be necessary for an implementation to search all non-  
 3302 empty, non-blocked signal queues and select from among them the pending signal with  
 3303 the highest priority. This would significantly increase the cost of and decrease the  
 3304 determinism of signal delivery.

3305 — Given the specified selection of the lowest numeric unblocked pending signal,  
 3306 preemptive priority signal delivery can be achieved using signal numbers and signal  
 3307 masks by ensuring that the *sa\_mask* for each signal number blocks all signals with a  
 3308 higher numeric value.

3309 For realtime applications that want to use only the newly defined realtime signal numbers  
 3310 without interference from the standard signals, this can be achieved by blocking all of the  
 3311 standard signals in the process signal mask and in the *sa\_mask* installed by the signal  
 3312 action for the realtime signal handlers.

3313 IEEE Std. 1003.1-200x explicitly leaves unspecified the ordering of signals outside of the  
 3314 range of realtime signals and the ordering of signals within this range with respect to those  
 3315 outside the range. It was believed that this would unduly constrain implementations or  
 3316 standards in the future definition of new signals.

### 3317 B.2.4.3 Signal Actions

3318 Early proposals mentioned SIGCONT as a second exception to the rule that signals are not  
 3319 delivered to stopped processes until continued. Because IEEE Std. 1003.1-200x now specifies that  
 3320 SIGCONT causes the stopped process to continue when it is generated, delivery of SIGCONT is  
 3321 not prevented because a process is stopped, even without an explicit exception to this rule.

3322 Ignoring a signal by setting the action to SIG\_IGN (or SIG\_DFL for signals whose default action  
 3323 is to ignore) is not the same as installing a signal-catching function that simply returns. Invoking  
 3324 such a function will interrupt certain system functions that block processes (for example, *wait()*,  
 3325 *sigsuspend()*, *pause()*, *read()*, *write()*) while ignoring a signal has no such effect on the process.

3326 Historical implementations discard pending signals when the action is set to SIG\_IGN.  
 3327 However, they do not always do the same when the action is set to SIG\_DFL and the default  
 3328 action is to ignore the signal. IEEE Std. 1003.1-200x requires this for the sake of consistency and  
 3329 also for completeness, since the only signal this applies to is SIGCHLD, and  
 3330 IEEE Std. 1003.1-200x disallows setting its action to SIG\_IGN.

3331 The specification of the effects of SIG\_IGN on SIGCHLD as implementation-defined permits,  
 3332 but does not require, the System V effect of causing terminating children to be ignored by *wait()*.  
 3333 Yet it permits SIGCHLD to be effectively ignored in an implementation-defined manner by use  
 3334 of SIG\_DFL.

3335 Some implementations (System V, for example) assign different semantics for SIGCLD  
 3336 depending on whether the action is set to SIG\_IGN or SIG\_DFL. Since POSIX.1 requires that the  
 3337 default action for SIGCHLD be to ignore the signal, applications should always set the action to  
 3338 SIG\_DFL in order to avoid SIGCHLD.

3339 Some implementations (System V, for example) will deliver a SIGCLD signal immediately when  
 3340 a process establishes a signal-catching function for SIGCLD when that process has a child that  
 3341 has already terminated. Other implementations, such as 4.3 BSD, do not generate a new  
 3342 SIGCHLD signal in this way. In general, a process should not attempt to alter the signal action  
 3343 for the SIGCHLD signal while it has any outstanding children. However, it is not always  
 3344 possible for a process to avoid this; for example, shells sometimes start up processes in pipelines

3345 with other processes from the pipeline as children. Processes that cannot ensure that they have  
3346 no children when altering the signal action for SIGCHLD thus need to be prepared for, but not  
3347 depend on, generation of an immediate SIGCHLD signal.

3348 The default action of the stop signals (SIGSTOP , SIGTSTP, SIGTTIN, SIGTTOU) is to stop a  
3349 process that is executing. If a stop signal is delivered to a process that is already stopped, it has  
3350 no effect. In fact, if a stop signal is generated for a stopped process whose signal mask blocks the  
3351 signal, the signal will never be delivered to the process since the process must receive a  
3352 SIGCONT, which discards all pending stop signals, in order to continue executing.

3353 The SIGCONT signal shall continue a stopped process even if SIGCONT is blocked (or ignored).  
3354 However, if a signal-catching routine has been established for SIGCONT, it will not be entered  
3355 until SIGCONT is unblocked.

3356 If a process in an orphaned process group stops, it is no longer under the control of a job control  
3357 shell and hence would not normally ever be continued. Because of this, orphaned processes that  
3358 receive terminal-related stop signals (SIGTSTP , SIGTTIN, SIGTTOU, but not SIGSTOP ) must  
3359 not be allowed to stop. The goal is to prevent stopped processes from languishing forever. (As  
3360 SIGSTOP is sent only via *kill()*, it is assumed that the process or user sending a SIGSTOP can  
3361 send a SIGCONT when desired.) Instead, the system must discard the stop signal. As an  
3362 extension, it may also deliver another signal in its place. 4.3 BSD sends a SIGKILL, which is  
3363 overly effective because SIGKILL is not catchable. Another possible choice is SIGHUP. 4.3 BSD  
3364 also does this for orphaned processes (processes whose parent has terminated) rather than for  
3365 members of orphaned process groups; this is less desirable because job control shells manage  
3366 process groups. POSIX.1 also prevents SIGTTIN and SIGTTOU signals from being generated for  
3367 processes in orphaned process groups as a direct result of activity on a terminal, preventing  
3368 infinite loops when *read()* and *write()* calls generate signals that are discarded; see Section  
3369 A.11.1.4 (on page 3371). A similar restriction on the generation of SIGTSTP was considered, but  
3370 that would be unnecessary and more difficult to implement due to its asynchronous nature.

3371 Although POSIX.1 requires that signal-catching functions be called with only one argument,  
3372 there is nothing to prevent conforming implementations from extending POSIX.1 to pass  
3373 additional arguments, as long as Strictly Conforming POSIX.1 Applications continue to compile  
3374 and execute correctly. Most historical implementations do, in fact, pass additional, signal-  
3375 specific arguments to certain signal-catching routines.

3376 There was a proposal to change the declared type of the signal handler to:

```
3377 void func (int sig, ...);
```

3378 The usage of ellipses ("...") is ISO C standard syntax to indicate a variable number of  
3379 arguments. Its use was intended to allow the implementation to pass additional information to  
3380 the signal handler in a standard manner.

3381 Unfortunately, this construct would require all signal handlers to be defined with this syntax  
3382 because the ISO C standard allows implementations to use a different parameter passing  
3383 mechanism for variable parameter lists than for non-variable parameter lists. Thus, all existing  
3384 signal handlers in all existing applications would have to be changed to use the variable syntax  
3385 in order to be standard and portable. This is in conflict with the goal of Minimal Changes to  
3386 Existing Application Code.

3387 When terminating a process from a signal-catching function, processes should be aware of any  
3388 interpretation that their parent may make of the status returned by *wait()* or *waitpid()*. In  
3389 particular, a signal-catching function should not call *exit(0)* or *\_exit(0)* unless it wants to indicate  
3390 successful termination. A non-zero argument to *exit()* or *\_exit()* can be used to indicate  
3391 unsuccessful termination. Alternatively, the process can use *kill()* to send itself a fatal signal  
3392 (first ensuring that the signal is set to the default action and not blocked). See also the

3393 RATIONALE section of the `_exit()` function.

3394 The behavior of *unsafe* functions, as defined by this section, is undefined when they are invoked  
3395 from signal-catching functions in certain circumstances. The behavior of reentrant functions, as  
3396 defined by this section, is as specified by POSIX.1, regardless of invocation from a signal-  
3397 catching function. This is the only intended meaning of the statement that reentrant functions  
3398 may be used in signal-catching functions without restriction. Applications must still consider all  
3399 effects of such functions on such things as data structures, files, and process state. In particular,  
3400 application writers need to consider the restrictions on interactions when interrupting `sleep()`  
3401 (see `sleep()`) and interactions among multiple handles for a file description. The fact that any  
3402 specific function is listed as reentrant does not necessarily mean that invocation of that function  
3403 from a signal-catching function is recommended.

3404 In order to prevent errors arising from interrupting non-reentrant function calls, applications  
3405 should protect calls to these functions either by blocking the appropriate signals or through the  
3406 use of some programmatic semaphore. POSIX.1 does not address the more general problem of  
3407 synchronizing access to shared data structures. Note in particular that even the “safe” functions  
3408 may modify the global variable `errno`; the signal-catching function may want to save and restore  
3409 its value. The same principles apply to the reentrancy of application routines and asynchronous  
3410 data access.

3411 Note that `longjmp()` and `siglongjmp()` are not in the list of reentrant functions. This is because the  
3412 code executing after `longjmp()` or `siglongjmp()` can call any unsafe functions with the same  
3413 danger as calling those unsafe functions directly from the signal handler. Applications that use  
3414 `longjmp()` or `siglongjmp()` out of signal handlers require rigorous protection in order to be  
3415 portable. Many of the other functions that are excluded from the list are traditionally  
3416 implemented using either the C language `malloc()` or `free()` functions or the ISO C standard I/O  
3417 library, both of which traditionally use data structures in a non-reentrant manner. Because any  
3418 combination of different functions using a common data structure can cause reentrancy  
3419 problems, POSIX.1 does not define the behavior when any unsafe function is called in a signal  
3420 handler that interrupts any unsafe function.

3421 The only realtime extension to signal actions is the addition of the additional parameters to the  
3422 signal-catching function. This extension has been explained and motivated in the previous  
3423 section. In making this extension, though, developers of POSIX.1b ran into issues relating to  
3424 function prototypes. In response to input from the POSIX.1 standard developers, members were  
3425 added to the **sigaction** structure to specify function prototypes for the newer signal-catching  
3426 function specified by POSIX.1b. These members follow changes that are being made to POSIX.1.  
3427 Note that IEEE Std. 1003.1-200x explicitly states that these fields may overlap so that a union can  
3428 be defined. This will enable existing implementations of POSIX.1 to maintain binary-  
3429 compatibility when these extensions are added.

3430 The **siginfo\_t** structure was adopted for passing the application-defined value to match existing  
3431 practice, but the existing practice has no provision for an application-defined value, so this was  
3432 added. Note that POSIX normally reserves the “\_t” type designation for opaque types. The  
3433 **siginfo\_t** structure breaks with this convention to follow existing practice and thus promote  
3434 portability. Standardization of the existing practice for the other members of this structure may  
3435 be addressed in the future.

3436 Although it is not explicitly visible to applications, there are additional semantics for signal  
3437 actions implied by queued signals and their interaction with other POSIX.1b realtime functions.  
3438 Specifically:

- 3439 • It is not necessary to queue signals whose action is `SIG_IGN`.

- 3440           • For implementations that support POSIX.1b timers, some interaction with the timer functions  
3441           at signal delivery is implied to manage the timer overrun count.

#### 3442 **B.2.4.4** *Signal Effects on Other Functions*

3443           The most common behavior of an interrupted function after a signal-catching function returns is  
3444           for the interrupted function to give an [EINTR] error. However, there are a number of specific  
3445           exceptions, including *sleep()* and certain situations with *read()* and *write()*.

3446           The historical implementations of many functions defined by IEEE Std. 1003.1-200x are not  
3447           interruptible, but delay delivery of signals generated during their execution until after they  
3448           complete. This is never a problem for functions that are guaranteed to complete in a short  
3449           (imperceptible to a human) period of time. It is normally those functions that can suspend a  
3450           process indefinitely or for long periods of time (for example, *wait()*, *pause()*, *sigsuspend()*, *sleep()*,  
3451           or *read()/write()* on a slow device like a terminal] that are interruptible. This permits  
3452           applications to respond to interactive signals or to set timeouts on calls to most such functions  
3453           with *alarm()*. Therefore, implementations should generally make such functions (including ones  
3454           defined as extensions) interruptible.

3455           Functions not mentioned explicitly as interruptible may be so on some implementations,  
3456           possibly as an extension where the function gives an [EINTR] error. There are several functions  
3457           (for example, *getpid()*, *getuid()*) that are specified as never returning an error, which can thus  
3458           never be extended in this way.

### 3459 **B.2.5** **Standard I/O Streams**

#### 3460 **B.2.5.1** *Interaction of File Descriptors and Standard I/O Streams*

3461           There is no additional rationale for this section.

#### 3462 **B.2.5.2** *Stream Orientation and Encoding Rules*

3463           There is no additional rationale for this section.

### 3464 **B.2.6** **STREAMS**

3465           STREAMS are introduced into IEEE Std. 1003.1-200x as part of the alignment with the Single  
3466           UNIX Specification, but marked as an option in recognition that not all systems may wish to  
3467           implement the facility. The option within IEEE Std. 1003.1-200x is denoted by the XSR margin  
3468           marker. The standard developers made this option independent of the XSI option.

3469           STREAMS are a method of implementing network services and other character-based  
3470           input/output mechanisms, with the STREAM being a full-duplex connection between a process  
3471           and a device. STREAMS provides direct access to protocol modules, and optional protocol  
3472           modules can be interposed between the process-end of the STREAM and the device-driver at the  
3473           device-end of the STREAM. Pipes can be implemented using the STREAMS mechanism, so they  
3474           can provide process-to-process as well as process-to-device communications.

3475           This section introduces STREAMS I/O, the message types used to control them, an overview of  
3476           the priority mechanism, and the interfaces used to access them.

3477 **B.2.6.1** *Accessing STREAMS*

3478 There is no additional rationale for this section.

3479 **B.2.7** **XSI Interprocess Communication**

3480 There are two forms of IPC supported as options in IEEE Std. 1003.1-200x. The traditional  
3481 System V IPC routines derived from the SVID—that is, the *msg\**(), *sem\**(), and *shm\**()  
3482 interfaces—are mandatory on XSI-conformant systems. Thus, all XSI-conformant systems  
3483 provide the same mechanisms for manipulating messages, shared memory, and semaphores.

3484 In addition, the POSIX Realtime Extension provides an alternate set of routines for those systems  
3485 supporting the appropriate options.

3486 For maximum portability to UNIX systems, the former are recommended. However, if the  
3487 target for an application is a realtime system, then application developers are advised to write  
3488 their code in such a way that modules using IPC interfaces can be modified easily in the future  
3489 to use either interfaces.

3490 **B.2.7.1** *IPC General Information*

3491 General information that is shared by all three mechanisms is described in this section. The  
3492 common permissions mechanism is briefly introduced, describing the mode bits, and how they  
3493 are used to determine whether or not a process has access to read or write/alter the appropriate  
3494 instance of one of the IPC mechanisms. All other relevant information is contained in the  
3495 reference pages themselves.

3496 The semaphore type of IPC allows processes to communicate through the exchange of  
3497 semaphore values. A semaphore is a positive integer. Since many applications require the use of  
3498 more than one semaphore, XSI-conformant systems have the ability to create sets or arrays of  
3499 semaphores.

3500 Calls to support semaphores include:

3501 *semctl()*, *semget()*, *semop()*

3502 Semaphore sets are created by using the *semget()* function.

3503 The message type of IPC allows process to communicate through the exchange of data stored in  
3504 buffers. This data is transmitted between processes in discrete portions known as messages.

3505 Calls to support message queues include:

3506 *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

3507 The share memory type of IPC allows two or more processes to share memory and consequently  
3508 the data contained therein. This is done by allowing processes to set up access to a common  
3509 memory address space. This sharing of memory provides a fast means of exchange of data  
3510 between processes.

3511 Calls to support shared memory include:

3512 *shmctl()*, *shmdt()*, *shmget()*

3513 The *ftok()* interface is also provided.

3514 **B.2.8 Realtime**3515 **Advisory Information**

3516 POSIX.1b contains an Informative Annex with proposed interfaces for “real-time files”. These  
 3517 interfaces could determine groups of the exact parameters required to do “direct I/O” or  
 3518 “extents”. These interfaces were objected to by a significant portion of the balloting group as too  
 3519 complex. A portable application had little chance of correctly navigating the large parameter  
 3520 space to match its desires to the system. In addition, they only applied to a new type of file  
 3521 (realtime files) and they told the implementation exactly what to do as opposed to advising the  
 3522 implementation on application behavior and letting it optimize for the system the (portable)  
 3523 application was running on. For example, it was not clear how a system that had a disk array  
 3524 should set its parameters.

3525 There seemed to be several overall goals:

- 3526 • Optimizing sequential access
- 3527 • Optimizing caching behavior
- 3528 • Optimizing I/O data transfer
- 3529 • Preallocation

3530 The advisory interfaces, *posix\_fadvise()* and *posix\_madvise()*, satisfy the first two goals. The  
 3531 `POSIX_FADV_SEQUENTIAL` and `POSIX_MADV_SEQUENTIAL` advice tells the  
 3532 implementation to expect serial access. Typically the system will prefetch the next several serial  
 3533 accesses in order to overlap I/O. It may also free previously accessed serial data if memory is  
 3534 tight. If the application is not doing serial access it can use `POSIX_FADV_WILLNEED` and  
 3535 `POSIX_MADV_WILLNEED` to accomplish I/O overlap, as required. When the application  
 3536 advises `POSIX_FADV_RANDOM` or `POSIX_MADV_RANDOM` behavior, the implementation  
 3537 usually tries to fetch a minimum amount of data with each request and it does not expect much  
 3538 locality. `POSIX_FADV_DONTNEED` and `POSIX_MADV_DONTNEED` allow the system to free  
 3539 up caching resources as the data will not be required in the near future.

3540 `POSIX_FADV_NOREUSE` tells the system that caching the specified data is not optimal. For file  
 3541 I/O, the transfer should go directly to the user buffer instead of being cached internally by the  
 3542 implementation. To portably perform direct disk I/O on all systems, the application must  
 3543 perform its I/O transfers according to the following rules:

- 3544 1. The user buffer should be aligned according to the `{POSIX_REC_XFER_ALIGN} pathconf()`  
 3545 variable.
- 3546 2. The number of bytes transferred in an I/O operation should be a multiple of the  
 3547 `{POSIX_ALLOC_SIZE_MIN} pathconf()` variable.
- 3548 3. The offset into the file at the start of an I/O operation should be a multiple of the  
 3549 `{POSIX_ALLOC_SIZE_MIN} pathconf()` variable.
- 3550 4. The application should ensure that all threads which open a given file specify  
 3551 `POSIX_FADV_NOREUSE` to be sure that there is no unexpected interaction between  
 3552 threads using buffered I/O and threads using direct I/O to the same file.

3553 In some cases, a user buffer must be properly aligned in order to be transferred directly to/from  
 3554 the device. The `{POSIX_REC_XFER_ALIGN} pathconf()` variable tells the application the proper  
 3555 alignment.

3556 The preallocation goal is met by the space control function, *posix\_fallocate()*. The application can  
 3557 use *posix\_fallocate()* to guarantee no `[ENOSPC]` errors and to improve performance by prepaying

3558 any overhead required for block allocation.

3559 Implementations may use information conveyed by a previous *posix\_fadvise()* call to influence  
3560 the manner in which allocation is performed. For example, if an application did the following  
3561 calls:

```
3562 fd = open("file");
3563 posix_fadvise(fd, offset, len, POSIX_FADV_SEQUENTIAL);
3564 posix_fallocate(fd, len, size);
```

3565 an implementation might allocate the file contiguously on disk.

3566 Finally, the *pathconf()* variables {*POSIX\_REC\_MIN\_XFER\_SIZE*},  
3567 {*POSIX\_REC\_MAX\_XFER\_SIZE*}, and {*POSIX\_REC\_INCR\_XFER\_SIZE*} tell the application a  
3568 range of transfer sizes that are recommended for best I/O performance.

3569 Where bounded response time is required, the vendor can supply the appropriate settings of the  
3570 advisories to achieve a guaranteed performance level.

3571 The interfaces meet the goals while allowing applications using regular files to take advantage of  
3572 performance optimizations. The interfaces tell the implementation expected application  
3573 behavior which the implementation can use to optimize performance on a particular system  
3574 with a particular dynamic load.

3575 The *posix\_memalign()* function was added to allow for the allocation of specifically aligned  
3576 buffers; for example, for {*POSIX\_REC\_XFER\_ALIGN*}.

3577 The working group also considered the alternative of adding a function which would return an  
3578 aligned pointer to memory within a user supplied buffer. This was not considered to be the best  
3579 method, because it potentially wastes large amounts of memory when buffers need to be aligned  
3580 on large alignment boundaries.

### 3581 **Message Passing**

3582 This section provides the rationale for the definition of the message passing interface in  
3583 IEEE Std. 1003.1-200x. This is presented in terms of the objectives, models, and requirements  
3584 imposed upon this interface.

#### 3585 • Objectives

3586 Many applications, including both realtime and database applications, require a means of  
3587 passing arbitrary amounts of data between cooperating processes comprising the overall  
3588 application on one or more processors. Many conventional interfaces for interprocess  
3589 communication are insufficient for realtime applications in that efficient and deterministic  
3590 data passing methods cannot be implemented. This has prompted the definition of message  
3591 passing interfaces providing these facilities:

- 3592 — Open a message queue.
- 3593 — Send a message to a message queue.
- 3594 — Receive a message from a queue, either synchronously or asynchronously.
- 3595 — Alter message queue attributes for flow and resource control.

3596 It is assumed that an application may consist of multiple cooperating processes and that  
3597 these processes may wish to communicate and coordinate their activities. The message  
3598 passing facility described in IEEE Std. 1003.1-200x allows processes to communicate through  
3599 system-wide queues. These message queues are accessed through names that may be path  
3600 names. A message queue can be opened for use by multiple sending and/or multiple

- 3601 receiving processes.
- 3602 • Background on Embedded Applications
- 3603 Interprocess communication utilizing message passing is a key facility for the construction of  
3604 deterministic, high-performance realtime applications. The facility is present in all realtime  
3605 systems and is the framework upon which the application is constructed. The performance of  
3606 the facility is usually a direct indication of the performance of the resulting application.
- 3607 Realtime applications, especially for embedded systems, are typically designed around the  
3608 performance constraints imposed by the message passing mechanisms. Applications for  
3609 embedded systems are typically very tightly constrained. Application writers expect to  
3610 design and control the entire system. In order to minimize system costs, the writer will  
3611 attempt to use all resources to their utmost and minimize the requirement to add additional  
3612 memory or processors.
- 3613 The embedded applications usually share address spaces and only a simple message passing  
3614 mechanism is required. The application can readily access common data incurring only  
3615 mutual-exclusion overheads. The models desired are the simplest possible with the  
3616 application building higher-level facilities only when needed.
- 3617 • Requirements
- 3618 The following requirements determined the features of the message passing facilities defined  
3619 in IEEE Std. 1003.1-200x:
- 3620 — Naming of Message Queues
- 3621 The mechanism for gaining access to a message queue is a path name evaluated in a  
3622 context that is allowed to be a file system name space, or it can be independent of any file  
3623 system. This is a specific attempt to allow implementations based on either method in  
3624 order to address both embedded systems and to also allow implementation in larger  
3625 systems.
- 3626 The interface of *mq\_open()* is defined to allow but not require the access control and name  
3627 conflicts resulting from utilizing a file system for name resolution. All required behavior  
3628 is specified for the access control case. Yet a conforming implementation, such as an  
3629 embedded system kernel, may define that there are no distinctions between users and  
3630 may define that all process have all access privileges.
- 3631 — Embedded System Naming
- 3632 Embedded systems need to be able to utilize independent name spaces for accessing the  
3633 various system objects. They typically do not have a file system, precluding its utilization  
3634 as a common name resolution mechanism. The modularity of an embedded system limits  
3635 the connections between separate mechanisms that can be allowed.
- 3636 Embedded systems typically do not have any access protection. Since the system does not  
3637 support the mixing of applications from different areas, and usually does not even have  
3638 the concept of an authorization entity, access control is not useful.
- 3639 — Large System Naming
- 3640 On systems with more functionality, the name resolution must support the ability to use  
3641 the file system as the name resolution mechanism/object storage medium and to have  
3642 control over access to the objects. Utilizing the path name space can result in further  
3643 errors when the names conflict with other objects.
- 3644 — Fixed Size of Messages



3645 The interfaces impose a fixed upper bound on the size of messages that can be sent to a  
3646 specific message queue. The size is set on an individual queue basis and cannot be  
3647 changed dynamically.

3648 The purpose of the fixed size is to increase the ability of the system to optimize the  
3649 implementation of *mq\_send()* and *mq\_receive()*. With fixed sizes of messages and fixed  
3650 numbers of messages, specific message blocks can be pre-allocated. This eliminates a  
3651 significant amount of checking for errors and boundary conditions. Additionally, an  
3652 implementation can optimize data copying to maximize performance. Finally, with a  
3653 restricted range of message sizes, an implementation is better able to provide  
3654 deterministic operations.

#### 3655 — Prioritization of Messages

3656 Message prioritization allows the application to determine the order in which messages  
3657 are received. Prioritization of messages is a key facility that is provided by most realtime  
3658 kernels and is heavily utilized by the applications. The major purpose of having priorities  
3659 in message queues is to avoid priority inversions in the message system, where a high-  
3660 priority message is delayed behind one or more lower-priority messages. It has been  
3661 observed that a significant problem with Ada rendezvous is that it queues tasks in strict  
3662 FIFO order, ignoring priorities. This allows the applications to be designed so that they do  
3663 not need to be interrupted in order to change the flow of control when exceptional  
3664 conditions occur. The prioritization does add additional overhead to the message  
3665 operations in those cases it is actually used but a clever implementation can optimize for  
3666 the FIFO case to make that more efficient.

#### 3667 — Asynchronous Notification

3668 The interface supports the ability to have a task asynchronously notified of the  
3669 availability of a message on the queue. The purpose of this facility is to allow the task to  
3670 perform other functions and yet still be notified that a message has become available on  
3671 the queue.

3672 To understand the requirement for this function, it is useful to understand two models of  
3673 application design: a single task performing multiple functions and multiple tasks  
3674 performing a single function. Each of these models has advantages.

3675 Asynchronous notification is required to build the model of a single task performing  
3676 multiple operations. This model typically results from either the expectation that  
3677 interruption is less expensive than utilizing a separate task or from the growth of the  
3678 application to include additional functions.

### 3679 **Semaphores**

3680 Semaphores are a high-performance process synchronization mechanism. Semaphores are  
3681 named by null-terminated strings of characters.

3682 A semaphore is created using the *sem\_init()* function or the *sem\_open()* function with the  
3683 *O\_CREAT* flag set in *oflag*.

3684 To use a semaphore, a process has to first initialize the semaphore or inherit an open descriptor  
3685 for the semaphore via *fork()*.

3686 A semaphore preserves its state when the last reference is closed. For example, if a semaphore  
3687 has a value of 13 when the last reference is closed, it will have a value of 13 when it is next  
3688 opened.

3689 When a semaphore is created, an initial state for the semaphore has to be provided. This value is  
3690 a non-negative integer. Negative values are not possible since they indicate the presence of  
3691 blocked processes. The persistence of any of these objects across a system crash or a system  
3692 reboot is undefined. Conforming applications shall not depend on any sort of persistence across  
3693 a system reboot or a system crash.

3694 • Models and Requirements

3695 A realtime system requires synchronization and communication between the processes  
3696 comprising the overall application. An efficient and reliable synchronization mechanism has  
3697 to be provided in a realtime system that will allow more than one schedulable process  
3698 mutually-exclusive access to the same resource. This synchronization mechanism has to  
3699 allow for the optimal implementation of synchronization or systems implementors will  
3700 define other, more cost-effective methods.

3701 At issue are the methods whereby multiple processes (tasks) can be designed and  
3702 implemented to work together in order to perform a single function. This requires  
3703 interprocess communication and synchronization. A semaphore mechanism is the lowest  
3704 level of synchronization that can be provided by an operating system.

3705 A semaphore is defined as an object that has an integral value and a set of blocked processes  
3706 associated with it. If the value is positive or zero, then the set of blocked processes is empty;  
3707 otherwise, the size of the set is equal to the absolute value of the semaphore value. The value  
3708 of the semaphore can be incremented or decremented by any process with access to the  
3709 semaphore and must be done as an indivisible operation. When a semaphore value is less  
3710 than or equal to zero, any process that attempts to lock it again will block or be informed that  
3711 it is not possible to perform the operation.

3712 A semaphore may be used to guard access to any resource accessible by more than one  
3713 schedulable task in the system. It is a global entity and not associated with any particular  
3714 process. As such, a method of obtaining access to the semaphore has to be provided by the  
3715 operating system. A process that wants access to a critical resource (section) has to wait on  
3716 the semaphore that guards that resource. When the semaphore is locked on behalf of a  
3717 process, it knows that it can utilize the resource without interference by any other  
3718 cooperating process in the system. When the process finishes its operation on the resource,  
3719 leaving it in a well-defined state, it posts the semaphore, indicating that some other process  
3720 may now obtain the resource associated with that semaphore.

3721 In this section, mutexes and condition variables are specified as the synchronization  
3722 mechanisms between threads.

3723 These primitives are typically used for synchronizing threads that share memory in a single  
3724 process. However, this section provides an option allowing the use of these synchronization  
3725 interfaces and objects between processes that share memory, regardless of the method for  
3726 sharing memory.

3727 Much experience with semaphores shows that there are two distinct uses of synchronization:  
3728 locking, which is typically of short duration; and waiting, which is typically of long or  
3729 unbounded duration. These distinct usages map directly onto mutexes and condition  
3730 variables, respectively.

3731 Semaphores are provided in IEEE Std. 1003.1-200x primarily to provide a means of  
3732 synchronization for processes; these processes may or may not share memory. Mutexes and  
3733 condition variables are specified as synchronization mechanisms between threads; these  
3734 threads always share (some) memory. Both are synchronization paradigms that have been in  
3735 widespread use for a number of years. Each set of primitives is particularly well matched to  
3736 certain problems.

3737 With respect to binary semaphores, experience has shown that condition variables and  
 3738 mutexes are easier to use for many synchronization problems than binary semaphores. The  
 3739 primary reason for this is the explicit appearance of a Boolean predicate that specifies when  
 3740 the condition wait is satisfied. This Boolean predicate terminates a loop, including the call to  
 3741 *pthread\_cond\_wait()*. As a result, extra wakeups are benign since the predicate governs  
 3742 whether the thread will actually proceed past the condition wait. With stateful primitives,  
 3743 such as binary semaphores, the wakeup in itself typically means that the wait is satisfied. The  
 3744 burden of ensuring correctness for such waits is thus placed on *all* signalers of the semaphore  
 3745 rather than on an *explicitly coded* Boolean predicate located at the condition wait. Experience  
 3746 has shown that the latter creates a major improvement in safety and ease-of-use.

3747 Counting semaphores are well matched to dealing with producer/consumer problems,  
 3748 including those that might exist between threads of different processes, or between a signal  
 3749 handler and a thread. In the former case, there may be little or no memory shared by the  
 3750 processes; in the latter case, one is not communicating between co-equal threads, but  
 3751 between a thread and an interruptlike entity. It is for these reasons that IEEE Std. 1003.1-200x  
 3752 allows semaphores to be used by threads.

3753 Mutexes and condition variables have been effectively used with and without priority  
 3754 inheritance, priority ceiling, and other attributes to synchronize threads that share memory.  
 3755 The efficiency of their implementation is comparable to or better than that of other  
 3756 synchronization primitives that are sometimes harder to use (for example, binary  
 3757 semaphores). Furthermore, there is at least one known implementation of Ada tasking that  
 3758 uses these primitives. Mutexes and condition variables together constitute an appropriate,  
 3759 sufficient, and complete set of interthread synchronization primitives.

3760 Efficient multi-threaded applications require high-performance synchronization primitives.  
 3761 Considerations of efficiency and generality require a small set of primitives upon which more  
 3762 sophisticated synchronization functions can be built.

#### 3763 • Standardization Issues

3764 It is possible to implement very high-performance semaphores using test-and-set  
 3765 instructions on shared memory locations. The library routines that implement such a high-  
 3766 performance interface has to properly ensure that a *sem\_wait()* or *sem\_trywait()* operation  
 3767 that cannot be performed will issue a blocking semaphore system call or properly report the  
 3768 condition to the application. The same interface to the application program would be  
 3769 provided by a high-performance implementation.

### 3770 B.2.8.1 Realtime Signals

#### 3771 **Realtime Signals Extension**

3772 This portion of the rationale presents models, requirements, and standardization issues relevant  
 3773 to the Realtime Signals Extension. This extension provides the capability required to support  
 3774 reliable, deterministic, asynchronous notification of events. While a new mechanism,  
 3775 unencumbered by the historical usage and semantics of POSIX.1 signals, might allow for a more  
 3776 efficient implementation, the application requirements for event notification can be met with a  
 3777 small number of extensions to signals. Therefore, a minimal set of extensions to signals to  
 3778 support the application requirements is specified.

3779 The realtime signal extensions specified in this section are used by other realtime functions  
 3780 requiring asynchronous notification:

#### 3781 • Models

3782 The model supported is one of multiple cooperating processes, each of which handles  
3783 multiple asynchronous external events. Events represent occurrences that are generated as  
3784 the result of some activity in the system. Examples of occurrences that can constitute an  
3785 event include:

3786 — Completion of an asynchronous I/O request

3787 — Expiration of a POSIX.1b timer

3788 — Arrival of an interprocess message

3789 — Generation of a user-defined event

3790 Processing of these events may occur synchronously via polling for event notifications or  
3791 asynchronously via a software interrupt mechanism. Existing practice for this model is well  
3792 established for traditional proprietary realtime operating systems, realtime executives, and  
3793 realtime extended POSIX-like systems.

3794 A contrasting model is that of “cooperating sequential processes” where each process  
3795 handles a single priority of events via polling. Each process blocks while waiting for events,  
3796 and each process depends on the preemptive, priority-based process scheduling mechanism  
3797 to arbitrate between events of different priority that need to be processed concurrently.  
3798 Existing practice for this model is also well established for small realtime executives that  
3799 typically execute in an unprotected physical address space, but it is just emerging in the  
3800 context of a fuller function operating system with multiple virtual address spaces.

3801 It could be argued that the cooperating sequential process model, and the facilities supported  
3802 by the POSIX Threads Extension obviate a software interrupt model. But, even with the  
3803 cooperating sequential process model, the need has been recognized for a software interrupt  
3804 model to handle exceptional conditions and process aborting, so the mechanism must be  
3805 supported in any case. Furthermore, it is not the purview of IEEE Std. 1003.1-200x to attempt  
3806 to convince realtime practitioners that their current application models based on software  
3807 interrupts are “broken” and should be replaced by the cooperating sequential process model.  
3808 Rather, it is the charter of IEEE Std. 1003.1-200x to provide standard extensions to  
3809 mechanisms that support existing realtime practice.

3810 • Requirements

3811 This section discusses the following realtime application requirements for asynchronous  
3812 event notification:

3813 — Reliable delivery of asynchronous event notification

3814 The events notification mechanism shall guarantee delivery of an event notification.  
3815 Asynchronous operations (such as asynchronous I/O and timers) that complete  
3816 significantly after they are invoked have to guarantee that delivery of the event  
3817 notification can occur at the time of completion.

3818 — Prioritized handling of asynchronous event notifications

3819 The events notification mechanism shall support the assigning of a user function as an  
3820 event notification handler. Furthermore, the mechanism shall support the preemption of  
3821 an event handler function by a higher priority event notification and shall support the  
3822 selection of the highest priority pending event notification when multiple notifications (of  
3823 different priority) are pending simultaneously.

3824 The model here is based on hardware interrupts. Asynchronous event handling allows  
3825 the application to ensure that time-critical events are immediately processed when  
3826 delivered, without the indeterminism of being at a random location within a polling loop.

- 3827            Use of handler priority allows the specification of how handlers are interrupted by other  
3828            higher priority handlers.
- 3829            — Differentiation between multiple occurrences of event notifications of the same type
- 3830            The events notification mechanism shall pass an application-defined value to the event  
3831            handler function. This value can be used for a variety of purposes, such as enabling the  
3832            application to identify which of several possible events of the same type (for example,  
3833            timer expirations) has occurred.
- 3834            — Polled reception of asynchronous event notifications
- 3835            The events notification mechanism shall support blocking and non-blocking polls for  
3836            asynchronous event notification.
- 3837            The polled mode of operation is often preferred over the interrupt mode by those  
3838            practitioners accustomed to this model. Providing support for this model facilitates the  
3839            porting of applications based on this model to POSIX.1b conforming systems.
- 3840            — Deterministic response to asynchronous event notifications
- 3841            The events notification mechanism shall not preclude implementations that provide  
3842            deterministic event dispatch latency and shall minimize the number of system calls  
3843            needed to use the event facilities during realtime processing.
- 3844            • Rationale for Extension
- 3845            POSIX.1 signals have many of the characteristics necessary to support the asynchronous  
3846            handling of event notifications, and the Realtime Signals Extension addresses the following  
3847            deficiencies in the POSIX.1 signal mechanism:
- 3848            — Signals do not support reliable delivery of event notification. Subsequent occurrences of  
3849            a pending signal are not guaranteed to be delivered.
- 3850            — Signals do not support prioritized delivery of event notifications. The order of signal  
3851            delivery when multiple unblocked signals are pending is undefined.
- 3852            — Signals do not support the differentiation between multiple signals of the same type.

### 3853 *B.2.8.2 Asynchronous I/O*

3854            Many applications need to interact with the I/O subsystem in an asynchronous manner. The  
3855            asynchronous I/O mechanism provides the ability to overlap application processing and I/O  
3856            operations initiated by the application. The asynchronous I/O mechanism allows a single  
3857            process to perform I/O simultaneously to a single file multiple times or to multiple files  
3858            multiple times.

#### 3859 **Overview**

3860            Asynchronous I/O operations proceed in logical parallel with the processing done by the  
3861            application after the asynchronous I/O has been initiated. Other than this difference,  
3862            asynchronous I/O behaves similarly to normal I/O using *read()*, *write()*, *lseek()*, and *fsync()*.  
3863            The effect of issuing an asynchronous I/O request is as if a separate thread of execution were to  
3864            perform atomically the implied *lseek()* operation, if any, and then the requested I/O operation  
3865            (either *read()*, *write()*, or *fsync()*). There is no seek implied with a call to *aio\_fsync()*. Concurrent  
3866            asynchronous operations and synchronous operations applied to the same file update the file as  
3867            if the I/O operations had proceeded serially.

3868            When asynchronous I/O completes, a signal can be delivered to the application to indicate the  
3869            completion of the I/O. This signal can be used to indicate that buffers and control blocks used

3870 for asynchronous I/O can be reused. Signal delivery is not required for an asynchronous  
 3871 operation and may be turned off on a per-operation basis by the application. Signals may also be  
 3872 synchronously polled using *aio\_suspend()*, *sigtimedwait()*, or *sigwaitinfo()*.

3873 Normal I/O has a return value and an error status associated with it. Asynchronous I/O returns  
 3874 a value and an error status when the operation is first submitted, but that only relates to whether  
 3875 the operation was successfully queued up for servicing. The I/O operation itself also has a  
 3876 return status and an error value. To allow the application to retrieve the return status and the  
 3877 error value, functions are provided that, given the address of an asynchronous I/O control  
 3878 block, yield the return and error status associated with the operation. Until an asynchronous I/O  
 3879 operation is done, its error status shall be [EINPROGRESS]. Thus, an application can poll for  
 3880 completion of an asynchronous I/O operation by waiting for the error status to become equal to  
 3881 a value other than [EINPROGRESS]. The return status of an asynchronous I/O operation is  
 3882 undefined so long as the error status is equal to [EINPROGRESS].

3883 Storage for asynchronous operation return and error status may be limited. Submission of  
 3884 asynchronous I/O operations may fail if this storage is exceeded. When an application retrieves  
 3885 the return status of a given asynchronous operation, therefore, any system-maintained storage  
 3886 used for this status and the error status may be reclaimed for use by other asynchronous  
 3887 operations.

3888 Asynchronous I/O can be performed on file descriptors that have been enabled for POSIX.1b  
 3889 synchronized I/O. In this case, the I/O operation still occurs asynchronously, as defined herein;  
 3890 however, the asynchronous operation I/O in this case is not completed until the I/O has reached  
 3891 either the state of synchronized I/O data integrity completion or synchronized I/O file integrity  
 3892 completion, depending on the sort of synchronized I/O that is enabled on the file descriptor.

### 3893 **Models**

3894 Three models illustrate the use of asynchronous I/O: a journalization model, a data acquisition  
 3895 model, and a model of the use of asynchronous I/O in supercomputing applications.

- 3896 • **Journalization Model**

3897 Many realtime applications perform low-priority journalizing functions. Journalizing  
 3898 requires that logging records be queued for output without blocking the initiating process.

- 3899 • **Data Acquisition Model**

3900 A data acquisition process may also serve as a model. The process has two or more channels  
 3901 delivering intermittent data that must be read within a certain time. The process issues one  
 3902 asynchronous read on each channel. When one of the channels needs data collection, the  
 3903 process reads the data and posts it through an asynchronous write to secondary memory for  
 3904 future processing.

- 3905 • **Supercomputing Model**

3906 The supercomputing community has used asynchronous I/O much like that specified herein  
 3907 for many years. This community requires the ability to perform multiple I/O operations to  
 3908 multiple devices with a minimal number of entries to “the system”; each entry to “the  
 3909 system” provokes a major delay in operations when compared to the normal progress made  
 3910 by the application. This existing practice motivated the use of combined *lseek()* and *read()* or  
 3911 *write()* calls, as well as the *lio\_listio()* call. Another common practice is to disable signal  
 3912 notification for I/O completion, and simply poll for I/O completion at some interval by  
 3913 which the I/O should be completed. Likewise, interfaces like *aio\_cancel()* have been in  
 3914 successful commercial use for many years. Note also that an underlying implementation of  
 3915 asynchronous I/O will require the ability, at least internally, to cancel outstanding

3916 asynchronous I/O, at least when the process exits. (Consider an asynchronous read from a  
3917 terminal, when the process intends to exit immediately.)

### 3918 **Requirements**

3919 Asynchronous input and output for realtime implementations have these requirements:

- 3920 • The ability to queue multiple asynchronous read and write operations to a single open  
3921 instance. Both sequential and random access should be supported.
- 3922 • The ability to queue asynchronous read and write operations to multiple open instances.
- 3923 • The ability to obtain completion status information by polling and/or asynchronous event  
3924 notification.
- 3925 • Asynchronous event notification on asynchronous I/O completion is optional.
- 3926 • It has to be possible for the application to associate the event with the *aioctx* for the operation  
3927 that generated the event.
- 3928 • The ability to cancel queued requests.
- 3929 • The ability to wait upon asynchronous I/O completion in conjunction with other types of  
3930 events.
- 3931 • The ability to accept an *aio\_read()* and an *aio\_cancel()* for a device that accepts a *read()*, and  
3932 the ability to accept an *aio\_write()* and an *aio\_cancel()* for a device that accepts a *write()*. This  
3933 does not imply that the operation is asynchronous.

### 3934 **Standardization Issues**

3935 The following issues are addressed by the standardization of asynchronous I/O:

- 3936 • Rationale for New Interface

3937 Non-blocking I/O does not satisfy the needs of either realtime or high-performance  
3938 computing models; these models require that a process overlap program execution and I/O  
3939 processing. Realtime applications will often make use of direct I/O to or from the address  
3940 space of the process, or require synchronized (unbuffered) I/O; they also require the ability  
3941 to overlap this I/O with other computation. In addition, asynchronous I/O allows an  
3942 application to keep a device busy at all times, possibly achieving greater throughput.  
3943 Supercomputing and database architectures will often have specialized hardware that can  
3944 provide true asynchrony underlying the logical asynchrony provided by this interface. In  
3945 addition, asynchronous I/O should be supported by all types of files and devices in the same  
3946 manner.

- 3947 • Effect of Buffering

3948 If asynchronous I/O is performed on a file that is buffered prior to being actually written to  
3949 the device, it is possible that asynchronous I/O will offer no performance advantage over  
3950 normal I/O; the cycles *stolen* to perform the asynchronous I/O will be taken away from the  
3951 running process and the I/O will occur at interrupt time. This potential lack of gain in  
3952 performance in no way obviates the need for asynchronous I/O by realtime applications,  
3953 which very often will use specialized hardware support; multiple processors; and/or  
3954 unbuffered, synchronized I/O.

3955 **B.2.8.3 Memory Management**

3956 All memory management and shared memory definitions are located in the `<sys/mman.h>`  
3957 header. This is for alignment with historical practice.

3958 **Memory Locking Functions**

3959 This portion of the rationale presents models, requirements, and standardization issues relevant  
3960 to process memory locking.

## 3961 • Models

3962 Realtime systems that conform to IEEE Std. 1003.1-200x are expected (and desired) to be  
3963 supported on systems with demand-paged virtual memory management, non-paged  
3964 swapping memory management, and physical memory systems with no memory  
3965 management hardware. The general case, however, is the demand-paged, virtual memory  
3966 system with each POSIX process running in a virtual address space. Note that this includes  
3967 architectures where each process resides in its own virtual address space and architectures  
3968 where the address space of each process is only a portion of a larger global virtual address  
3969 space.

3970 The concept of memory locking is introduced to eliminate the indeterminacy introduced by  
3971 paging and swapping, and to support an upper bound on the time required to access the  
3972 memory mapped into the address space of a process. Ideally, this upper bound will be the  
3973 same as the time required for the processor to access “main memory”, including any address  
3974 translation and cache miss overheads. But some implementations—primarily on  
3975 mainframes—will not actually force locked pages to be loaded and held resident in main  
3976 memory. Rather, they will handle locked pages so that accesses to these pages will meet the  
3977 performance metrics for locked process memory in the implementation. Also, although it is  
3978 not, for example, the intention that this interface, as specified, be used to lock process  
3979 memory into “cache”, it is conceivable that an implementation could support a large static  
3980 RAM memory and define this as “main memory” and use a large[r] dynamic RAM as  
3981 “backing store”. These interfaces could then be interpreted as supporting the locking of  
3982 process memory into the static RAM. Support for multiple levels of backing store would  
3983 require extensions to these interfaces.

3984 Implementations may also use memory locking to guarantee a fixed translation between  
3985 virtual and physical addresses where such is beneficial to improving determinacy for  
3986 direct-to/from-process input/output. IEEE Std. 1003.1-200x does not guarantee to the  
3987 application that the virtual-to-physical address translations, if such exist, are fixed, because  
3988 such behavior would not be implementable on all architectures on which implementations of  
3989 IEEE Std. 1003.1-200x are expected. But IEEE Std. 1003.1-200x does mandate that an  
3990 implementation define, for the benefit of potential users, whether or not locking guarantees  
3991 fixed translations.

3992 Memory locking is defined with respect to the address space of a process. Only the pages  
3993 mapped into the address space of a process may be locked by the process, and when the  
3994 pages are no longer mapped into the address space—for whatever reason—the locks  
3995 established with respect to that address space are removed. Shared memory areas warrant  
3996 special mention, as they may be mapped into more than one address space or mapped more  
3997 than once into the address space of a process; locks may be established on pages within these  
3998 areas with respect to several of these mappings. In such a case, the lock state of the  
3999 underlying physical pages is the logical OR of the lock state with respect to each of the  
4000 mappings. Only when all such locks have been removed are the shared pages considered  
4001 unlocked.



4002 In recognition of the page granularity of Memory Management Units (MMU), and in order to  
4003 support locking of ranges of address space, memory locking is defined in terms of “page”  
4004 granularity. That is, for the interfaces that support an address and size specification for the  
4005 region to be locked, the address must be on a page boundary, and all pages mapped by the  
4006 specified range are locked, if valid. This means that the length is implicitly rounded up to a  
4007 multiple of the page size. The page size is implementation-defined and is available to  
4008 applications as a compile time symbolic constant or at runtime via *sysconf()*.

4009 A “real memory” POSIX.1b implementation that has no MMU could elect not to support  
4010 these interfaces, returning [ENOSYS]. But an application could easily interpret this as  
4011 meaning that the implementation would unconditionally page or swap the application when  
4012 such is not the case. It is the intention of IEEE Std. 1003.1-200x that such a system could  
4013 define these interfaces as “NO-OPs”, returning success without actually performing any  
4014 function except for mandated argument checking.

4015 • Requirements

4016 For realtime applications, memory locking is generally considered to be required as part of  
4017 application initialization. This locking is performed after an application has been loaded (that  
4018 is, *exec'd*) and the program remains locked for its entire lifetime. But to support applications  
4019 that undergo major mode changes where, in one mode, locking is required, but in another it  
4020 is not, the specified interfaces allow repeated locking and unlocking of memory within the  
4021 lifetime of a process.

4022 When a realtime application locks its address space, it should not be necessary for the  
4023 application to then “touch” all of the pages in the address space to guarantee that they are  
4024 resident or else suffer potential paging delays the first time the page is referenced. Thus,  
4025 IEEE Std. 1003.1-200x requires that the pages locked by the specified interfaces be resident  
4026 when the locking functions return successfully.

4027 Many architectures support system-managed stacks that grow automatically when the  
4028 current extent of the stack is exceeded. A realtime application has a requirement to be able to  
4029 “preallocate” sufficient stack space and lock it down so that it will not suffer page faults to  
4030 grow the stack during critical realtime operation. There was no consensus on a portable way  
4031 to specify how much stack space is needed, so IEEE Std. 1003.1-200x supports no specific  
4032 interface for preallocating stack space. But an application can portably lock down a specific  
4033 amount of stack space by specifying *MCL\_FUTURE* in a call to *memlockall()* and then calling  
4034 a dummy function that declares an automatic array of the desired size.

4035 Memory locking for realtime applications is also generally considered to be an “all or  
4036 nothing” proposition. That is, the entire process, or none, is locked down. But, for  
4037 applications that have well-defined sections that need to be locked and others that do not,  
4038 IEEE Std. 1003.1-200x supports an optional set of interfaces to lock or unlock a range of  
4039 process addresses. Reasons for locking down a specific range include:

4040 — An asynchronous event handler function that must respond to external events in a  
4041 deterministic manner such that page faults cannot be tolerated

4042 — An input/output “buffer” area that is the target for direct-to-process I/O, and the  
4043 overhead of implicit locking and unlocking for each I/O call cannot be tolerated

4044 Finally, locking is generally viewed as an “application-wide” function. That is, the  
4045 application is globally aware of which regions are locked and which are not over time. This is  
4046 in contrast to a function that is used temporarily within a “third party” library routine whose  
4047 function is unknown to the application, and therefore must have no “side effects”. The  
4048 specified interfaces, therefore, do not support “lock stacking” or “lock nesting” within a  
4049 process. But, for pages that are shared between processes or mapped more than once into a

4050 process address space, “lock stacking” is essentially mandated by the requirement that  
4051 unlocking of pages that are mapped by more than one process or more than once by the same  
4052 process does not affect locks established on the other mappings.

4053 There was some support for “lock stacking” so that locking could be transparently used in  
4054 library functions or opaque modules. But the consensus was not to burden all  
4055 implementations with lock stacking (and reference counting), and an implementation option  
4056 was proposed. There were strong objections to the option because applications would have  
4057 to support both options in order to remain portable. The consensus was to eliminate lock  
4058 stacking altogether, primarily through overwhelming support for the System V  
4059 “m[un]lock[all]” interface on which IEEE Std. 1003.1-200x is now based.

4060 Locks are not inherited across *fork()*s because some systems implement *fork()* by creating  
4061 new address spaces for the child. In such an implementation, requiring locks to be inherited  
4062 would lead to new situations in which a fork would fail due to the inability of the system to  
4063 lock sufficient memory to lock both the parent and the child. The consensus was that there  
4064 was no benefit to such inheritance. Note that this does not mean that locks are removed  
4065 when, for instance, a thread is created in the same address space.

4066 Similarly, locks are not inherited across *exec* because some systems implement *exec* by  
4067 unmapping all of the pages in the address space (which, by definition, removes the locks on  
4068 these pages), and maps in pages of the *exec*'d image. In such an implementation, requiring  
4069 locks to be inherited would lead to new situations in which *exec* would fail. Reporting this  
4070 failure would be very cumbersome to detect in time to report to the calling process, and no  
4071 appropriate mechanism exists for informing the *exec*'d process of its status.

4072 It was determined that, if the newly loaded application required locking, it was the  
4073 responsibility of that application to establish the locks. This is also in keeping with the  
4074 general view that it is the responsibility of the application to be aware of all locks that are  
4075 established.

4076 There was one request to allow (not mandate) locks to be inherited across *fork()*, and a  
4077 request for a flag, MCL\_INHERIT, that would specify inheritance of memory locks across  
4078 *exec*s. Given the difficulties raised by this and the general lack of support for the feature in  
4079 IEEE Std. 1003.1-200x, it was not added. IEEE Std. 1003.1-200x does not preclude an  
4080 implementation from providing this feature for administrative purposes, such as a “run”  
4081 command that will lock down and execute specified program. Additionally, the rationale for  
4082 the objection equated *fork()* with creating a thread in the address space. IEEE Std. 1003.1-200x  
4083 does not mandate releasing locks when creating additional threads in an existing process.

#### 4084 • Standardization Issues

4085 One goal of IEEE Std. 1003.1-200x is to define a set of primitives that provide the necessary  
4086 functionality for realtime applications, with consideration for the needs of other application  
4087 domains where such were identified, which is based to the extent possible on existing  
4088 industry practice.

4089 The Memory Locking option is required by many realtime applications to tune performance.  
4090 Such a facility is accomplished by placing constraints on the virtual memory system to limit  
4091 paging of time of the process or of critical sections of the process. This facility should not be  
4092 used by most non-realtime applications.

4093 Optional features provided in IEEE Std. 1003.1-200x allow applications to lock selected  
4094 address ranges with the caveat that the process is responsible for being aware of the page  
4095 granularity of locking and the un-nested nature of the locks.

4096 **Mapped Files Functions**

4097 The Memory Mapped Files option provides a mechanism that allows a process to access files by  
4098 directly incorporating file data into its address space. Once a file is “mapped” into a process  
4099 address space, the data can be manipulated by instructions as memory. The use of mapped files  
4100 can significantly reduce I/O data movement since file data does not have to be copied into  
4101 process data buffers as in *read()* and *write()*. If more than one process maps a file, its contents  
4102 are shared among them. This provides a low overhead mechanism by which processes can  
4103 synchronize and communicate.

## 4104 • Historical Perspective

4105 Realtime applications have historically been implemented using a collection of cooperating  
4106 processes or tasks. In early systems, these processes ran on bare hardware (that is, without an  
4107 operating system) with no memory relocation or protection. The application paradigms that  
4108 arose from this environment involve the sharing of data between the processes.

4109 When realtime systems were implemented on top of vendor-supplied operating systems, the  
4110 paradigm or performance benefits of direct access to data by multiple processes was still  
4111 deemed necessary. As a result, operating systems that claim to support realtime applications  
4112 must support the shared memory paradigm.

4113 Additionally, a number of realtime systems provide the ability to map specific sections of the  
4114 physical address space into the address space of a process. This ability is required if an  
4115 application is to obtain direct access to memory locations that have specific properties (for  
4116 example, refresh buffers or display devices, dual ported memory locations, DMA target  
4117 locations). The use of this ability is common enough to warrant some degree of  
4118 standardization of its interface. This ability overlaps the general paradigm of shared  
4119 memory in that, in both instances, common global objects are made addressable by  
4120 individual processes or tasks.

4121 Finally, a number of systems also provide the ability to map process addresses to files. This  
4122 provides both a general means of sharing persistent objects, and using files in a manner that  
4123 optimizes memory and swapping space usage.

4124 Simple shared memory is clearly a special case of the more general file mapping capability.  
4125 In addition, there is relatively widespread agreement and implementation of the file  
4126 mapping interface. In these systems, many different types of objects can be mapped (for  
4127 example, files, memory, devices, and so on) using the same mapping interfaces. This  
4128 approach both minimizes interface proliferation and maximizes the generality of programs  
4129 using the mapping interfaces.

## 4130 • Memory Mapped Files Usage

4131 A memory object can be concurrently mapped into the address space of one or more  
4132 processes. The *mmap()* and *munmap()* functions allow a process to manipulate their address  
4133 space by mapping portions of memory objects into it and removing them from it. When  
4134 multiple processes map the same memory object, they can share access to the underlying  
4135 data. Implementations may restrict the size and alignment of mappings to be on *page*-size  
4136 boundaries. The page size, in bytes, is the value of the system-configurable variable  
4137 {PAGESIZE}, typically accessed by calling *sysconf()* with a *name* argument of  
4138 *\_SC\_PAGESIZE*. If an implementation has no restrictions on size or alignment, it may  
4139 specify a 1-byte page size.

4140 To map memory, a process first opens a memory object. The *ftruncate()* function can be used  
4141 to contract or extend the size of the memory object even when the object is currently  
4142 mapped. If the memory object is extended, the contents of the extended areas are zeros.

4143 After opening a memory object, the application maps the object into its address space using  
4144 the *mmap()* function call. Once a mapping has been established, it remains mapped until  
4145 unmapped with *munmap()*, even if the memory object is closed. The *mprotect()* function can  
4146 be used to change the memory protections initially established by *mmap()*.

4147 A *close()* of the file descriptor, while invalidating the file descriptor itself, does not unmap  
4148 any mappings established for the memory object. The address space, including all mapped  
4149 regions, is inherited on *fork()*. The entire address space is unmapped on process termination  
4150 or by successful calls to any of the *exec* family of functions.

4151 The *msync()* function is used to force mapped file data to permanent storage.

4152 • Effects on Other Functions

4153 When the Memory Mapped Files option is supported, the operation of the *open()*, *creat()*, and  
4154 *unlink()* functions are a natural result of using the file system name space to map the global  
4155 names for memory objects.

4156 The *ftruncate()* function can be used to set the length of a sharable memory object.

4157 The meaning of *stat()* fields other than the size and protection information is undefined on  
4158 implementations where memory objects are not implemented using regular files. When  
4159 regular files are used, the times reflect when the implementation updated the file image of  
4160 the data, not when a process updated the data in memory.

4161 The operations of *fdopen()*, *write()*, *read()*, and *lseek()* were made unspecified for objects  
4162 opened with *shm\_open()*, so that implementations that did not implement memory objects as  
4163 regular files would not have to support the operation of these functions on shared memory  
4164 objects.

4165 The behavior of memory objects with respect to *close()*, *dup()*, *dup2()*, *open()*, *close()*, *fork()*,  
4166 *\_exit()*, and the *exec* family of functions is the same as the behavior of the existing practice of  
4167 the *mmap()* function.

4168 A memory object can still be referenced after a close. That is, any mappings made to the file  
4169 are still in effect, and reads and writes that are made to those mappings are still valid and are  
4170 shared with other processes that have the same mapping. Likewise, the memory object can  
4171 still be used if any references remain after its name(s) have been deleted. Any references that  
4172 remain after a close must not appear to the application as file descriptors.

4173 This is existing practice for *mmap()* and *close()*. In addition, there are already mappings  
4174 present (text, data, stack) that do not have open file descriptors. The text mapping in  
4175 particular is considered a reference to the file containing the text. The desire was to treat all  
4176 mappings by the process uniformly. Also, many modern implementations use *mmap()* to  
4177 implement shared libraries, and it would not be desirable to keep file descriptors for each of  
4178 the many libraries an application can use. It was felt there were many other existing  
4179 programs that used this behavior to free a file descriptor, and thus IEEE Std. 1003.1-200x  
4180 could not forbid it and still claim to be using existing practice.

4181 For implementations that implement memory objects using memory only, memory objects  
4182 will retain the memory allocated to the file after the last close and will use that same memory  
4183 on the next open. Note that closing the memory object is not the same as deleting the name,  
4184 since the memory object is still defined in the memory object name space.

4185 The locks of *fcntl()* do not block any read or write operation, including read or write access to  
4186 shared memory or mapped files. In addition, implementations that only support shared  
4187 memory objects should not be required to implement record locks. The reference to *fcntl()* is  
4188 added to make this point explicitly. The other *fcntl()* commands are useful with shared

4189 memory objects.

4190 The size of pages that mapping hardware may be able to support may be a configurable  
4191 value, or it may change based on hardware implementations. The addition of the  
4192 `_SC_PAGESIZE` parameter to the `sysconf()` function is provided for determining the mapping  
4193 page size at runtime.

### 4194 Shared Memory Functions

4195 Implementations may support the Shared Memory Objects option without supporting a general  
4196 Memory Mapped Files option. Shared memory objects are named regions of storage that may be  
4197 independent of the file system and can be mapped into the address space of one or more  
4198 processes to allow them to share the associated memory.

#### 4199 • Requirements

4200 Shared memory is used to share data among several processes, each potentially running at  
4201 different priority levels, responding to different inputs, or performing separate tasks. Shared  
4202 memory is not just simply providing common access to data, it is providing the fastest  
4203 possible communication between the processes. With one memory write operation, a process  
4204 can pass information to as many processes as have the memory region mapped.

4205 As a result, shared memory provides a mechanism that can be used for all other interprocess  
4206 communications facilities. It may also be used by an application for implementing more  
4207 sophisticated mechanisms than semaphores and message queues.

4208 The need for a shared memory interface is obvious for virtual memory systems, where the  
4209 operating system is directly preventing processes from accessing each other's data. However,  
4210 in unprotected systems, such as those found in some embedded controllers, a shared  
4211 memory interface is needed to provide a portable mechanism to allocate a region of memory  
4212 to be shared and then to communicate the address of that region to other processes.

4213 This, then, provides the minimum functionality that a shared memory interface must have in  
4214 order to support realtime applications: to allocate and name an object to be mapped into  
4215 memory for potential sharing (`open()` or `shm_open()`), and to make the memory object  
4216 available within the address space of a process (`mmap()`). To complete the interface, a  
4217 mechanism to release the claim of a process on a shared memory object (`munmap()`) is also  
4218 needed, as well as a mechanism for deleting the name of a sharable object that was  
4219 previously created (`unlink()` or `shm_unlink()`).

4220 After a mapping has been established, an implementation should not have to provide  
4221 services to maintain that mapping. All memory writes into that area will appear immediately  
4222 in the memory mapping of that region by any other processes.

4223 Thus, requirements include:

4224 — Support creation of sharable memory objects and the mapping of these objects into the  
4225 address space of a process.

4226 — Sharable memory objects should be accessed by global names accessible from all  
4227 processes.

4228 — Support the mapping of specific sections of physical address space (such as a memory  
4229 mapped device) into the address space of a process. This should not be done by the  
4230 process specifying the actual address, but again by an implementation-defined global  
4231 name (such as a special device name) dedicated to this purpose.

4232 — Support the mapping of discrete portions of these memory objects.

- 4233 — Support for minimum hardware configurations that contain no physical media on which  
4234 to store shared memory contents permanently.
- 4235 — The ability to preallocate the entire shared memory region so that minimum hardware  
4236 configurations without virtual memory support can guarantee contiguous space.
- 4237 — The maximizing of performance by not requiring functionality that would require  
4238 implementation interaction above creating the shared memory area and returning the  
4239 mapping.
- 4240 Note that the above requirements do not preclude:
- 4241 — The sharable memory object from being implemented using actual files on an actual file  
4242 system.
- 4243 — The global name that is accessible from all processes being restricted to a file system area  
4244 that is dedicated to handling shared memory.
- 4245 — An implementation not providing implementation-defined global names for the purpose  
4246 of physical address mapping.
- 4247 • Shared Memory Objects Usage
- 4248 If the Shared Memory Objects option is supported, a shared memory object may be created,  
4249 or opened if it already exists, with the *shm\_open()* function. If the shared memory object is  
4250 created, it has a length of zero. The *truncate()* function can be used to set the size of the  
4251 shared memory object after creation. The *shm\_unlink()* function removes the name for a  
4252 shared memory object created by *shm\_open()*.
- 4253 • Shared Memory Overview
- 4254 The shared memory facility defined by IEEE Std. 1003.1-200x usually results in memory  
4255 locations being added to the address space of the process. The implementation returns the  
4256 address of the new space to the application by means of a pointer. This works well in  
4257 languages like C. However, in languages such as FORTRAN, it will not work because these  
4258 languages do not have pointer types. In the bindings for such a language, either a special  
4259 COMMON section will need to be defined (which is unlikely), or the binding will have to  
4260 allow existing structures to be mapped. The implementation will likely have to place  
4261 restrictions on the size and alignment of such structures or will have to map a suitable region  
4262 of the address space of the process into the memory object, and thus into other processes.  
4263 These are issues for that particular language binding. For IEEE Std. 1003.1-200x, however, the  
4264 practice will not be forbidden, merely undefined.
- 4265 Two potentially different name spaces are used for naming objects that may be mapped into  
4266 process address spaces. When the Memory Mapped Files option is supported, files may be  
4267 accessed via *open()*. When the Shared Memory Objects option is supported, sharable  
4268 memory objects that might not be files may be accessed via the *shm\_open()* function. These  
4269 options are not mutually-exclusive.
- 4270 Some systems supporting the Shared Memory Objects option may choose to implement the  
4271 shared memory object name space as part of the file system name space. There are several  
4272 reasons for this:
- 4273 — It allows applications to prevent name conflicts by use of the directory structure.
- 4274 — It uses an existing mechanism for accessing global objects and prevents the creation of a  
4275 new mechanism for naming global objects.
- 4276 In such implementations, memory objects can be implemented using regular files, if that is  
4277 what the implementation chooses. The *shm\_open()* function can be implemented as an *open()*

4278 call in a fixed directory followed by a call to *fcntl()* to set *FD\_CLOEXEC*. The *shm\_unlink()*  
 4279 function can be implemented as an *unlink()* call.

4280 On the other hand, it is also expected that small embedded systems that support the Shared  
 4281 Memory Objects option may wish to implement shared memory without having any file  
 4282 systems present. In this case, the implementations may choose to use a simple string valued  
 4283 name space for shared memory regions. The *shm\_open()* function permits either type of  
 4284 implementation.

4285 Some systems have hardware that supports protection of mapped data from certain classes  
 4286 of access and some do not. Systems that supply this functionality can support the Memory  
 4287 Protection option.

4288 Some implementations restrict size, alignment, and protections to be on *page-size*  
 4289 boundaries. If an implementation has no restrictions on size or alignment, it may specify a 1-  
 4290 byte page size. Applications on implementations that do support larger pages must be  
 4291 cognizant of the page size since this is the alignment and protection boundary.

4292 Simple embedded implementations may have a 1-byte page size and only support the Shared  
 4293 Memory Objects option. This provides simple shared memory between processes without  
 4294 requiring mapping hardware.

4295 IEEE Std. 1003.1-200x is silent about how implementations that chose to implement memory  
 4296 objects directly would treat them with standard utilities such as *ls*, because utilities are not  
 4297 within the charter of IEEE Std. 1003.1-200x.

4298 IEEE Std. 1003.1-200x specifically allows a memory object to remain referenced after a close  
 4299 because that is existing practice for the *mmap()* function.

#### 4300 **Typed Memory Functions**

4301 Implementations may support the Typed Memory Objects option without supporting either the  
 4302 Shared Memory option or the Memory Mapped Files option. Typed memory objects are pools of  
 4303 specialized storage, different from the main memory resource normally used by a processor to  
 4304 hold code and data, that can be mapped into the address space of one or more processes.

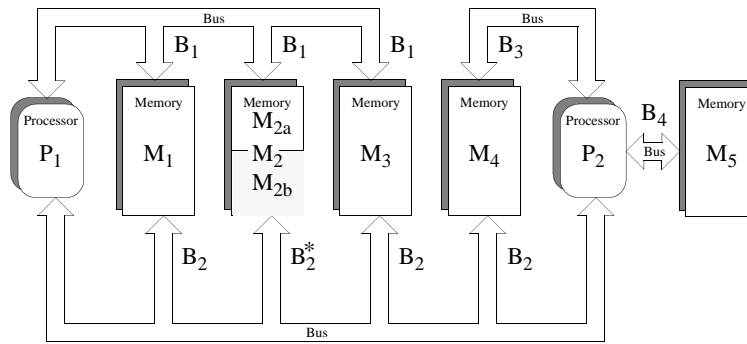
##### 4305 • Model

4306 Realtime systems conforming to one of the POSIX.13 realtime profiles are expected (and  
 4307 desired) to be supported on systems with more than one type or pool of memory (for  
 4308 example, SRAM, DRAM, ROM, EPROM, EEPROM), where each type or pool of memory may  
 4309 be accessible by one or more processors via one or more busses (ports). Memory mapped  
 4310 files, shared memory objects, and the language-specific storage allocation operators (*malloc()*  
 4311 for the ISO C standard, *new* for ANSI Ada) fail to provide application program interfaces  
 4312 versatile enough to allow applications to control their utilization of such diverse memory  
 4313 resources. The typed memory interfaces *posix\_typed\_mem\_open()*, *posix\_mem\_offset()*,  
 4314 *posix\_typed\_mem\_get\_info()*, *mmap()*, and *munmap()* defined herein support the model of  
 4315 typed memory described below.

4316 For purposes of this model, a system comprises several processors (for example, P1 and P2),  
 4317 several physical memory pools (for example, M1, M2, M2a, M2b, M3, M4, and M5), and  
 4318 several busses or “ports” (for example, B1, B2, B3, and B4) interconnecting the various  
 4319 processors and memory pools in some system-specific way. Notice that some memory pools  
 4320 may be contained in others (for example, M2a and M2b are contained in M2).

4321 Figure B-1 (on page 3420) shows an example of such a model. In a system like this, an  
 4322 application should be able to perform the following operations:

4323



\* All addresses in pool  $M_2$  (comprising pools  $M_{2a}$  and  $M_{2b}$ ) accessible via port  $B_1$ .  
 Addresses in pool  $M_{2b}$  are also accessible via port  $B_2$ .  
 Addresses in pool  $M_{2a}$  are NOT accessible via port  $B_2$ .

4324

**Figure B-1** Example of a System with Typed Memory

4325

#### — Typed Memory Allocation

4326

4327

4328

4329

4330

4331

An application should be able to allocate memory dynamically from the desired pool using the desired bus, and map it into a process' address space. For example, processor P1 can allocate some portion of memory pool M1 through port B1, treating all unmapped subareas of M1 as a heap-storage resource from which memory may be allocated. This portion of memory is mapped into the process' address space, and subsequently deallocated when unmapped from all processes.

4332

#### — Using the Same Storage Region from Different Busses

4333

4334

4335

4336

4337

An application process with a mapped region of storage that is accessed from one bus should be able to map that same storage area at another address (subject to page size restrictions detailed in *mmap()*), to allow it to be accessed from another bus. For example, processor P1 may wish to access the same region of memory pool M2b both through ports B1 and B2.

4338

#### — Sharing Typed Memory Regions

4339

4340

4341

4342

4343

4344

4345

4346

4347

4348

4349

4350

Several application processes running on the same or different processors may wish to share a particular region of a typed memory pool. Each process or processor may wish to access this region through different busses. For example, processor P1 may want to share a region of memory pool M4 with processor P2, and they may be required to use busses B2 and B3, respectively, to minimize bus contention. A problem arises here when a process allocates and maps a portion of fragmented memory and then wants to share this region of memory with another process, either in the same processor or different processors. The solution adopted is to allow the first process to find out the memory map (offsets and lengths) of all the different fragments of memory that were mapped into its address space, by repeatedly calling *posix\_mem\_offset()*. Then, this process can pass the offsets and lengths obtained to the second process, which can then map the same memory fragments into its address space.

4351

#### — Contiguous Allocation

4352

4353

4354

4355

The problem of finding the memory map of the different fragments of the memory pool that were mapped into logically contiguous addresses of a given process, can be solved by requesting contiguous allocation. For example, a process in P1 can allocate 10 Kbytes of physically contiguous memory from M3-B1, and obtain the offset (within pool M3) of



4356 this block of memory. Then, it can pass this offset (and the length) to a process in P2 using  
 4357 some interprocess communication mechanism. The second process can map the same  
 4358 block of memory by using the offset transferred and specifying M3-B2.

4359 — Unallocated Mapping

4360 Any subarea of a memory pool that is mapped to a process, either as the result of an  
 4361 allocation request or an explicit mapping, is normally unavailable for allocation. Special  
 4362 processes such as debuggers, however, may need to map large areas of a typed memory  
 4363 pool, yet leave those areas available for allocation.

4364 Typed memory allocation and mapping has to coexist with storage allocation operators like  
 4365 *malloc()*, but systems are free to choose how to implement this coexistence. For example, it  
 4366 may be system configuration-dependent if all available system memory is made part of one  
 4367 of the typed memory pools or if some part will be restricted to conventional allocation  
 4368 operators. Equally system configuration-dependent may be the availability of operators like  
 4369 *malloc()* to allocate storage from certain typed memory pools. It is not excluded to configure  
 4370 a system such that a given named pool, P1, is in turn split into non-overlapping named  
 4371 subpools. For example, M1-B1, M2-B1, and M3-B1 could also be accessed as one common  
 4372 pool M123-B1. A call to *malloc()* on P1 could work on such a larger pool while full  
 4373 optimization of memory usage by P1 would require typed memory allocation at the subpool  
 4374 level.

4375 • Existing Practice

4376 OS-9 provides for the naming (numbering) and prioritization of memory types by a system  
 4377 administrator. It then provides APIs to request memory allocation of typed (colored)  
 4378 memory by number, and to generate a bus address from a mapped memory address  
 4379 (translate). When requesting colored memory, the user can specify type 0 to signify allocation  
 4380 from the first available type in priority order.

4381 HP-RT presents interfaces to map different kinds of storage regions that are visible through a  
 4382 VME bus, although it does not provide allocation operations. It also provides functions to  
 4383 perform address translation between VME addresses and virtual addresses. It represents a  
 4384 VME-bus unique solution to the general problem.

4385 The PSOS approach is similar (that is, based on a pre-established mapping of bus address  
 4386 ranges to specific memories) with a concept of segments and regions (regions dynamically  
 4387 allocated from a heap which is a special segment). Therefore, PSOS does not fully address the  
 4388 general allocation problem either. PSOS does not have a “process”-based model, but more of  
 4389 a “thread”-only-based model of multi-tasking. So mapping to a process address space is not  
 4390 an issue.

4391 QNX (a Canadian OS vendor specializing in realtime embedded systems on 80x86-based  
 4392 processors) uses the System V approach of opening specially named devices (shared memory  
 4393 segments) and using *mmap()* to then gain access from the process. They do not address  
 4394 allocation directly, but once typed shared memory can be mapped, an “allocation manager”  
 4395 process could be written to handle requests for allocation.

4396 The System V approach also included allocation, implemented by opening yet other special  
 4397 “devices” which allocate, rather than appearing as a whole memory object.

4398 The Orkid realtime kernel interface definition has operations to manage memory “regions”  
 4399 and “pools”, which are areas of memory that may reflect the differing physical nature of the  
 4400 memory. Operations to allocate memory from these regions and pools are also provided.

4401 • Requirements

4402 Existing practice in SVID-derived UNIX systems relies on functionality similar to *mmap()*  
4403 and its related interfaces to achieve mapping and allocation of typed memory. However, the  
4404 issue of sharing typed memory (allocated or mapped) and the complication of multiple ports  
4405 are not addressed in any consistent way by existing UNIX system practice. Part of this  
4406 functionality is existing practice in specialized realtime operating systems. In order to  
4407 solidify the capabilities implied by the model above, the following requirements are imposed  
4408 on the interface:

4409 — Identification of Typed Memory Pools and Ports

4410 All processes (running in all processors) in the system shall be able to identify a particular  
4411 (system configured) typed memory pool accessed through a particular (system  
4412 configured) port by a name. That name shall be a member of a name space common to all  
4413 these processes, but need not be the same name space as that containing ordinary file  
4414 names. The association between memory pools/ports and corresponding names is  
4415 typically established when the system is configured. The “open” operation for typed  
4416 memory objects should be distinct from the *open()* function, for consistency with other  
4417 similar services, but implementable on top of *open()*. This implies that the handle for a  
4418 typed memory object will be a file descriptor.

4419 — Allocation and Mapping of Typed Memory

4420 Once a typed memory object has been identified by a process, it shall be possible to both  
4421 map user-selected subareas of that object into process address space and to map system-  
4422 selected (that is, dynamically allocated) subareas of that object, with user-specified  
4423 length, into process address space. It shall also be possible to determine the maximum  
4424 length of memory allocation that may be requested from a given typed memory object.

4425 — Sharing Typed Memory

4426 Two or more processes shall be able to share portions of typed memory, either user-  
4427 selected or dynamically allocated. This requirement applies also to dynamically allocated  
4428 regions of memory that are composed of several non-contiguous pieces.

4429 — Contiguous Allocation

4430 For dynamic allocation, it shall be the user’s option whether the system is required to  
4431 allocate a contiguous subarea within the typed memory object, or whether it is permitted  
4432 to allocate discontinuous fragments which appear contiguous in the process mapping.  
4433 Contiguous allocation simplifies the process of sharing allocated typed memory, while  
4434 discontinuous allocation allows for potentially better recovery of deallocated typed  
4435 memory.

4436 — Accessing Typed Memory Through Different Ports

4437 Once a subarea of a typed memory object has been mapped, it shall be possible to  
4438 determine the location and length corresponding to a user-selected portion of that object  
4439 within the memory pool. This location and length can then be used to remap that portion  
4440 of memory for access from another port. If the referenced portion of typed memory was  
4441 allocated discontinuously, the length thus determined may be shorter than anticipated,  
4442 and the user code shall adapt to the value returned.

4443 — Deallocation

4444 When a previously mapped subarea of typed memory is no longer mapped by any  
4445 process in the system—as a result of a call or calls to *munmap()*—that subarea shall  
4446 become potentially reusable for dynamic allocation; actual reuse of the subarea is a  
4447 function of the dynamic typed memory allocation policy.

- 4448 — Unallocated Mapping
- 4449 It shall be possible to map user-selected subareas of a typed memory object without  
4450 marking that subarea as unavailable for allocation. This option is not the default behavior,  
4451 and shall require appropriate privilege.
- 4452 • Scenario
- 4453 The following scenario will serve to clarify the use of the typed memory interfaces.
- 4454 Process A running on P1 (see Figure B-1 (on page 3420)) wants to allocate some memory  
4455 from memory pool M2, and it wants to share this portion of memory with process B running  
4456 on P2. Since P2 only has access to the lower part of M2, both processes will use the memory  
4457 pool named M2b which is the part of M2 that is accessible both from P1 and P2. The  
4458 operations that both processes need to perform are shown below:
- 4459 — Allocating Typed Memory
- 4460 Process A calls *posix\_typed\_mem\_open()* with the name **/typed.m2b-b1** and a *tflag* of  
4461 POSIX\_TYPED\_MEM\_ALLOCATE to get a file descriptor usable for allocating from pool  
4462 M2b accessed through port B1. It then calls *mmap()* with this file descriptor requesting a  
4463 length of 4 096 bytes. The system allocates two discontinuous blocks of sizes 1 024 and  
4464 3 072 bytes within M2b. The *mmap()* function returns a pointer to a 4 096 byte array in  
4465 process A's logical address space, mapping the allocated blocks contiguously. Process A  
4466 can then utilize the array, and store data in it.
- 4467 — Determining the Location of the Allocated Blocks
- 4468 Process A can determine the lengths and offsets (relative to M2b) of the two blocks  
4469 allocated, by using the following procedure: First, process A calls *posix\_mem\_offset()*  
4470 with the address of the first element of the array and length 4 096. Upon return, the offset and  
4471 length (1 024 bytes) of the first block are returned. A second call to *posix\_mem\_offset()* is  
4472 then made using the address of the first element of the array plus 1 024 (the length of the  
4473 first block), and a new length of 4 096–1 024. If there were more fragments allocated, this  
4474 procedure could have been continued within a loop until the offsets and lengths of all the  
4475 blocks were obtained. Notice that this relatively complex procedure can be avoided if  
4476 contiguous allocation is requested (by opening the typed memory object with the *tflag*  
4477 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG).
- 4478 — Sharing Data Across Processes
- 4479 Process A passes the two offset values and lengths obtained from the *posix\_mem\_offset()*  
4480 calls to process B running on P2, via some form of interprocess communication. Process B  
4481 can gain access to process A's data by calling *posix\_typed\_mem\_open()* with the name  
4482 **/typed.m2b-b2** and a *tflag* of zero, then using two *mmap()* calls on the resulting file  
4483 descriptor to map the two subareas of that typed memory object to its own address space.
- 4484 • Rationale for no *mem\_alloc()* and *mem\_free()*
- 4485 The standard developers had originally proposed a pair of new flags to *mmap()* which, when  
4486 applied to a typed memory object descriptor, would cause *mmap()* to allocate dynamically  
4487 from an unallocated and unmapped area of the typed memory object. Deallocation was  
4488 similarly accomplished through the use of *munmap()*. This was rejected by the ballot group  
4489 because it excessively complicated the (already rather complex) *mmap()* interface and  
4490 introduced semantics useful only for typed memory, to a function which must also map  
4491 shared memory and files. They felt that a memory allocator should be built on top of *mmap()*  
4492 instead of being incorporated within the same interface, much as the ISO C standard libraries  
4493 build *malloc()* on top of the virtual memory mapping functions *brk()* and *sbrk()*. This would

4494 eliminate the complicated semantics involved with unmapping only part of an allocated  
4495 block of typed memory.

4496 To attempt to achieve ballot group consensus, typed memory allocation and deallocation was  
4497 first migrated from *mmap()* and *munmap()* to a pair of complementary functions modeled on  
4498 the ISO C standard *malloc()* and *free()*. The *mem\_alloc()* function specified explicitly the  
4499 typed memory object (typed memory pool/access port) from which allocation takes place,  
4500 unlike *malloc()* where the memory pool and port are unspecified. The *mem\_free()* function  
4501 handled deallocation. These new semantics still met all of the requirements detailed above  
4502 without modifying the behavior of *mmap()* except to allow it to map specified areas of typed  
4503 memory objects. An implementation would have been free to implement *mem\_alloc()* and  
4504 *mem\_free()* over *mmap()*, through *mmap()*, or independently but cooperating with *mmap()*.

4505 The ballot group was queried to see if this was an acceptable alternative, and while there was  
4506 some agreement that it achieved the goal of removing the complicated semantics of  
4507 allocation from the *mmap()* interface, several balloters realized that it just created two  
4508 additional functions that behaved, in great part, like *mmap()*. These balloters proposed an  
4509 alternative which has been implemented here in place of a separate *mem\_alloc()* and  
4510 *mem\_free()*. This alternative is based on four specific suggestions:

- 4511 1. The *posix\_typed\_mem\_open()* function should provide a flag which specifies “allocate  
4512 on *mmap()*” (otherwise, *mmap()* just maps the underlying object). This allows things  
4513 roughly similar to */dev/zero* versus */dev/swap*. Two such flags have been implemented,  
4514 one of which forces contiguous allocation.
- 4515 2. The *posix\_mem\_offset()* function is acceptable because it can be applied usefully to  
4516 mapped objects in general. It should return the file descriptor of the underlying object.
- 4517 3. The *mem\_get\_info()* function in an earlier draft should be renamed  
4518 *posix\_typed\_mem\_get\_info()* because it is not generally applicable to memory objects. It  
4519 should probably return the file descriptor’s allocation attribute. We have implemented  
4520 the renaming of the function, but reject having it return a piece of information which is  
4521 readily known by an application without this function. Its whole purpose is to query  
4522 the typed memory object for attributes that are not user-specified, but determined by  
4523 the implementation.
- 4524 4. There should be no separate *mem\_alloc()* or *mem\_free()* functions. Instead, using  
4525 *mmap()* on a typed memory object opened with an “allocate on *mmap()*” flag should be  
4526 used to force allocation. These are precisely the semantics defined in the current draft.

4527 • Rationale for no Typed Memory Access Management

4528 The working group had originally defined an additional interface (and an additional kind of  
4529 object: typed memory master) to establish and dissolve mappings to typed memory on  
4530 behalf of devices or processors which were independent of the operating system and had no  
4531 inherent capability to directly establish mappings on their own. This was to have provided  
4532 functionality similar to device driver interfaces such as *physio()* and their underlying bus-  
4533 specific interfaces (for example, *mballoc()*) which serve to set up and break down DMA  
4534 pathways, and derive mapped addresses for use by hardware devices and processor cards.

4535 The ballot group felt that this was beyond the scope of POSIX.1 and its amendments.  
4536 Furthermore, the removal of interrupt handling interfaces from a preceding amendment (the  
4537 IEEE Std. 1003.1d-1999) during its balloting process renders these typed memory access  
4538 management interfaces an incomplete solution to portable device management from a user  
4539 process; it would be possible to initiate a device transfer to/from typed memory, but  
4540 impossible to handle the transfer-complete interrupt in a portable way.

4541 To achieve ballot group consensus, all references to typed memory access management  
4542 capabilities were removed. The concept of portable interfaces from a device driver to both  
4543 operating system and hardware is being addressed by the Uniform Driver Interface (UDI)  
4544 industry forum, with formal standardization deferred until proof of concept and industry-  
4545 wide acceptance and implementation.

#### 4546 *B.2.8.4 Process Scheduling*

4547 This portion of the rationale presents models, requirements, and standardization issues relevant  
4548 to process scheduling; see also Section B.2.9.4 (on page 3464).

4549 In an operating system supporting multiple concurrent processes, the system determines the  
4550 order in which processes execute to meet system-defined goals. For time-sharing systems, the  
4551 goal is to enhance system throughput and promote fairness; the application is provided little or  
4552 no control over this sequencing function. While this is acceptable and desirable behavior in a  
4553 time-sharing system, it is inappropriate in a realtime system; realtime applications must  
4554 specifically control the execution sequence of their concurrent processes in order to meet  
4555 externally defined response requirements.

4556 In IEEE Std. 1003.1-200x, the control over process sequencing is provided using a concept of  
4557 scheduling policies. These policies, described in detail in this section, define the behavior of the  
4558 system whenever processor resources are to be allocated to competing processes. Only the  
4559 behavior of the policy is defined; conforming implementations are free to use any mechanism  
4560 desired to achieve the described behavior.

#### 4561 • Models

4562 In an operating system supporting multiple concurrent processes, the system determines the  
4563 order in which processes execute and might force long-running processes to yield to other  
4564 processes at certain intervals. Typically, the scheduling code is executed whenever an event  
4565 occurs that might alter the process to be executed next.

4566 The simplest scheduling strategy is a “first-in, first-out” (FIFO) dispatcher. Whenever a  
4567 process becomes runnable, it is placed on the end of a ready list. The process at the front of  
4568 the ready list is executed until it exits or becomes blocked, at which point it is removed from  
4569 the list. This scheduling technique is also known as “run-to-completion” or “run-to-block”.

4570 A natural extension to this scheduling technique is the assignment of a “non-migrating  
4571 priority” to each process. This policy differs from strict FIFO scheduling in only one respect:  
4572 whenever a process becomes runnable, it is placed at the end of the list of processes runnable  
4573 at that priority level. When selecting a process to run, the system always selects the first  
4574 process from the highest priority queue with a runnable process. Thus, when a process  
4575 becomes unblocked, it will preempt a running process of lower priority without otherwise  
4576 altering the ready list. Further, if a process elects to alter its priority, it is removed from the  
4577 ready list and reinserted, using its new priority, according to the policy above.

4578 While the above policy might be considered unfriendly in a time-sharing environment in  
4579 which multiple users require more balanced resource allocation, it could be ideal in a  
4580 realtime environment for several reasons. The most important of these is that it is  
4581 deterministic: the highest-priority process is always run and, among processes of equal  
4582 priority, the process that has been runnable for the longest time is executed first. Because of  
4583 this determinism, cooperating processes can implement more complex scheduling simply by  
4584 altering their priority. For instance, if processes at a single priority were to reschedule  
4585 themselves at fixed time intervals, a time-slice policy would result.

4586 In a dedicated operating system in which all processes are well-behaved realtime  
4587 applications, non-migrating priority scheduling is sufficient. However, many existing

4588 implementations provide for more complex scheduling policies.

4589 IEEE Std. 1003.1-200x specifies a linear scheduling model. In this model, every process in the  
4590 system has a priority. The system scheduler always dispatches a process that has the highest  
4591 (generally the most time-critical) priority among all runnable processes in the system. As  
4592 long as there is only one such process, the dispatching policy is trivial. When multiple  
4593 processes of equal priority are eligible to run, they are ordered according to a strict run-to-  
4594 completion (FIFO) policy.

4595 The priority is represented as a positive integer and is inherited from the parent process. For  
4596 processes running under a fixed priority scheduling policy, the priority is never altered  
4597 except by an explicit function call.

4598 It was determined arbitrarily that larger integers correspond to “higher priorities”.

4599 Certain implementations might impose restrictions on the priority ranges to which processes  
4600 can be assigned. There also can be restrictions on the set of policies to which processes can be  
4601 set.

4602 • Requirements

4603 Realtime processes require that scheduling be fast and deterministic, and that it guarantees  
4604 to preempt lower priority processes.

4605 Thus, given the linear scheduling model, realtime processes require that they be run at a  
4606 priority that is higher than other processes. Within this framework, realtime processes are  
4607 free to yield execution resources to each other in a completely portable and implementation-  
4608 defined manner.

4609 As there is a generally perceived requirement for processes at the same priority level to share  
4610 processor resources more equitably, provisions are made by providing a scheduling policy  
4611 (that is, SCHED\_RR) intended to provide a timeslice-like facility.

4612 **Note:** The following topics assume that low numeric priority implies low scheduling  
4613 criticality and *vice versa*.

4614 • Rationale for New Interface

4615 Realtime applications need to be able to determine when processes will run in relation to  
4616 each other. It must be possible to guarantee that a critical process will run whenever it is  
4617 runnable; that is, whenever it wants to for as long as it needs. SCHED\_FIFO satisfies this  
4618 requirement. Additionally, SCHED\_RR was defined to meet a realtime requirement for a  
4619 well-defined time-sharing policy for processes at the same priority.

4620 It would be possible to use the BSD *setpriority()* and *getpriority()* functions by redefining the  
4621 meaning of the “nice” parameter according to the scheduling policy currently in use by the  
4622 process. The System V *nice()* interface was felt to be undesirable for realtime because it  
4623 specifies an adjustment to the “nice” value, rather than setting it to an explicit value.  
4624 Realtime applications will usually want to set priority to an explicit value. Also, System V  
4625 *nice()* does not allow for changing the priority of another process.

4626 With the POSIX.1b interfaces, the traditional “nice” value does not affect the SCHED\_FIFO  
4627 or SCHED\_RR scheduling policies. If a “nice” value is supported, it is implementation-  
4628 defined whether it affects the SCHED\_OTHER policy.

4629 An important aspect of IEEE Std. 1003.1-200x is the explicit description of the queuing and  
4630 preemption rules. It is critical, to achieve deterministic scheduling, that such rules be stated  
4631 clearly in IEEE Std. 1003.1-200x.

4632 IEEE Std. 1003.1-200x does not address the interaction between priority and swapping. The  
4633 issues involved with swapping and virtual memory paging are extremely implementation-  
4634 defined and would be nearly impossible to standardize at this point. The proposed  
4635 scheduling paradigm, however, fully describes the scheduling behavior of runnable  
4636 processes, of which one criterion is that the working set be resident in memory. Assuming  
4637 the existence of a portable interface for locking portions of a process in memory, paging  
4638 behavior need not affect the scheduling of realtime processes.

4639 IEEE Std. 1003.1-200x also does not address the priorities of “system” processes. In general,  
4640 these processes should always execute in low-priority ranges to avoid conflict with other  
4641 realtime processes. Implementations should document the priority ranges in which system  
4642 processes run.

4643 The default scheduling policy is not defined. The effect of I/O interrupts and other system  
4644 processing activities is not defined. The temporary lending of priority from one process to  
4645 another (such as for the purposes of affecting freeing resources) by the system is not  
4646 addressed. Preemption of resources is not addressed. Restrictions on the ability of a process  
4647 to affect other processes beyond a certain level (influence levels) is not addressed.

4648 The rationale used to justify the simple time-quantum scheduler is that it is common practice  
4649 to depend upon this type of scheduling to assure “fair” distribution of processor resources  
4650 among portions of the application that must interoperate in a serial fashion. Note that  
4651 IEEE Std. 1003.1-200x is silent with respect to the setting of this time quantum, or whether it  
4652 is a system-wide value or a per-process value, although it appears that the prevailing  
4653 realtime practice is for it to be a system-wide value.

4654 In a system with  $N$  processes at a given priority, all processor-bound, in which the time  
4655 quantum is equal for all processes at a specific priority level, the following assumptions are  
4656 made of such a scheduling policy:

- 4657 1. A time quantum  $Q$  exists and the current process will own control of the processor for  
4658 at least a duration of  $Q$  and will have the processor for a duration of  $Q$ .
- 4659 2. The  $N$ th process at that priority will control a processor within a duration of  $(N-1) \times Q$ .

4660 These assumptions are necessary to provide equal access to the processor and bounded  
4661 response from the application.

4662 The assumptions hold for the described scheduling policy only if no system overhead, such  
4663 as interrupt servicing, is present. If the interrupt servicing load is non-zero, then one of the  
4664 two assumptions becomes fallacious, based upon how  $Q$  is measured by the system.

4665 If  $Q$  is measured by clock time, then the assumption that the process obtains a duration  $Q$   
4666 processor time is false if interrupt overhead exists. Indeed, a scenario can be constructed with  
4667  $N$  processes in which a single process undergoes complete processor starvation if a  
4668 peripheral device, such as an analog-to-digital converter, generates significant interrupt  
4669 activity periodically with a period of  $N \times Q$ .

4670 If  $Q$  is measured as actual processor time, then the assumption that the  $N$ th process runs in  
4671 within the duration  $(N-1) \times Q$  is false.

4672 It should be noted that SCHED\_FIFO suffers from interrupt-based delay as well. However,  
4673 for SCHED\_FIFO, the implied response of the system is “as soon as possible”, so that the  
4674 interrupt load for this case is a vendor selection and not a compliance issue.

4675 With this in mind, it is necessary either to complete the definition by including bounds on the  
4676 interrupt load, or to modify the assumptions that can be made about the scheduling policy.

4677 Since the motivation of inclusion of the policy is common usage, and since current  
4678 applications do not enjoy the luxury of bounded interrupt load, item (2) above is sufficient to  
4679 express existing application needs and is less restrictive in the standard definition. No  
4680 difference in interface is necessary.

4681 In an implementation in which the time quantum is equal for all processes at a specific  
4682 priority, our assumptions can then be restated as:

4683 — A time quantum  $Q$  exists, and a processor-bound process will be rescheduled after a  
4684 duration of, at most,  $Q$ . Time quantum  $Q$  may be defined in either wall clock time or  
4685 execution time.

4686 — In general, the  $N$ th process of a priority level should wait no longer than  $(N-1) \times Q$  time  
4687 to execute, assuming no processes exist at higher priority levels.

4688 — No process should wait indefinitely.

4689 For implementations supporting per-process time quanta, these assumptions can be readily  
4690 extended.

### 4691 **Sporadic Server Scheduling Policy**

4692 The sporadic server is a mechanism defined for scheduling aperiodic activities in time-critical  
4693 realtime systems. This mechanism reserves a certain bounded amount of execution capacity for  
4694 processing aperiodic events at a high priority level. Any aperiodic events that cannot be  
4695 processed within the bounded amount of execution capacity are executed in the background at a  
4696 low priority level. Thus, a certain amount of execution capacity can be guaranteed to be  
4697 available for processing periodic tasks, even under burst conditions in the arrival of aperiodic  
4698 processing requests (that is, a large number of requests in a short time interval). The sporadic  
4699 server also simplifies the schedulability analysis of the realtime system, because it allows  
4700 aperiodic processes or threads to be treated as if they were periodic. The sporadic server was  
4701 first described by Sprunt, et al.

4702 The key concept of the sporadic server is to provide and limit a certain amount of computation  
4703 capacity for processing aperiodic events at their assigned normal priority, during a time interval  
4704 called the *replenishment period*. Once the entity controlled by the sporadic server mechanism is  
4705 initialized with its period and execution-time budget attributes, it preserves its execution  
4706 capacity until an aperiodic request arrives. The request will be serviced (if there are no higher  
4707 priority activities pending) as long as there is execution capacity left. If the request is completed,  
4708 the actual execution time used to service it is subtracted from the capacity, and a replenishment  
4709 of this amount of execution time is scheduled to happen one replenishment period after the  
4710 arrival of the aperiodic request. If the request is not completed, because there is no execution  
4711 capacity left, then the aperiodic process or thread is assigned a lower background priority. For  
4712 each portion of consumed execution capacity the execution time used is replenished after one  
4713 replenishment period. At the time of replenishment, if the sporadic server was executing at a  
4714 background priority level, its priority is elevated to the normal level. Other similar  
4715 replenishment policies have been defined, but the one presented here represents a compromise  
4716 between efficiency and implementation complexity.

4717 The interface that appears in this section defines a new scheduling policy for threads and  
4718 processes that behaves according to the rules of the sporadic server mechanism. Scheduling  
4719 attributes are defined and functions are provided to allow the user to set and get the parameters  
4720 that control the scheduling behavior of this mechanism, namely the normal and low priority, the  
4721 replenishment period, the maximum number of pending replenishment operations, and the  
4722 initial execution-time budget.



- 4723 • Scheduling Aperiodic Activities
- 4724 Virtually all realtime applications are required to process aperiodic activities. In many cases,  
4725 there are tight timing constraints that the response to the aperiodic events must meet. Usual  
4726 timing requirements imposed on the response to these events are:
  - 4727 — The effects of an aperiodic activity on the response time of lower priority activities must  
4728 be controllable and predictable.
  - 4729 — The system must provide the fastest possible response time to aperiodic events.
  - 4730 — It must be possible to take advantage of all the available processing bandwidth not  
4731 needed by time-critical activities to enhance average-case response times to aperiodic  
4732 events.
- 4733 Traditional methods for scheduling aperiodic activities are background processing, polling  
4734 tasks, and direct event execution:
  - 4735 — Background processing consists of assigning a very low priority to the processing of  
4736 aperiodic events. It utilizes all the available bandwidth in the system that has not been  
4737 consumed by higher priority threads. However, it is very difficult, or impossible, to meet  
4738 requirements on average-case response time, because the aperiodic entity has to wait for  
4739 the execution of all other entities which have higher priority.
  - 4740 — Polling consists of creating a periodic process or thread for servicing aperiodic requests.  
4741 At regular intervals, the polling entity is started and it services accumulated pending  
4742 aperiodic requests. If no aperiodic requests are pending, the polling entity suspends itself  
4743 until its next period. Polling allows the aperiodic requests to be processed at a higher  
4744 priority level. However, worst and average-case response times of polling entities are a  
4745 direct function of the polling period, and there is execution overhead for each polling  
4746 period, even if no event has arrived. If the deadline of the aperiodic activity is short  
4747 compared to the inter-arrival time, the polling frequency must be increased to guarantee  
4748 meeting the deadline. For this case, the increase in frequency can dramatically reduce the  
4749 efficiency of the system and, therefore, its capacity to meet all deadlines. Yet, polling  
4750 represents a good way to handle a large class of practical problems because it preserves  
4751 system predictability, and because the amortized overhead drops as load increases.
  - 4752 — Direct event execution consists of executing the aperiodic events at a high fixed-priority  
4753 level. Typically, the aperiodic event is processed by an interrupt service routine as soon as  
4754 it arrives. This technique provides predictable response times for aperiodic events, but  
4755 makes the response times of all lower priority activities completely unpredictable under  
4756 burst arrival conditions. Therefore, if the density of aperiodic event arrivals is  
4757 unbounded, it may be a dangerous technique for time-critical systems. Yet, for those cases  
4758 in which the physics of the system imposes a bound on the event arrival rate, it is  
4759 probably the most efficient technique.
  - 4760 — The sporadic server scheduling algorithm combines the predictability of the polling  
4761 approach with the short response times of the direct event execution. Thus, it allows  
4762 systems to meet an important class of application requirements that cannot be met by  
4763 using the traditional approaches. Multiple sporadic servers with different attributes can  
4764 be applied to the scheduling of multiple classes of aperiodic events, each with different  
4765 kinds of timing requirements, such as individual deadlines, average response times, and  
4766 so on. It also has many other interesting applications for realtime, such as scheduling  
4767 producer/consumer tasks in time-critical systems, limiting the effects of faults on the  
4768 estimation of task execution-time requirements, and so on.

- 4769       • Existing Practice  
4770       The sporadic server has been used in different kinds of applications, including military  
4771       avionics, robot control systems, industrial automation systems, and so on. There are  
4772       examples of many systems that cannot be successfully scheduled using the classic  
4773       approaches, such as direct event execution, or polling, and are schedulable using a sporadic  
4774       server scheduler. The sporadic server algorithm itself can successfully schedule all systems  
4775       scheduled with direct event execution or polling.  
  
4776       The sporadic server scheduling policy has been implemented as a commercial product in the  
4777       run-time system of the Verdex Ada compiler. There are also many applications that have  
4778       used a much less efficient application-level sporadic server. These real-time applications  
4779       would benefit from a sporadic server scheduler implemented at the scheduler level.
- 4780       • Library-Level *versus* Kernel-Level Implementation  
4781       The sporadic server interface described in this section requires the sporadic server policy to  
4782       be implemented at the same level as the scheduler. This means that the process sporadic  
4783       server shall be implemented at the kernel level and the thread sporadic server policy shall be  
4784       implemented at the same level as the thread scheduler; that is, kernel or library level.  
  
4785       In an earlier interface for the sporadic server, this mechanism was implementable at a  
4786       different level than the scheduler. This feature allowed the implementer to choose between  
4787       an efficient scheduler-level implementation, or a simpler user or library-level  
4788       implementation. However, the working group considered that this interface made the use of  
4789       sporadic servers more complex, and that library-level implementations would lack some of  
4790       the important functionality of the sporadic server, namely the limitation of the actual  
4791       execution time of aperiodic activities. The working group also felt that the interface  
4792       described in this chapter does not preclude library-level implementations of threads intended  
4793       to provide efficient low-overhead scheduling for those threads that are not scheduled under  
4794       the sporadic server policy.
- 4795       • Range of Scheduling Priorities  
4796       Each of the scheduling policies supported in IEEE Std. 1003.1-200x has an associated range of  
4797       priorities. The priority ranges for each policy might or might not overlap with the priority  
4798       ranges of other policies. For time-critical realtime applications it is usual for periodic and  
4799       aperiodic activities to be scheduled together in the same processor. Periodic activities will  
4800       usually be scheduled using the SCHED\_FIFO scheduling policy, while aperiodic activities  
4801       may be scheduled using SCHED\_SPORADIC. Since the application developer will require  
4802       complete control over the relative priorities of these activities in order to meet his timing  
4803       requirements, it would be desirable for the priority ranges of SCHED\_FIFO and  
4804       SCHED\_SPORADIC to overlap completely. Therefore, although IEEE Std. 1003.1-200x does  
4805       not require any particular relationship between the different priority ranges, it is  
4806       recommended that these two ranges should coincide.
- 4807       • Dynamically Setting the Sporadic Server Policy  
4808       Several members of the working group requested that implementations should not be  
4809       required to support dynamically setting the sporadic server scheduling policy for a thread.  
4810       The reason is that this policy may have a high overhead for library-level implementations of  
4811       threads, and if threads are allowed to dynamically set this policy, this overhead can be  
4812       experienced even if the thread does not use that policy. By disallowing the dynamic setting  
4813       of the sporadic server scheduling policy, these implementations can accomplish efficient  
4814       scheduling for threads using other policies. If a strictly conforming application needs to use  
4815       the sporadic server policy, and is therefore willing to pay the overhead, it must set this policy  
4816       at the time of thread creation.

4817 • Limitation of the Number of Pending Replenishments

4818 The number of simultaneously pending replenishment operations must be limited for each  
 4819 sporadic server for two reasons: an unlimited number of replenishment operations would  
 4820 need an unlimited number of system resources to store all the pending replenishment  
 4821 operations; on the other hand, in some implementations each replenishment operation will  
 4822 represent a source of priority inversion (just for the duration of the replenishment operation)  
 4823 and thus, the maximum amount of replenishments must be bounded to guarantee bounded  
 4824 response times. The way in which the number of replenishments is bounded is by lowering  
 4825 the priority of the sporadic server to *sched\_ss\_low\_priority* when the number of pending  
 4826 replenishments has reached its limit. In this way, no new replenishments are scheduled until  
 4827 the number of pending replenishments decreases.

4828 In the sporadic server scheduling policy defined in IEEE Std. 1003.1-200x, the application can  
 4829 specify the maximum number of pending replenishment operations for a single sporadic  
 4830 server, by setting the value of the *sched\_ss\_max\_repl* scheduling parameter. This value must  
 4831 be between one and {SS\_REPL\_MAX}, which is a maximum limit imposed by the  
 4832 implementation. The limit {SS\_REPL\_MAX} must be greater than or equal to  
 4833 {\_POSIX\_SS\_REPL\_MAX}, which is defined to be four in IEEE Std. 1003.1-200x. The  
 4834 minimum limit of four was chosen so that an application can at least guarantee that four  
 4835 different aperiodic events can be processed during each interval of length equal to the  
 4836 replenishment period.

4837 **B.2.8.5** *Clocks and Timers*

4838 • Clocks

4839 IEEE Std. 1003.1-200x and the ISO C standard both define functions for obtaining system  
 4840 time. Implicit behind these functions is a mechanism for measuring passage of time. This  
 4841 specification makes this mechanism explicit and calls it a clock. The *CLOCK\_REALTIME*  
 4842 clock required by IEEE Std. 1003.1-200x is a higher resolution version of the clock that  
 4843 maintains POSIX.1 system time. This is a “system-wide” clock, in that it is visible to all  
 4844 processes and, were it possible for multiple processes to all read the clock at the same time,  
 4845 they would see the same value.

4846 An extensible interface was defined, with the ability for implementations to define additional  
 4847 clocks. This was done because of the observation that many realtime platforms support  
 4848 multiple clocks, and it was desired to fit this model within the standard interface. But  
 4849 implementation-defined clocks need not represent actual hardware devices, nor are they  
 4850 necessarily system-wide.

4851 • Timers

4852 Two timer types are required for a system to support realtime applications:

4853 1. One-shot

4854 A one-shot timer is a timer that is armed with an initial expiration time, either relative  
 4855 to the current time or at an absolute time (based on some timing base, such as time in  
 4856 seconds and nanoseconds since the Epoch). The timer expires once and then is  
 4857 disarmed. With the specified facilities, this is accomplished by setting the *it\_value*  
 4858 member of the *value* argument to the desired expiration time and the *it\_interval* member  
 4859 to zero.

4860 2. Periodic

4861 A periodic timer is a timer that is armed with an initial expiration time, again either  
 4862 relative or absolute, and a repetition interval. When the initial expiration occurs, the

4863 timer is reloaded with the repetition interval and continues counting. With the  
 4864 specified facilities, this is accomplished by setting the *it\_value* member of the *value*  
 4865 argument to the desired initial expiration time and the *it\_interval* member to the desired  
 4866 repetition interval.

4867 For both of these types of timers, the time of the initial timer expiration can be specified in  
 4868 two ways:

- 4869 1. Relative (to the current time)
- 4870 2. Absolute

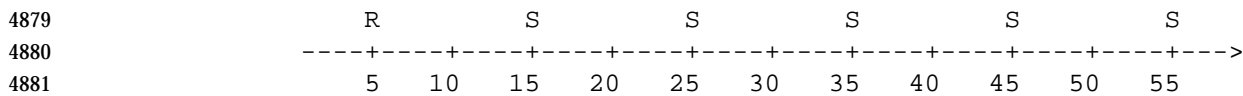
#### 4871 • Examples of Using Realtime Timers

4872 In the diagrams below, *S* indicates a program schedule, *R* shows a schedule method request,  
 4873 and *E* suggests an internal operating system event.

##### 4874 — Periodic Timer: Data Logging

4875 During an experiment, it might be necessary to log realtime data periodically to an  
 4876 internal buffer or to a mass storage device. With a periodic scheduling method, a logging  
 4877 module can be started automatically at fixed time intervals to log the data.

4878 Program schedule is requested every 10 seconds.



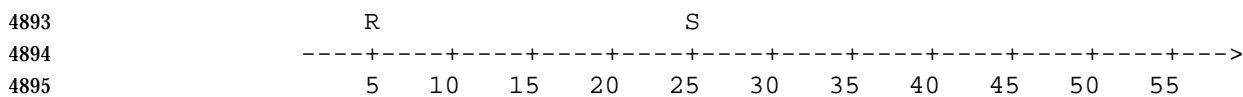
4882 [Time (in Seconds)]

4883 To achieve this type of scheduling using the specified facilities, one would allocate a per-  
 4884 process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be armed via  
 4885 a call to `timer_settime()` with the `TIMER_ABSTIME` flag reset, and with an initial  
 4886 expiration value and a repetition interval of 10 seconds.

##### 4887 — One-shot Timer (Relative Time): Device Initialization

4888 In an emission test environment, large sample bags are used to capture the exhaust from  
 4889 a vehicle. The exhaust is purged from these bags before each and every test. With a one-  
 4890 shot timer, a module could initiate the purge function and then suspend itself for a  
 4891 predetermined period of time while the sample bags are prepared.

4892 Program schedule requested 20 seconds after call is issued.



4896 [Time (in Seconds)]

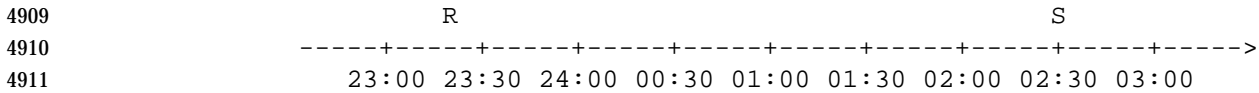
4897 To achieve this type of scheduling using the specified facilities, one would allocate a per-  
 4898 process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be armed via  
 4899 a call to `timer_settime()` with the `TIMER_ABSTIME` flag reset, and with an initial  
 4900 expiration value of 20 seconds and a repetition interval of zero.

4901 Note that if the program wishes merely to suspend itself for the specified interval, it  
 4902 could more easily use `nanosleep()`.

##### 4903 — One-shot Timer (Absolute Time): Data Transmission

4904 The results from an experiment are often moved to a different system within a network  
 4905 for postprocessing or archiving. With an absolute one-shot timer, a module that moves  
 4906 data from a test-cell computer to a host computer can be automatically scheduled on a  
 4907 daily basis.

4908 Program schedule requested for 2:30 a.m.



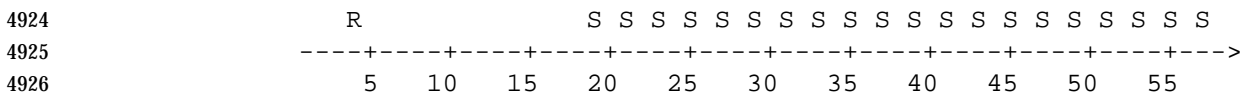
4912 [Time of Day]

4913 To achieve this type of scheduling using the specified facilities, one would allocate a per-  
 4914 process timer based on clock ID CLOCK\_REALTIME. Then the timer would be armed via  
 4915 a call to *timer\_settime()* with the *TIMER\_ABSTIME* flag set, and an initial expiration value  
 4916 equal to 2:30 a.m. of the next day.

4917 — Periodic Timer (Relative Time): Signal Stabilization

4918 Some measurement devices, such as emission analyzers, do not respond instantaneously  
 4919 to an introduced sample. With a periodic timer with a relative initial expiration time, a  
 4920 module that introduces a sample and records the average response could suspend itself  
 4921 for a predetermined period of time while the signal is stabilized and then sample at a  
 4922 fixed rate.

4923 Program schedule requested 15 seconds after call is issued and every 2 seconds thereafter.



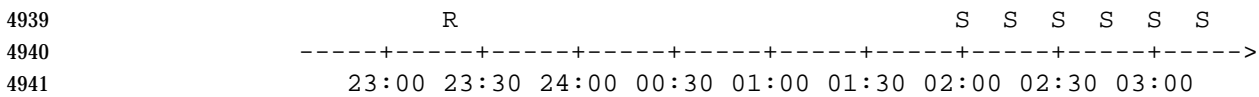
4927 [Time (in Seconds)]

4928 To achieve this type of scheduling using the specified facilities, one would allocate a per-  
 4929 process timer based on clock ID CLOCK\_REALTIME. Then the timer would be armed via  
 4930 a call to *timer\_settime()* with *TIMER\_ABSTIME* flag reset, and with an initial expiration  
 4931 value of 15 seconds and a repetition interval of 2 seconds.

4932 — Periodic Timer (Absolute Time): Work Shift-related Processing

4933 Resource utilization data is useful when time to perform experiments is being scheduled  
 4934 at a facility. With a periodic timer with an absolute initial expiration time, a module can  
 4935 be scheduled at the beginning of a work shift to gather resource utilization data  
 4936 throughout the shift. This data can be used to allocate resources effectively to minimize  
 4937 bottlenecks and delays and maximize facility throughput.

4938 Program schedule requested for 2:00 a.m. and every 15 minutes thereafter.



4942 [Time of Day]

4943 To achieve this type of scheduling using the specified facilities, one would allocate a per-  
 4944 process timer based on clock ID CLOCK\_REALTIME. Then the timer would be armed via  
 4945 a call to *timer\_settime()* with *TIMER\_ABSTIME* flag set, and with an initial expiration  
 4946 value equal to 2:00 a.m. and a repetition interval equal to 15 minutes.

4947 • Relationship of Timers to Clocks

4948 The relationship between clocks and timers armed with an absolute time is straightforward:  
 4949 a timer expiration signal is requested when the associated clock reaches or exceeds the  
 4950 specified time. The relationship between clocks and timers armed with a relative time (an  
 4951 interval) is less obvious, but not unintuitive. In this case, a timer expiration signal is  
 4952 requested when the specified interval, *as measured by the associated clock*, has passed. For the  
 4953 required CLOCK\_REALTIME clock, this allows timer expiration signals to be requested at  
 4954 specified “wall clock” times (absolute), or when a specified interval of “realtime” has passed  
 4955 (relative). For an implementation-defined clock—say, a process virtual time clock—timer  
 4956 expirations could be requested when the process has used a specified total amount of virtual  
 4957 time (absolute), or when it has used a specified *additional* amount of virtual time (relative).

4958 The interfaces also allow flexibility in the implementation of the functions. For example, an  
 4959 implementation could convert all absolute times to intervals by subtracting the clock value at  
 4960 the time of the call from the requested expiration time and “counting down” at the  
 4961 supported resolution. Or it could convert all relative times to absolute expiration time by  
 4962 adding in the clock value at the time of the call and comparing the clock value to the  
 4963 expiration time at the supported resolution. Or it might even choose to maintain absolute  
 4964 times as absolute and compare them to the clock value at the supported resolution for  
 4965 absolute timers, and maintain relative times as intervals and count them down at the  
 4966 resolution supported for relative timers. The choice will be driven by efficiency  
 4967 considerations and the underlying hardware or software clock implementation.

4968 • Data Definitions for Clocks and Timers

4969 IEEE Std. 1003.1-200x uses a time representation capable of supporting nanosecond  
 4970 resolution timers for the following reasons:

- 4971 — To enable IEEE Std. 1003.1-200x to represent those computer systems already using  
 4972 nanosecond or submicrosecond resolution clocks.
- 4973 — To accommodate those per-process timers that might need nanoseconds to specify an  
 4974 absolute value of system-wide clocks, even though the resolution of the per-process timer  
 4975 may only be milliseconds, or *vice versa*.
- 4976 — Because the number of nanoseconds in a second can be represented in 32 bits.

4977 Time values are represented in the **timespec** structure. The *tv\_sec* member is of type **time\_t**  
 4978 so that this member is compatible with time values used by POSIX.1 functions and the ISO C  
 4979 standard. The *tv\_nsec* member is a **signed long** in order to simplify and clarify code that  
 4980 decrements or finds differences of time values. Note that because 1 billion (number of  
 4981 nanoseconds per second) is less than half of the value representable by a signed 32-bit value,  
 4982 it is always possible to add two valid fractional seconds represented as integral nanoseconds  
 4983 without overflowing the signed 32-bit value.

4984 A maximum allowable resolution for the CLOCK\_REALTIME clock of 20 ms (1/50 seconds)  
 4985 was chosen to allow line frequency clocks in European countries to be conforming. 60 Hz  
 4986 clocks in the U.S. will also be conforming, as will finer granularity clocks, although a Strictly  
 4987 Conforming Application cannot assume a granularity of less than 20 ms (1/50 seconds).

4988 The minimum allowable maximum time allowed for the CLOCK\_REALTIME clock and the  
 4989 function *nanosleep()*, and timers created with *clock\_id*=CLOCK\_REALTIME, is determined by  
 4990 the fact that the *tv\_sec* member is of type **time\_t**.

4991 IEEE Std. 1003.1-200x specifies that timer expirations shall not be delivered early, nor shall  
 4992 *nanosleep()* return early due to quantization error. IEEE Std. 1003.1-200x discusses the various  
 4993 implementations of *alarm()* in the rationale and states that implementations that do not

4994 allow alarm signals to occur early are the most appropriate, but refrained from mandating  
4995 this behavior. Because of the importance of predictability to realtime applications,  
4996 IEEE Std. 1003.1-200x takes a stronger stance.

4997 The developers of IEEE Std. 1003.1-200x considered using a time representation that differs  
4998 from POSIX.1b in the second 32 bit of the 64-bit value. Whereas POSIX.1b defines this field  
4999 as a fractional second in nanoseconds, the other methodology defines this as a binary fraction  
5000 of one second, with the radix point assumed before the most significant bit.

5001 POSIX.1b is a software, source-level standard and most of the benefits of the alternate  
5002 representation are enjoyed by hardware implementations of clocks and algorithms. It was  
5003 felt that mandating this format for POSIX.1b clocks and timers would unnecessarily burden  
5004 the application writer with writing, possibly non-portable, multiple precision arithmetic  
5005 packages to perform conversion between binary fractions and integral units such as  
5006 nanoseconds, milliseconds, and so on.

#### 5007 **Rationale for the Monotonic Clock**

5008 For those applications that use time services to achieve realtime behavior, changing the value of  
5009 the clock on which these services rely may cause erroneous timing behavior. For these  
5010 applications, it is necessary to have a monotonic clock which cannot run backwards, and which  
5011 has a maximum clock jump that is required to be documented by the implementation.  
5012 Additionally, it is desirable (but not required by IEEE Std. 1003.1-200x) that the monotonic clock  
5013 increases its value uniformly. This clock should not be affected by changes to the system time;  
5014 for example, to synchronize the clock with an external source or to account for leap seconds.  
5015 Such changes would cause errors in the measurement of time intervals for those time services  
5016 that use the absolute value of the clock.

5017 One could argue that by defining the behavior of time services when the value of a clock is  
5018 changed, deterministic realtime behavior can be achieved. For example, one could specify that  
5019 relative time services should be unaffected by changes in the value of a clock. However, there  
5020 are time services that are based upon an absolute time, but that are essentially intended as  
5021 relative time services. For example, *pthread\_cond\_timedwait()* uses an absolute time to allow it to  
5022 wake up after the required interval despite spurious wakeups. Although sometimes the  
5023 *pthread\_cond\_timedwait()* timeouts are absolute in nature, there are many occasions in which  
5024 they are relative, and their absolute value is determined from the current time plus a relative  
5025 time interval. In this latter case, if the clock changes while the thread is waiting, the wait interval  
5026 will not be the expected length. If a *pthread\_cond\_timedwait()* function were created that would  
5027 take a relative time, it would not solve the problem because to retain the intended “deadline” a  
5028 thread would need to compensate for latency due to the spurious wakeup, and preemption  
5029 between wakeup and the next wait.

5030 The solution is to create a new monotonic clock, whose value does not change except for the  
5031 regular ticking of the clock, and use this clock for implementing the various relative timeouts  
5032 that appear in the different POSIX interfaces, as well as allow *pthread\_cond\_timedwait()* to choose  
5033 this new clock for its timeout. A new *clock\_nanosleep()* function is created to allow an application  
5034 to take advantage of this newly defined clock. Notice that the monotonic clock may be  
5035 implemented using the same hardware clock as the system clock.

5036 Relative timeouts for *sigtimedwait()* and *aio\_suspend()* have been redefined to use the monotonic  
5037 clock, if present. The *alarm()* function has not been redefined, because the same effect but with  
5038 better resolution can be achieved by creating a timer (for which the appropriate clock may be  
5039 chosen).

5040 The *pthread\_cond\_timedwait()* function has been treated in a different way, compared to other  
5041 functions with absolute timeouts, because it is used to wait for an event, and thus it may have a

5042 deadline, while the other timeouts are generally used as an error recovery mechanism, and for  
5043 them the use of the monotonic clock is not so important. Since the desired timeout for the  
5044 *pthread\_cond\_timedwait()* function may either be a relative interval, or an absolute time of day  
5045 deadline, a new initialization attribute has been created for condition variables, to specify the  
5046 clock that shall be used for measuring the timeout in a call to *pthread\_cond\_timedwait()*. In this  
5047 way, if a relative timeout is desired, the monotonic clock will be used; if an absolute deadline is  
5048 required instead, the `CLOCK_REALTIME` or another appropriate clock may be used. This  
5049 capability has not been added to other functions with absolute timeouts because for those  
5050 functions the expected use of the timeout is mostly to prevent errors, and not so often to meet  
5051 precise deadlines. As a consequence, the complexity of adding this capability is not justified by  
5052 its perceived application usage.

5053 The *nanosleep()* function has not been modified with the introduction of the monotonic clock.  
5054 Instead, a new *clock\_nanosleep()* function has been created, in which the desired clock may be  
5055 specified in the function call.

5056 • History of Resolution Issues

5057 Due to the shift from relative to absolute timeouts in IEEE Std. 1003.1d-1999, the  
5058 amendments to the *sem\_timedwait()*, *pthread\_mutex\_timedlock()*, *mq\_timedreceive()*, and  
5059 *mq\_timedsend()* functions of that standard have been removed. Those amendments specified  
5060 that `CLOCK_MONOTONIC` would be used for the (relative) timeouts if the Monotonic  
5061 Clock option was supported.

5062 Having these functions continue to be tied solely to `CLOCK_MONOTONIC` would not  
5063 work. Since the absolute value of a time value obtained from `CLOCK_MONOTONIC` is  
5064 unspecified, under the absolute timeouts interface, applications would behave differently  
5065 depending on whether the Monotonic Clock option was supported or not (because the  
5066 absolute value of the clock would have different meanings in either case).

5067 Two options were considered:

- 5068 1. Leave the current behavior unchanged, which specifies the `CLOCK_REALTIME` clock  
5069 for these (absolute) timeouts, to allow portability of applications between  
5070 implementations supporting or not the Monotonic Clock option.
- 5071 2. Modify these functions in the way that *pthread\_cond\_timedwait()* was modified to allow  
5072 a choice of clock, so that an application could use `CLOCK_REALTIME` when it is trying  
5073 to achieve an absolute timeout and `CLOCK_MONOTONIC` when it is trying to achieve  
5074 a relative timeout.

5075 It was decided that the features of `CLOCK_MONOTONIC` are not as critical to these  
5076 functions as they are to *pthread\_cond\_timedwait()*. The *pthread\_cond\_timedwait()* function is  
5077 given a relative timeout; the timeout may represent a deadline for an event. When these  
5078 functions are given relative timeouts, the timeouts are typically for error recovery purposes  
5079 and need not be so precise.

5080 Therefore, it was decided that these functions should be tied to `CLOCK_REALTIME` and not  
5081 complicated by being given a choice of clock.



5082           **Execution Time Monitoring**

## 5083           • Introduction

5084           The main goals of the execution time monitoring facilities defined in this chapter are to  
5085           measure the execution time of processes and threads and to allow an application to establish  
5086           CPU time limits for these entities.

5087           The analysis phase of time-critical realtime systems often relies on the measurement of  
5088           execution times of individual threads or processes to determine whether the timing  
5089           requirements will be met. Also, performance analysis techniques for soft deadline realtime  
5090           systems rely heavily on the determination of these execution times. The execution time  
5091           monitoring functions provide application developers with the ability to measure these  
5092           execution times online and open the possibility of dynamic execution-time analysis and  
5093           system reconfiguration, if required.

5094           The second goal of allowing an application to establish execution time limits for individual  
5095           processes or threads and detecting when they overrun allows program robustness to be  
5096           increased by enabling online checking of the execution times.

5097           If errors are detected—possibly because of erroneous program constructs, the existence of  
5098           errors in the analysis phase, or a burst of event arrivals—online detection and recovery is  
5099           possible in a portable way. This feature can be extremely important for many time-critical  
5100           applications. Other applications require trapping CPU-time errors as a normal way to exit an  
5101           algorithm; for instance, some realtime artificial intelligence applications trigger a number of  
5102           independent inference processes of varying accuracy and speed, limit how long they can run,  
5103           and pick the best answer available when time runs out. In many periodic systems, overrun  
5104           processes are simply restarted in the next resource period, after necessary end-of-period  
5105           actions have been taken. This allows algorithms that are inherently data-dependent to be  
5106           made predictable.

5107           The interface that appears in this chapter defines a new type of clock, the CPU-time clock,  
5108           which measures execution time. Each process or thread can invoke the clock and timer  
5109           functions defined in POSIX.1 to use them. Functions are also provided to access the CPU-  
5110           time clock of other processes or threads to enable remote monitoring of these clocks.  
5111           Monitoring of threads of other processes is not supported, since these threads are not visible  
5112           from outside of their own process with the interfaces defined in POSIX.1.

## 5113           • Execution Time Monitoring Interface

5114           The clock and timer interface defined in POSIX.1 historically only defined one clock, which  
5115           measures wall-clock time. The requirements for measuring execution time of processes and  
5116           threads, and setting limits to their execution time by detecting when they overrun, can be  
5117           accomplished with that interface if a new kind of clock is defined. These new clocks measure  
5118           execution time, and one is associated with each process and with each thread. The clock  
5119           functions currently defined in POSIX.1 can be used to read and set these CPU-time clocks,  
5120           and timers can be created using these clocks as their timing base. These timers can then be  
5121           used to send a signal when some specified execution time has been exceeded. The CPU-time  
5122           clocks of each process or thread can be accessed by using the symbols  
5123           CLOCK\_PROCESS\_CPUTIME\_ID or CLOCK\_THREAD\_CPUTIME\_ID.

5124           The clock and timer interface defined in POSIX.1 and extended with the new kind of CPU-  
5125           time clock would only allow processes or threads to access their own CPU-time clocks.  
5126           However, many realtime systems require the possibility of monitoring the execution time of  
5127           processes or threads from independent monitoring entities. In order to allow applications to  
5128           construct independent monitoring entities that do not require cooperation from or  
5129           modification of the monitored entities, two functions have been added: *clock\_getcpuclckid()*,

5130 for accessing CPU-time clocks of other processes, and *pthread\_getcpuclockid()*, for accessing  
5131 CPU-time clocks of other threads. These functions return the clock identifier associated with  
5132 the process or thread specified in the call. These clock IDs can then be used in the rest of the  
5133 clock function calls.

5134 The clocks accessed through these functions could also be used as a timing base for the  
5135 creation of timers, thereby allowing independent monitoring entities to limit the CPU-time  
5136 consumed by other entities. However, this possibility would imply additional complexity  
5137 and overhead because of the need to maintain a timer queue for each process or thread, to  
5138 store the different expiration times associated with timers created by different processes or  
5139 threads. The working group decided this additional overhead was not justified by  
5140 application requirements. Therefore, creation of timers attached to the CPU-time clocks of  
5141 other processes or threads has been specified as implementation-defined.

5142 • Overhead Considerations

5143 The measurement of execution time may introduce additional overhead in the thread  
5144 scheduling, because of the need to keep track of the time consumed by each of these entities.  
5145 In library-level implementations of threads, the efficiency of scheduling could be somehow  
5146 compromised because of the need to make a kernel call, at each context switch, to read the  
5147 process CPU-time clock. Consequently, a thread creation attribute called *cpu-clock-*  
5148 *requirement* was defined, to allow threads to disconnect their respective CPU-time clocks.  
5149 However, the Ballot Group considered that this attribute itself introduced some overhead,  
5150 and that in current implementations it was not worth the effort. Therefore, the attribute was  
5151 deleted, and thus thread CPU-time clocks are required for all threads if the Thread CPU-Time  
5152 Clocks option is supported.

5153 • Accuracy of CPU-time Clocks

5154 The mechanism used to measure the execution time of processes and threads is specified in  
5155 IEEE Std. 1003.1-200x as implementation-defined. The reason for this is that both the  
5156 underlying hardware and the implementation architecture have a very strong influence on  
5157 the accuracy achievable for measuring CPU time. For some implementations, the  
5158 specification of strict accuracy requirements would represent very large overheads, or even  
5159 the impossibility of being implemented.

5160 Since the mechanism for measuring execution time is implementation-defined, realtime  
5161 applications will be able to take advantage of accurate implementations using a portable  
5162 interface. Of course, strictly conforming applications cannot rely on any particular degree of  
5163 accuracy, in the same way as they cannot rely on a very accurate measurement of wall clock  
5164 time. There will always exist applications whose accuracy or efficiency requirements on the  
5165 implementation are more rigid than the values defined in IEEE Std. 1003.1-200x or any other  
5166 standard.

5167 In any case, there is a minimum set of characteristics that realtime applications would expect  
5168 from most implementations. One such characteristic is that the sum of all the execution times  
5169 of all the threads in a process equals the process execution time, when no CPU-time clocks  
5170 are disabled. This need not always be the case because implementations may differ in how  
5171 they account for time during context switches. Another characteristic is that the sum of the  
5172 execution times of all processes in a system equals the number of processors, multiplied by  
5173 the elapsed time, assuming that no processor is idle during that elapsed time. However, in  
5174 some systems it might not be possible to relate CPU-time to elapsed time. For example, in a  
5175 heterogeneous multi-processor system in which each processor runs at a different speed, an  
5176 implementation may choose to define each “second” of CPU-time to be a certain number of  
5177 “cycles” that a CPU has executed.

- 5178           • Existing Practice
- 5179           Measuring and limiting the execution time of each concurrent activity are common features  
5180           of most industrial implementations of realtime systems. Almost all critical realtime systems  
5181           are currently built upon a cyclic executive. With this approach, a regular timer interrupt kicks  
5182           off the next sequence of computations. It also checks that the current sequence has  
5183           completed. If it has not, then some error recovery action can be undertaken (or at least an  
5184           overrun is avoided). Current software engineering principles and the increasing complexity  
5185           of software are driving application developers to implement these systems on multi-  
5186           threaded or multi-process operating systems. Therefore, if a POSIX operating system is to be  
5187           used for this type of application, then it must offer the same level of protection.
- 5188           Execution time clocks are also common in most UNIX implementations, although these  
5189           clocks usually have requirements different from those of realtime applications. The POSIX.1  
5190           *times()* function supports the measurement of the execution time of the calling process, and  
5191           its terminated child processes. This execution time is measured in clock ticks and is supplied  
5192           as two different values with the user and system execution times, respectively. BSD supports  
5193           the function *getrusage()*, which allows the calling process to get information about the  
5194           resources used by itself and/or all of its terminated child processes. The resource usage  
5195           includes user and system CPU time. Some UNIX systems have options to specify high  
5196           resolution (up to one microsecond) CPU time clocks using the *times()* or the *getrusage()*  
5197           functions.
- 5198           The *times()* and *getrusage()* interfaces do not meet important realtime requirements, such as  
5199           the possibility of monitoring execution time from a different process or thread, or the  
5200           possibility of detecting an execution time overrun. The latter requirement is supported in  
5201           some UNIX implementations that are able to send a signal when the execution time of a  
5202           process has exceeded some specified value. For example, BSD defines the functions  
5203           *getitimer()* and *setitimer()*, which can operate either on a realtime clock (wall-clock), or on  
5204           virtual-time or profile-time clocks which measure CPU time in two different ways. These  
5205           functions do not support access to the execution time of other processes.
- 5206           IBM's MVS operating system supports per-process and per-thread execution time clocks. It  
5207           also supports limiting the execution time of a given process.
- 5208           Given all this existing practice, the working group considered that the POSIX.1 clocks and  
5209           timers interface was appropriate to meet most of the requirements that realtime applications  
5210           have for execution time clocks. Functions were added to get the CPU time clock IDs, and to  
5211           allow/disallow the thread CPU time clocks (in order to preserve the efficiency of some  
5212           implementations of threads).
- 5213           • Clock Constants
- 5214           The definition of the manifest constants `CLOCK_PROCESS_CPUTIME_ID` and  
5215           `CLOCK_THREAD_CPUTIME_ID` allows processes or threads, respectively, to access their  
5216           own execution-time clocks. However, given a process or thread, access to its own execution-  
5217           time clock is also possible if the clock ID of this clock is obtained through a call to  
5218           *clock\_getcpuclockid()* or *pthread\_getcpuclockid()*. Therefore, these constants are not necessary  
5219           and could be deleted to make the interface simpler. Their existence saves one system call in  
5220           the first access to the CPU-time clock of each process or thread. The working group  
5221           considered this issue and decided to leave the constants in IEEE Std. 1003.1-200x because  
5222           they are closer to the POSIX.1b use of clock identifiers.
- 5223           • Library Implementations of Threads
- 5224           In library implementations of threads, kernel entities and library threads can coexist. In this  
5225           case, if the CPU-time clocks are supported, most of the clock and timer functions will need to

5226 have two implementations: one in the thread library, and one in the system calls library. The  
 5227 main difference between these two implementations is that the thread library  
 5228 implementation will have to deal with clocks and timers that reside in the thread space,  
 5229 while the kernel implementation will operate on timers and clocks that reside in kernel space.  
 5230 In the library implementation, if the clock ID refers to a clock that resides in the kernel, a  
 5231 kernel call will have to be made. The correct version of the function can be chosen by  
 5232 specifying the appropriate order for the libraries during the link process.

5233 • History of Resolution Issues: Deletion of the *enable* Attribute

5234 In the draft corresponding to the first balloting round, CPU-time clocks had an attribute  
 5235 called *enable*. This attribute was introduced by the working group to allow implementations  
 5236 to avoid the overhead of measuring execution time for those processes or threads for which  
 5237 this measurement was not required. However, the *enable* attribute got several ballot  
 5238 objections. The main reason was that processes are already required to measure execution  
 5239 time by the POSIX.1 *times()* function. Consequently, the *enable* attribute was considered  
 5240 unnecessary, and was deleted from the draft.

#### 5241 **Rationale Relating to Timeouts**

5242 • Requirements for Timeouts

5243 Realtime systems which must operate reliably over extended periods without human  
 5244 intervention are characteristic in embedded applications such as avionics, machine control,  
 5245 and space exploration, as well as more mundane applications such as cable TV, security  
 5246 systems, and plant automation. A multi-tasking paradigm, in which many independent  
 5247 and/or cooperating software functions relinquish the processor(s) while waiting for a  
 5248 specific stimulus, resource, condition, or operation completion, is very useful in producing  
 5249 well engineered programs for such systems. For such systems to be robust and fault-tolerant,  
 5250 expected occurrences that are unduly delayed or that never occur must be detected so that  
 5251 appropriate recovery actions may be taken. This is difficult if there is no way for a task to  
 5252 regain control of a processor once it has relinquished control (blocked) awaiting an  
 5253 occurrence which, perhaps because of corrupted code, hardware malfunction, or latent  
 5254 software bugs, will not happen when expected. Therefore, the common practice in realtime  
 5255 operating systems is to provide a capability to timeout such blocking services. Although  
 5256 there are several methods to achieve this already defined by POSIX, none are as reliable or  
 5257 efficient as initiating a timeout simultaneously with initiating a blocking service. This is  
 5258 especially critical in hard-realtime embedded systems because the processors typically have  
 5259 little time reserve, and allowed fault recovery times are measured in milliseconds rather than  
 5260 seconds.

5261 The working group largely agreed that such timeouts were necessary and ought to become  
 5262 part of IEEE Std. 1003.1-200x, particularly vendors of realtime operating systems whose  
 5263 customers had already expressed a strong need for timeouts. There was some resistance to  
 5264 inclusion of timeouts in IEEE Std. 1003.1-200x because the desired effect, fault tolerance,  
 5265 could, in theory, be achieved using existing facilities and alternative software designs, but  
 5266 there was no compelling evidence that realtime system designers would embrace such  
 5267 designs at the sacrifice of performance and/or simplicity.

5268 • Which Services should be Timed Out?

5269 Originally, the working group considered the prospect of providing timeouts on all blocking  
 5270 services, including those currently existing in POSIX.1, POSIX.1b, and POSIX.1c, and future  
 5271 interfaces to be defined by other working groups, as sort of a general policy. This was rather  
 5272 quickly rejected because of the scope of such a change, and the fact that many of those  
 5273 services would not normally be used in a realtime context. More traditional timesharing

5274 solutions to timeout would suffice for most of the POSIX.1 interfaces, while others had  
 5275 asynchronous alternatives which, while more complex to utilize, would be adequate for  
 5276 some realtime and all non-realtime applications.

5277 The list of potential candidates for timeouts was narrowed to the following for further  
 5278 consideration:

5279 — POSIX.1b

5280 — *sem\_wait()*

5281 — *mq\_receive()*

5282 — *mq\_send()*

5283 — *lio\_listio()*

5284 — *aio\_suspend()*

5285 — *sigwait()* (timeout already implemented by *sigtimedwait()*)

5286 — POSIX.1c

5287 — *pthread\_mutex\_lock()*

5288 — *pthread\_join()*

5289 — *pthread\_cond\_wait()* (timeout already implemented by *pthread\_cond\_timedwait()*)

5290 — POSIX.1

5291 — *read()*

5292 — *write()*

5293 After further review by the working group, the *lio\_listio()*, *read()*, and *write()* functions (all  
 5294 forms of blocking synchronous I/O) were eliminated from the list because of the following:

5295 — Asynchronous alternatives exist

5296 — Timeouts can be implemented, albeit non-portably, in device drivers

5297 — A strong desire not to introduce modifications to POSIX.1 interfaces

5298 The working group ultimately rejected *pthread\_join()* since both that interface and a timed  
 5299 variant of that interface are non-minimal and may be implemented as a library function. See  
 5300 below for a library implementation of *pthread\_join()*.

5301 Thus, there was a consensus among the working group members to add timeouts to 4 of the  
 5302 remaining 5 functions (the timeout for *aio\_suspend()* was ultimately added directly to  
 5303 POSIX.1b, while the others were added by POSIX.1d). However, *pthread\_mutex\_lock()*  
 5304 remained contentious.

5305 Many feel that *pthread\_mutex\_lock()* falls into the same class as the other functions; that is, it  
 5306 is desirable to timeout a mutex lock because a mutex may fail to be unlocked due to errant or  
 5307 corrupted code in a critical section (looping or branching outside of the unlock code), and  
 5308 therefore is equally in need of a reliable, simple, and efficient timeout. In fact, since mutexes  
 5309 are intended to guard small critical sections, most *pthread\_mutex\_lock()* calls would be  
 5310 expected to obtain the lock without blocking nor utilizing any kernel service, even in  
 5311 implementations of threads with global contention scope; the timeout alternative need only  
 5312 be considered after it is determined that the thread must block.

5313 Those opposed to timing out mutexes feel that the very simplicity of the mutex is  
 5314 compromised by adding a timeout semantic, and that to do so is senseless. They claim that if

5315 a timed mutex is really deemed useful by a particular application, then it can be constructed  
 5316 from the facilities already in POSIX.1b and POSIX.1c. The following two C-language library  
 5317 implementations of mutex locking with timeout represent the solutions offered (in both  
 5318 implementations, the timeout parameter is specified as absolute time, not relative time as in  
 5319 the proposed POSIX.1c interfaces).

5320 • Spinlock Implementation

```

5321 #include <pthread.h>
5322 #include <time.h>
5323 #include <errno.h>

5324 int pthread_mutex_timedlock(pthread_mutex_t *mutex,
5325 const struct timespec *timeout)
5326 {
5327 struct timespec timenow;

5328 while (pthread_mutex_trylock(mutex) == EBUSY)
5329 {
5330 clock_gettime(CLOCK_REALTIME, &timenow);
5331 if (timespec_cmp(&timenow, timeout) >= 0)
5332 {
5333 return ETIMEDOUT;
5334 }
5335 pthread_yield();
5336 }
5337 return 0;
5338 }

```

5339 The Spinlock implementation is generally unsuitable for any application using priority-based  
 5340 thread scheduling policies such as SCHED\_FIFO or SCHED\_RR, since the mutex could  
 5341 currently be held by a thread of lower priority within the same allocation domain, but since  
 5342 the waiting thread never blocks, only threads of equal or higher priority will ever run, and  
 5343 the mutex cannot be unlocked. Setting priority inheritance or priority ceiling protocol on the  
 5344 mutex does not solve this problem, since the priority of a mutex owning thread is only  
 5345 boosted if higher priority threads are blocked waiting for the mutex; clearly not the case for  
 5346 this spinlock.

5347 • Condition Wait Implementation

```

5348 #include <pthread.h>
5349 #include <time.h>
5350 #include <errno.h>

5351 struct timed_mutex
5352 {
5353 int locked;
5354 pthread_mutex_t mutex;
5355 pthread_cond_t cond;
5356 };
5357 typedef struct timed_mutex timed_mutex_t;

5358 int timed_mutex_lock(timed_mutex_t *tm,
5359 const struct timespec *timeout)
5360 {
5361 int timedout=FALSE;
5362 int error_status;

```

```

5363 pthread_mutex_lock(&tm->mutex);
5364 while (tm->locked && !timedout)
5365 {
5366 if ((error_status=pthread_cond_timedwait(&tm->cond,
5367 &tm->mutex,
5368 timeout))!=0)
5369 {
5370 if (error_status==ETIMEDOUT) timedout = TRUE;
5371 }
5372 }
5373 if(timedout)
5374 {
5375 pthread_mutex_unlock(&tm->mutex);
5376 return ETIMEDOUT;
5377 }
5378 else
5379 {
5380 tm->locked = TRUE;
5381 pthread_mutex_unlock(&tm->mutex);
5382 return 0;
5383 }
5384 }
5385 void timed_mutex_unlock(timed_mutex_t *tm)
5386 {
5387 pthread_mutex_lock(&tm->mutex); / for case assignment not atomic /
5388 tm->locked = FALSE;
5389 pthread_mutex_unlock(&tm->mutex);
5390 pthread_cond_signal(&tm->cond);
5391 }

```

5392 The Condition Wait implementation effectively substitutes the *pthread\_cond\_timedwait()*  
5393 function (which is currently timed out) for the desired *pthread\_mutex\_timedlock()*. Since waits  
5394 on condition variables currently do not include protocols which avoid priority inversion, this  
5395 method is generally unsuitable for realtime applications because it does not provide the same  
5396 priority inversion protection as the untimed *pthread\_mutex\_lock()*. Also, for any given  
5397 implementations of the current mutex and condition variable primitives, this library  
5398 implementation has a performance cost at least 2.5 times that of the untimed  
5399 *pthread\_mutex\_lock()* even in the case where the timed mutex is readily locked without  
5400 blocking (the interfaces required for this case are shown in bold). Even in uniprocessors or  
5401 where assignment is atomic, at least an additional *pthread\_cond\_signal()* is required.  
5402 *pthread\_mutex\_timedlock()* could be implemented at effectively no performance penalty in  
5403 this case because the timeout parameters need only be considered after it is determined that  
5404 the mutex cannot be locked immediately.

5405 Thus it has not yet been shown that the full semantics of mutex locking with timeout can be  
5406 efficiently and reliably achieved using existing interfaces. Even if the existence of an  
5407 acceptable library implementation were proven, it is difficult to justify why the interface  
5408 itself should not be made portable, especially considering approval for the other four  
5409 timeouts.

5410 • Rationale for Library Implementation of *pthread\_timedjoin()*

```

5411 Library implementation of pthread_timedjoin():
5412 /*
5413 * Construct a thread variety entirely from existing functions
5414 * with which a join can be done, allowing the join to time out.
5415 */
5416 #include <pthread.h>
5417 #include <time.h>
5418
5418 struct timed_thread {
5419 pthread_t t;
5420 pthread_mutex_t m;
5421 int exiting;
5422 pthread_cond_t exit_c;
5423 void *(*start_routine)(void *arg);
5424 void *arg;
5425 void *status;
5426 };
5427
5427 typedef struct timed_thread *timed_thread_t;
5428 static pthread_key_t timed_thread_key;
5429 static pthread_once_t timed_thread_once = PTHREAD_ONCE_INIT;
5430
5430 static void timed_thread_init()
5431 {
5432 pthread_key_create(&timed_thread_key, NULL);
5433 }
5434
5434 static void *timed_thread_start_routine(void *args)
5435
5435 /*
5436 * Routine to establish thread-specific data value and run the actual
5437 * thread start routine which was supplied to timed_thread_create().
5438 */
5439 {
5440 timed_thread_t tt = (timed_thread_t) args;
5441
5441 pthread_once(&timed_thread_once, timed_thread_init);
5442 pthread_setspecific(timed_thread_key, (void *)tt);
5443 timed_thread_exit((tt->start_routine)(tt->arg));
5444 }
5445
5445 int timed_thread_create(timed_thread_t ttp, const pthread_attr_t *attr,
5446 void *(*start_routine)(void *), void *arg)
5447
5447 /*
5448 * Allocate a thread which can be used with timed_thread_join().
5449 */
5450 {
5451 timed_thread_t tt;
5452 int result;
5453
5453 tt = (timed_thread_t) malloc(sizeof(struct timed_thread));
5454 pthread_mutex_init(&tt->m, NULL);
5455 tt->exiting = FALSE;
5456 pthread_cond_init(&tt->exit_c, NULL);
5457 tt->start_routine = start_routine;

```



```

5458 tt->arg = arg;
5459 tt->status = NULL;

5460 if ((result = pthread_create(&tt->t, attr,
5461 timed_thread_start_routine, (void *)tt)) != 0) {
5462 free(tt);
5463 return result;
5464 }

5465 pthread_detach(tt->t);
5466 ttp = tt;
5467 return 0;
5468 }

5469 int timed_thread_join(timed_thread_t tt,
5470 struct timespec *timeout,
5471 void **status)
5472 {
5473 int result;

5474 pthread_mutex_lock(&tt->m);
5475 result = 0;
5476 /*
5477 * Wait until the thread announces that it is exiting,
5478 * or until timeout.
5479 */
5480 while (result == 0 && ! tt->exiting) {
5481 result = pthread_cond_timedwait(&tt->exit_c, &tt->m, timeout);
5482 }
5483 pthread_mutex_unlock(&tt->m);
5484 if (result == 0 && tt->exiting) {
5485 *status = tt->status;
5486 free((void *)tt);
5487 return result;
5488 }
5489 return result;
5490 }

5491 void timed_thread_exit(void *status)
5492 {
5493 timed_thread_t tt;
5494 void *specific;

5495 if ((specific=pthread_getspecific(timed_thread_key)) == NULL){
5496 /*
5497 * Handle cases which won't happen with correct usage.
5498 */
5499 pthread_exit(NULL);
5500 }
5501 tt = (timed_thread_t) specific;
5502 pthread_mutex_lock(&tt->m);
5503 /*
5504 * Tell a joiner that we're exiting.
5505 */
5506 tt->status = status;

```

```

5507 tt->exiting = TRUE;
5508 pthread_cond_signal(&tt->exit_c);
5509 pthread_mutex_unlock(&tt->m);
5510 /*
5511 * Call pthread_exit() to call destructors and really
5512 * exit the thread.
5513 */
5514 pthread_exit(NULL);
5515 }

```

5516 The *pthread\_join()* C-language example shown above demonstrates that it is possible, using  
5517 existing pthread facilities, to construct a variety of thread which allows for joining such a  
5518 thread, but which allows the join operation to time out. It does this by using a  
5519 *pthread\_cond\_timedwait()* to wait for the thread to exit. A **timed\_thread\_t** descriptor structure  
5520 is used to pass parameters from the creating thread to the created thread, and from the  
5521 exiting thread to the joining thread. This implementation is roughly equivalent to what a  
5522 normal *pthread\_join()* implementation would do, with the single change being that  
5523 *pthread\_cond\_timedwait()* is used in place of a simple *pthread\_cond\_wait()*.

5524 Since it is possible to implement such a facility entirely from existing pthread interfaces, and  
5525 with roughly equal efficiency and complexity to an implementation which would be  
5526 provided directly by a pthreads implementation, it was the consensus of the working group  
5527 members that any *pthread\_timedjoin()* facility would be unnecessary, and should not be  
5528 provided.

#### 5529 • Form of the Timeout Interfaces

5530 The working group considered a number of alternative ways to add timeouts to blocking  
5531 services. At first, a system interface which would specify a one-shot or persistent timeout to  
5532 be applied to subsequent blocking services invoked by the calling process or thread was  
5533 considered because it allowed all blocking services to be timed out in a uniform manner with  
5534 a single additional interface; this was rather quickly rejected because it could easily result in  
5535 the wrong services being timed out.

5536 It was suggested that a timeout value might be specified as an attribute of the object  
5537 (semaphore, mutex, message queue, and so on), but there was no consensus on this, either on  
5538 a case-by-case basis or for all timeouts.

5539 Looking at the two existing timeouts for blocking services indicates that the working group  
5540 members favor a separate interface for the timed version of a function. However,  
5541 *pthread\_cond\_timedwait()* utilizes an absolute timeout value while *sigtimedwait()* uses a  
5542 relative timeout value. The working group members agreed that relative timeout values are  
5543 appropriate where the timeout mechanism's primary use was to deal with an unexpected or  
5544 error situation, but they are inappropriate when the timeout must expire at a particular time,  
5545 or before a specific deadline. For the timeouts being introduced in IEEE Std. 1003.1-200x, the  
5546 working group considered allowing both relative and absolute timeouts as is done with  
5547 POSIX.1b timers, but ultimately favored the simpler absolute timeout form.

5548 An absolute time measure can be easily implemented on top of an interface that specifies  
5549 relative time, by reading the clock, calculating the difference between the current time and  
5550 the desired wake-up time, and issuing a relative timeout call. But there is a race condition  
5551 with this approach because the thread could be preempted after reading the clock, but before  
5552 making the timed out call; in this case, the thread would be awakened later than it should  
5553 and, thus, if the wake up time represented a deadline, it would miss it.

5554 There is also a race condition when trying to build a relative timeout on top of an interface  
5555 that specifies absolute timeouts. In this case, we would have to read the clock to calculate the  
5556 absolute wake-up time as the sum of the current time plus the relative timeout interval. In  
5557 this case, if the thread is preempted after reading the clock but before making the timed out  
5558 call, the thread would be awakened earlier than desired.

5559 But the race condition with the absolute timeouts interface is not as bad as the one that  
5560 happens with the relative timeout interface, because there are simple workarounds. For the  
5561 absolute timeouts interface, if the timing requirement is a deadline, we can still meet this  
5562 deadline because the thread woke up earlier than the deadline. If the timeout is just used as  
5563 an error recovery mechanism, the precision of timing is not really important. If the timing  
5564 requirement is that between actions A and B a minimum interval of time must elapse, we can  
5565 safely use the absolute timeout interface by reading the clock after action A has been started.  
5566 It could be argued that, since the call with the absolute timeout is atomic from the  
5567 application point of view, it is not possible to read the clock after action A, if this action is  
5568 part of the timed out call. But if we look at the nature of the calls for which we specify  
5569 timeouts (locking a mutex, waiting for a semaphore, waiting for a message, or waiting until  
5570 there is space in a message queue), the timeouts that an application would build on these  
5571 actions would not be triggered by these actions themselves, but by some other external  
5572 action. For example, if we want to wait for a message to arrive to a message queue, and wait  
5573 for at least 20 milliseconds, this time interval would start to be counted from some event that  
5574 would trigger both the action that produces the message, as well as the action that waits for  
5575 the message to arrive, and not by the wait-for-message operation itself. In this case, we could  
5576 use the workaround proposed above.

5577 For these reasons, the absolute timeout is preferred over the relative timeout interface.

## 5578 **B.2.9 Threads**

5579 Threads will normally be more expensive than subroutines (or functions, routines, and so on) if  
5580 specialized hardware support is not provided. Nevertheless, threads should be sufficiently  
5581 efficient to encourage their use as a medium to fine-grained structuring mechanism for  
5582 parallelism in an application. Structuring an application using threads then allows it to take  
5583 immediate advantage of any underlying parallelism available in the host environment. This  
5584 means implementors are encouraged to optimize for fast execution at the possible expense of  
5585 efficient utilization of storage. For example, a common thread creation technique is to cache  
5586 appropriate thread data structures. That is, rather than releasing system resources, the  
5587 implementation retains these resources and reuses them when the program next asks to create a  
5588 new thread. If this reuse of thread resources is to be possible, there has to be very little unique  
5589 state associated with each thread, because any such state has to be reset when the thread is  
5590 reused.

### 5591 **Thread Creation Attributes**

5592 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to  
5593 support probable future standardization in these areas without requiring that the interface itself  
5594 be changed. Attributes objects provide clean isolation of the configurable aspects of threads. For  
5595 example, “stack size” is an important attribute of a thread, but it cannot be expressed portably.  
5596 When porting a threaded program, stack sizes often need to be adjusted. The use of attributes  
5597 objects can help by allowing the changes to be isolated in a single place, rather than being spread  
5598 across every instance of thread creation.

5599 Attributes objects can be used to set up *classes* of threads with similar attributes; for example,  
5600 “threads with large stacks and high priority” or “threads with minimal stacks”. These classes  
5601 can be defined in a single place and then referenced wherever threads need to be created.

5602 Changes to “class” decisions become straightforward, and detailed analysis of each  
5603 *pthread\_create()* call is not required.

5604 The attributes objects are defined as opaque types as an aid to extensibility. If these objects had  
5605 been specified as structures, adding new attributes would force recompilation of all multi-  
5606 threaded programs when the attributes objects are extended; this might not be possible if  
5607 different program components were supplied by different vendors.

5608 Additionally, opaque attributes objects present opportunities for improving performance.  
5609 Argument validity can be checked once when attributes are set, rather than each time a thread is  
5610 created. Implementations will often need to cache kernel objects that are expensive to create.  
5611 Opaque attributes objects provide an efficient mechanism to detect when cached objects become  
5612 invalid due to attribute changes.

5613 Because assignment is not necessarily defined on a given opaque type, implementation-  
5614 dependent default values cannot be defined in a portable way. The solution to this problem is to  
5615 allow attribute objects to be initialized dynamically by attributes object initialization functions,  
5616 so that default values can be supplied automatically by the implementation.

5617 The following proposal was provided as a suggested alternative to the supplied attributes:

- 5618 1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to  
5619 the initialization routines (*pthread\_create()*, *pthread\_mutex\_init()*, *pthread\_cond\_init()*). The  
5620 parameter containing the flags should be an opaque type for extensibility. If no flags are  
5621 set in the parameter, then the objects are created with default characteristics. An  
5622 implementation may specify implementation-defined flag values and associated behavior.
- 5623 2. If further specialization of mutexes and condition variables is necessary, implementations  
5624 may specify additional procedures that operate on the **pthread\_mutex\_t** and  
5625 **pthread\_cond\_t** objects (instead of on attributes objects).

5626 The difficulties with this solution are:

- 5627 1. A bitmask is not opaque if bits have to be set into bit-vector attributes objects using  
5628 explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**,  
5629 application programmers need to know the location of each bit. If bits are set or read by  
5630 encapsulation (that is, *get\*()* or *set\*()* functions), then the bitmask is merely an  
5631 implementation of attributes objects as currently defined and should not be exposed to the  
5632 programmer.
- 5633 2. Many attributes are not Boolean or very small integral values. For example, scheduling  
5634 policy may be placed in 3 bits or 4 bits, but priority requires 5 bits or more, thereby taking  
5635 up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this,  
5636 the bitmask can only reasonably control whether particular attributes are set or not, and it  
5637 cannot serve as the repository of the value itself. The value needs to be specified as a  
5638 function parameter (which is non-extensible), or by setting a structure field (which is non-  
5639 opaque), or by *get\*()* and *set\*()* functions (making the bitmask a redundant addition to the  
5640 attributes objects).

5641 Stack size is defined as an optional attribute because the very notion of a stack is inherently  
5642 machine-dependent. Some implementations may not be able to change the size of the stack, for  
5643 example, and others may not need to because stack pages may be discontinuous and can be  
5644 allocated and released on demand.

5645 The attribute mechanism has been designed in large measure for extensibility. Future extensions  
5646 to the attribute mechanism or to any attributes object defined in IEEE Std. 1003.1-200x has to be  
5647 done with care so as not to affect binary-compatibility.

5648 Attribute objects, even if allocated by means of dynamic allocation functions such as *malloc()*,  
5649 may have their size fixed at compile time. This means, for example, a *pthread\_create()* in an  
5650 implementation with extensions to the **pthread\_attr\_t** cannot look beyond the area that the  
5651 binary application assumes is valid. This suggests that implementations should maintain a size  
5652 field in the attributes object, as well as possibly version information, if extensions in different  
5653 directions (possibly by different vendors) are to be accommodated.

#### 5654 **Thread Implementation Models**

5655 There are various thread implementation models. At one end of the spectrum is the “library-  
5656 thread model”. In such a model, the threads of a process are not visible to the operating system  
5657 kernel, and the threads are not kernel scheduled entities. The process is the only kernel  
5658 scheduled entity. The process is scheduled onto the processor by the kernel according to the  
5659 scheduling attributes of the process. The threads are scheduled onto the single kernel scheduled  
5660 entity (the process) by the runtime library according to the scheduling attributes of the threads.  
5661 A problem with this model is that it constrains concurrency. Since there is only one kernel  
5662 scheduled entity (namely, the process), only one thread per process can execute at a time. If the  
5663 thread that is executing blocks on I/O, then the whole process blocks.

5664 At the other end of the spectrum is the “kernel-thread model”. In this model, all threads are  
5665 visible to the operating system kernel. Thus, all threads are kernel scheduled entities, and all  
5666 threads can concurrently execute. The threads are scheduled onto processors by the kernel  
5667 according to the scheduling attributes of the threads. The drawback to this model is that the  
5668 creation and management of the threads entails operating system calls, as opposed to subroutine  
5669 calls, which makes kernel threads heavier weight than library threads.

5670 Hybrids of these two models are common. A hybrid model offers the speed of library threads  
5671 and the concurrency of kernel threads. In hybrid models, a process has some (relatively small)  
5672 number of kernel scheduled entities associated with it. It also has a potentially much larger  
5673 number of library threads associated with it. Some library threads may be bound to kernel  
5674 scheduled entities, while the other library threads are multiplexed onto the remaining kernel  
5675 scheduled entities. There are two levels of thread scheduling:

- 5676 1. The runtime library manages the scheduling of (unbound) library threads onto kernel  
5677 scheduled entities.
- 5678 2. The kernel manages the scheduling of kernel scheduled entities onto processors.

5679 For this reason, a hybrid model is referred to as a *two-level threads scheduling model*. In this model,  
5680 the process can have multiple concurrently executing threads; specifically, it can have as many  
5681 concurrently executing threads as it has kernel scheduled entities.

#### 5682 **Thread-Specific Data**

5683 Many applications require that a certain amount of context be maintained on a per-thread basis  
5684 across procedure calls. A common example is a multi-threaded library routine that allocates  
5685 resources from a common pool and maintains an active resource list for each thread. The  
5686 thread-specific data interface provided to meet these needs may be viewed as a two-dimensional  
5687 array of values with keys serving as the row index and thread IDs as the column index (although  
5688 the implementation need not work this way).

##### 5689 • Models

5690 Three possible thread-specific data models were considered:

- 5691 1. No Explicit Support

5692 A standard thread-specific data interface is not strictly necessary to support  
 5693 applications that require per-thread context. One could, for example, provide a hash  
 5694 function that converted a **pthread\_t** into an integer value that could then be used to  
 5695 index into a global array of per-thread data pointers. This hash function, in conjunction  
 5696 with *pthread\_self()*, would be all the interface required to support a mechanism of this  
 5697 sort. Unfortunately, this technique is cumbersome. It can lead to duplicated code as  
 5698 each set of cooperating modules implements their own per-thread data management  
 5699 schemes.

## 5700 2. Single (**void \***) Pointer

5701 Another technique would be to provide a single word of per-thread storage and a pair  
 5702 of functions to fetch and store the value of this word. The word could then hold a  
 5703 pointer to a block of per-thread memory. The allocation, partitioning, and general use  
 5704 of this memory would be entirely up to the application. Although this method is not as  
 5705 problematic as technique 1, it suffers from interoperability problems. For example, all  
 5706 modules using the per-thread pointer would have to agree on a common usage  
 5707 protocol.

## 5708 3. Key/Value Mechanism

5709 This method associates an opaque key (for example, stored in a variable of type  
 5710 **pthread\_key\_t**) with each per-thread datum. These keys play the role of identifiers for  
 5711 per-thread data. This technique is the most generic and avoids the problems noted  
 5712 above, albeit at the cost of some complexity.

5713 The primary advantage of the third model is its information hiding properties. Modules  
 5714 using this model are free to create and use their own key(s) independent of all other such  
 5715 usage, whereas the other models require that all modules that use thread-specific context  
 5716 explicitly cooperate with all other such modules. The data-independence provided by the  
 5717 third model is worth the additional interface.

### 5718 • Requirements

5719 It is important that it be possible to implement the thread-specific data interface without the  
 5720 use of thread private memory. To do otherwise would increase the weight of each thread,  
 5721 thereby limiting the range of applications for which the threads interfaces provided by  
 5722 IEEE Std. 1003.1-200x is appropriate.

5723 The values that one binds to the key via *pthread\_setspecific()* may, in fact, be pointers to  
 5724 shared storage locations available to all threads. It is only the key/value bindings that are  
 5725 maintained on a per-thread basis, and these can be kept in any portion of the address space  
 5726 that is reserved for use by the calling thread (for example, on the stack). Thus, no per-thread  
 5727 MMU state is required to implement the interface. On the other hand, there is nothing in the  
 5728 interface specification to preclude the use of a per-thread MMU state if it is available (for  
 5729 example, the key values returned by *pthread\_key\_create()* could be thread private memory  
 5730 addresses).

### 5731 • Standardization Issues

5732 Thread-specific data is a requirement for a usable thread interface. The binding described in  
 5733 this section provides a portable thread-specific data mechanism for languages that do not  
 5734 directly support a thread-specific storage class. A binding to IEEE Std. 1003.1-200x for a  
 5735 language that does include such a storage class need not provide this specific interface.

5736 If a language were to include the notion of thread-specific storage, it would be desirable (but  
 5737 *not* required) to provide an implementation of the pthreads thread-specific data interface  
 5738 based on the language feature. For example, assume that a compiler for a C-like language

5739 supports a *private* storage class that provides thread-specific storage. Something similar to  
 5740 the following macros might be used to effect a compatible implementation:

```
5741 #define pthread_key_t private void *
5742 #define pthread_key_create(key) /* no-op */
5743 #define pthread_setspecific(key,value) (key)=(value)
5744 #define pthread_getspecific(key) (key)
```

5745 **Note:** For the sake of clarity, this example ignores destructor functions. A correct  
 5746 implementation would have to support them.

## 5747 **Barriers**

### 5748 • Background

5749 Barriers are typically used in parallel DO/FOR loops to ensure that all threads have reached  
 5750 a particular stage in a parallel computation before allowing any to proceed to the next stage.  
 5751 Highly efficient implementation is possible on machines which support a “Fetch and Add”  
 5752 operation as described in the referenced Almasi and Gottlieb (1989).

5753 The use of return value PTHREAD\_BARRIER\_SERIAL\_THREAD is shown in the following  
 5754 example:

```
5755 if ((status=pthread_barrier_wait(&barrier)) ==
5756 PTHREAD_BARRIER_SERIAL_THREAD) {
5757 ...serial section
5758 }
5759 else if (status != 0) {
5760 ...error processing
5761 }
5762 status=pthread_barrier_wait(&barrier);
5763 ...
```

5764 This behavior allows a serial section of code to be executed by one thread as soon as all  
 5765 threads reach the first barrier. The second barrier prevents the other threads from proceeding  
 5766 until the serial section being executed by the one thread has completed.

5767 Although barriers can be implemented with mutexes and condition variables, the referenced  
 5768 Almasi and Gottlieb (1989) provides ample illustration that such implementations are  
 5769 significantly less efficient than is possible. While the relative efficiency of barriers may well  
 5770 vary by implementation, it is important that they be recognized in the IEEE Std. 1003.1-200x  
 5771 to facilitate application portability while providing the necessary freedom to implementors.

### 5772 • Lack of Timeout Feature

5773 Alternate versions of most blocking routines have been provided to support watchdog  
 5774 timeouts. No alternate interface of this sort has been provided for barrier waits for the  
 5775 following reasons:

- 5776 • Multiple threads may use different timeout values, some of which may be indefinite. It is  
 5777 not clear which threads should break through the barrier with a timeout error if and when  
 5778 these timeouts expire.
- 5779 • The barrier may become unusable once a thread breaks out of a *pthread\_barrier\_wait()*  
 5780 with a timeout error. There is, in general, no way to guarantee the consistency of a  
 5781 barrier's internal data structures once a thread has timed out of a *pthread\_barrier\_wait()*.  
 5782 Even the inclusion of a special barrier reinitialization function would not help much since  
 5783 it is not clear how this function would affect the behavior of threads that reach the barrier

5784                   between the original timeout and the call to the reinitialization function.

5785                   **Spin Locks**

5786                   • Background

5787                   Spin locks represent an extremely low-level synchronization mechanism suitable primarily  
5788                   for use on shared memory multi-processors. It is typically an atomically modified Boolean  
5789                   value that is set to one when the lock is held and to zero when the lock is freed.

5790                   When a caller requests a spin lock that is already held, it typically spins in a loop testing  
5791                   whether the lock has become available. Such spinning wastes processor cycles so the lock  
5792                   should only be held for short durations and not across sleep/block operations. Callers should  
5793                   unlock spin locks before calling sleep operations.

5794                   Spin locks are available on a variety of systems. The functions included in  
5795                   IEEE Std. 1003.1-200x are an attempt to standardize that existing practice.

5796                   • Lack of Timeout Feature

5797                   Alternate versions of most blocking routines have been provided to support watchdog  
5798                   timeouts. No alternate interface of this sort has been provided for spin locks for the following  
5799                   reasons:

5800                   • It is impossible to determine appropriate timeout intervals for spin locks in a portable  
5801                   manner. The amount of time one can expect to spend spin-waiting is inversely  
5802                   proportional to the degree of parallelism provided by the system.

5803                   It can vary from a few cycles when each competing thread is running on its own  
5804                   processor, to an indefinite amount of time when all threads are multiplexed on a single  
5805                   processor (which is why spin locking is not advisable on uniprocessors).

5806                   • When used properly, the amount of time the calling thread spends waiting on a spin lock  
5807                   should be considerably less than the time required to set up a corresponding watchdog  
5808                   timer. Since the primary purpose of spin locks is to provide a low-overhead  
5809                   synchronization mechanism for multi-processors, the overhead of a timeout mechanism  
5810                   was deemed unacceptable.

5811                   It was also suggested that an additional *count* argument be provided (on the  
5812                   *pthread\_spin\_lock()* call) in lieu of a true timeout so that a spin lock call could fail gracefully if  
5813                   it was unable to apply the lock after *count* attempts. This idea was rejected because it is not  
5814                   existing practice. Furthermore, the same effect can be obtained with *pthread\_spin\_trylock()*,  
5815                   as illustrated below:



```

5816 int n = MAX_SPIN;
5817 while (--n >= 0)
5818 {
5819 if (!pthread_spin_try_lock(...))
5820 break;
5821 }
5822 if (n >= 0)
5823 {
5824 /* Successfully acquired the lock */
5825 }
5826 else
5827 {
5828 /* Unable to acquire the lock */
5829 }

```

5830 • *process-shared* Attribute

5831 The initialization functions associated with most POSIX synchronization objects (for  
5832 example, mutexes, barriers, and read-write locks) take an attributes object with a *process-*  
5833 *shared* attribute that specifies whether or not the object is to be shared across processes. In the  
5834 draft corresponding to the first balloting round, two separate initialization functions are  
5835 provided for spin locks, however: one for spin locks that were to be shared across processes  
5836 (*spin\_init()*), and one for locks that were only used by multiple threads within a single  
5837 process (*pthread\_spin\_init()*). This was done so as to keep the overhead associated with spin  
5838 waiting to an absolute minimum. However, the balloting group requested that, since the  
5839 overhead associated to a bit check was small, spin locks should be consistent with the rest of  
5840 the synchronization primitives, and thus the *process-shared* attribute was introduced for spin  
5841 locks.

5842 • Spin Locks *versus* Mutexes

5843 It has been suggested that mutexes are an adequate synchronization mechanism and spin  
5844 locks are not necessary. Locking mechanisms typically must trade off the processor resources  
5845 consumed while setting up to block the thread and the processor resources consumed by the  
5846 thread while it is blocked. Spin locks require very little resources to set up the blocking of a  
5847 thread. Existing practice is to simply loop, repeating the atomic locking operation until the  
5848 lock is available. While the resources consumed to set up blocking of the thread are low, the  
5849 thread continues to consume processor resources while it is waiting.

5850 On the other hand, mutexes may be implemented such that the processor resources  
5851 consumed to block the thread are large relative to a spin lock. After detecting that the mutex  
5852 lock is not available, the thread must alter its scheduling state, add itself to a set of waiting  
5853 threads, and, when the lock becomes available again, undo all of this before taking over  
5854 ownership of the mutex. However, while a thread is blocked by a mutex, no processor  
5855 resources are consumed.

5856 Therefore, spin locks and mutexes may be implemented to have different characteristics.  
5857 Spin locks may have lower overall overhead for very short-term blocking, and mutexes may  
5858 have lower overall overhead when a thread will be blocked for longer periods of time. The  
5859 presence of both interfaces allows implementations with these two different characteristics,  
5860 both of which may be useful to a particular application.

5861 It has also been suggested that applications can build their own spin locks from the  
5862 *pthread\_mutex\_trylock()* function:

5863           while (pthread\_mutex\_trylock(&mutex));

5864           The apparent simplicity of this construct is somewhat deceiving, however. While the actual  
5865           wait is quite efficient, various guarantees on the integrity of mutex objects (for example,  
5866           priority inheritance rules) may add overhead to the successful path of the trylock operation  
5867           that is not required of spin locks. One could, of course, add an attribute to the mutex to  
5868           bypass such overhead, but the very act of finding and testing this attribute represents more  
5869           overhead than is found in the typical spin lock.

5870           The need to hold spin lock overhead to an absolute minimum also makes it impossible to  
5871           provide guarantees against starvation similar to those provided for mutexes or read-write  
5872           locks. The overhead required to implement such guarantees (for example, disabling  
5873           preemption before spinning) may well exceed the overhead of the spin wait itself by many  
5874           orders of magnitude. If a “safe” spin wait seems desirable, it can always be provided (albeit  
5875           at some performance cost) via appropriate mutex attributes.

### 5876           **XSI Supported Functions**

5877           On XSI-conformant systems, the following symbolic constants are always defined:

5878            \_POSIX\_READER\_WRITER\_LOCKS  
5879            \_POSIX\_THREAD\_ATTR\_STACKADDR  
5880            \_POSIX\_THREAD\_ATTR\_STACKSIZE  
5881            \_POSIX\_THREAD\_PROCESS\_SHARED  
5882            \_POSIX\_THREADS

5883           Therefore, the following threads functions are always supported:

|                                           |                                        |
|-------------------------------------------|----------------------------------------|
| 5884 <i>pthread_atfork()</i>              | <i>pthread_key_delete()</i>            |
| 5885 <i>pthread_attr_destroy()</i>        | <i>pthread_kill()</i>                  |
| 5886 <i>pthread_attr_getdetachstate()</i> | <i>pthread_mutex_destroy()</i>         |
| 5887 <i>pthread_attr_getguardsize()</i>   | <i>pthread_mutex_init()</i>            |
| 5888 <i>pthread_attr_getschedparam()</i>  | <i>pthread_mutex_lock()</i>            |
| 5889 <i>pthread_attr_getstackaddr()</i>   | <i>pthread_mutex_trylock()</i>         |
| 5890 <i>pthread_attr_getstacksize()</i>   | <i>pthread_mutex_unlock()</i>          |
| 5891 <i>pthread_attr_init()</i>           | <i>pthread_mutexattr_destroy()</i>     |
| 5892 <i>pthread_attr_setdetachstate()</i> | <i>pthread_mutexattr_getpshared()</i>  |
| 5893 <i>pthread_attr_setguardsize()</i>   | <i>pthread_mutexattr_gettype()</i>     |
| 5894 <i>pthread_attr_setschedparam()</i>  | <i>pthread_mutexattr_init()</i>        |
| 5895 <i>pthread_attr_setstackaddr()</i>   | <i>pthread_mutexattr_setpshared()</i>  |
| 5896 <i>pthread_attr_setstacksize()</i>   | <i>pthread_mutexattr_settype()</i>     |
| 5897 <i>pthread_cancel()</i>              | <i>pthread_once()</i>                  |
| 5898 <i>pthread_cleanup_pop()</i>         | <i>pthread_rwlock_destroy()</i>        |
| 5899 <i>pthread_cleanup_push()</i>        | <i>pthread_rwlock_init()</i>           |
| 5900 <i>pthread_cond_broadcast()</i>      | <i>pthread_rwlock_rdlock()</i>         |
| 5901 <i>pthread_cond_destroy()</i>        | <i>pthread_rwlock_tryrdlock()</i>      |
| 5902 <i>pthread_cond_init()</i>           | <i>pthread_rwlock_trywrlock()</i>      |
| 5903 <i>pthread_cond_signal()</i>         | <i>pthread_rwlock_unlock()</i>         |
| 5904 <i>pthread_cond_timedwait()</i>      | <i>pthread_rwlock_wrlock()</i>         |
| 5905 <i>pthread_cond_wait()</i>           | <i>pthread_rwlockattr_destroy()</i>    |
| 5906 <i>pthread_condattr_destroy()</i>    | <i>pthread_rwlockattr_getpshared()</i> |
| 5907 <i>pthread_condattr_getpshared()</i> | <i>pthread_rwlockattr_init()</i>       |
| 5908 <i>pthread_condattr_init()</i>       | <i>pthread_rwlockattr_setpshared()</i> |

|      |                                      |                                 |
|------|--------------------------------------|---------------------------------|
| 5909 | <i>pthread_condattr_setpshared()</i> | <i>pthread_self()</i>           |
| 5910 | <i>pthread_create()</i>              | <i>pthread_setcancelstate()</i> |
| 5911 | <i>pthread_detach()</i>              | <i>pthread_setcanceltype()</i>  |
| 5912 | <i>pthread_equal()</i>               | <i>pthread_setconcurrency()</i> |
| 5913 | <i>pthread_exit()</i>                | <i>pthread_setspecific()</i>    |
| 5914 | <i>pthread_getconcurrency()</i>      | <i>pthread_sigmask()</i>        |
| 5915 | <i>pthread_getspecific()</i>         | <i>pthread_testcancel()</i>     |
| 5916 | <i>pthread_join()</i>                | <i>sigwait()</i>                |
| 5917 | <i>pthread_key_create()</i>          |                                 |

5918 On XSI-conformant systems, the symbolic constant `_POSIX_THREAD_SAFE_FUNCTIONS` is  
5919 always defined. Therefore, the following functions are always supported:

|      |                           |                           |
|------|---------------------------|---------------------------|
| 5920 | <i>asctime_r()</i>        | <i>getpwnam_r()</i>       |
| 5921 | <i>ctime_r()</i>          | <i>getpwuid_r()</i>       |
| 5922 | <i>flockfile()</i>        | <i>gmtime_r()</i>         |
| 5923 | <i>ftrylockfile()</i>     | <i>localtime_r()</i>      |
| 5924 | <i>funlockfile()</i>      | <i>putc_unlocked()</i>    |
| 5925 | <i>getc_unlocked()</i>    | <i>putchar_unlocked()</i> |
| 5926 | <i>getchar_unlocked()</i> | <i>rand_r()</i>           |
| 5927 | <i>getgrgid_r()</i>       | <i>readdir_r()</i>        |
| 5928 | <i>getgrnam_r()</i>       | <i>strtok_r()</i>         |

5929 The following threads functions are only supported on XSI-conformant systems if the Realtime  
5930 Threads Option Group is supported :

|      |                                       |                                           |
|------|---------------------------------------|-------------------------------------------|
| 5931 | <i>pthread_attr_getinheritsched()</i> | <i>pthread_mutex_getprioceiling()</i>     |
| 5932 | <i>pthread_attr_getschedpolicy()</i>  | <i>pthread_mutex_setprioceiling()</i>     |
| 5933 | <i>pthread_attr_getscope()</i>        | <i>pthread_mutexattr_getprioceiling()</i> |
| 5934 | <i>pthread_attr_setinheritsched()</i> | <i>pthread_mutexattr_getprotocol()</i>    |
| 5935 | <i>pthread_attr_setschedpolicy()</i>  | <i>pthread_mutexattr_setprioceiling()</i> |
| 5936 | <i>pthread_attr_setscope()</i>        | <i>pthread_mutexattr_setprotocol()</i>    |
| 5937 | <i>pthread_getschedparam()</i>        | <i>pthread_setschedparam()</i>            |

### 5938 XSI Threads Extensions

5939 The following XSI extensions to POSIX.1c are now supported in IEEE Std. 1003.1-200x as part of  
5940 the alignment with the Single UNIX Specification:

- 5941 • Extended mutex attribute types
- 5942 • Read-write locks and attributes (also introduced by IEEE Std. 1003.1j-2000 amendment)
- 5943 • Thread concurrency level
- 5944 • Thread stack guard size
- 5945 • Parallel I/O

5946 A total of 19 new functions were added.

5947 These extensions carefully follow the threads programming model specified in POSIX.1c. As  
5948 with POSIX.1c, all the new functions return zero if successful; otherwise, an error number is  
5949 returned to indicate the error.

5950 The concept of attribute objects was introduced in POSIX.1c to allow implementations to extend  
 5951 IEEE Std. 1003.1-200x without changing the existing interfaces. Attribute objects were defined  
 5952 for threads, mutexes, and condition variables. Attributes objects are defined as implementation-  
 5953 defined opaque types to aid extensibility, and functions are defined to allow attributes to be set  
 5954 or retrieved. This model has been followed when adding the new type attribute of  
 5955 **pthread\_mutexattr\_t** or the new read-write lock attributes object **pthread\_rwlockattr\_t**.

5956 • Extended Mutex Attributes

5957 POSIX.1c defines a mutex attributes object as an implementation-defined opaque object of  
 5958 type **pthread\_mutexattr\_t**, and specifies a number of attributes which this object must have  
 5959 and a number of functions which manipulate these attributes. These attributes include  
 5960 *detachstate*, *inheritsched*, *schedparm*, *schedpolicy*, *contentionscope*, *stackaddr*, and *stacksize*.

5961 The System Interfaces volume of IEEE Std. 1003.1-200x specifies another mutex attribute  
 5962 called *type*. The *type* attribute allows applications to specify the behavior of mutex locking  
 5963 operations in situations where the POSIX.1c behavior is undefined. The OSF DCE threads  
 5964 implementation, based on Draft 4 of POSIX.1c, specified a similar attribute. Note that the  
 5965 names of the attributes have changed somewhat from the OSF DCE threads implementation.

5966 The System Interfaces volume of IEEE Std. 1003.1-200x also extends the specification of the  
 5967 following POSIX.1c functions which manipulate mutexes:

5968 *pthread\_mutex\_lock()*  
 5969 *pthread\_mutex\_trylock()*  
 5970 *pthread\_mutex\_unlock()*

5971 to take account of the new mutex attribute type and to specify behavior which was declared  
 5972 as undefined in POSIX.1c. How a calling thread acquires or releases a mutex now depends  
 5973 upon the mutex *type* attribute.

5974 The *type* attribute can have the following values:

5975 PTHREAD\_MUTEX\_NORMAL

5976 Basic mutex with no specific error checking built in. Does not report a deadlock error.

5977 PTHREAD\_MUTEX\_RECURSIVE

5978 Allows any thread to recursively lock a mutex. The mutex must be unlocked an equal  
 5979 number of times to release the mutex.

5980 PTHREAD\_MUTEX\_ERRORCHECK

5981 Detects and reports simple usage errors; that is, an attempt to unlock a mutex that is not  
 5982 locked by the calling thread or that is not locked at all, or an attempt to relock a mutex  
 5983 the thread already owns.

5984 PTHREAD\_MUTEX\_DEFAULT

5985 The default mutex type. May be mapped to any of the above mutex types or may be an  
 5986 implementation-defined type.

5987 *Normal* mutexes do not detect deadlock conditions; for example, a thread will hang if it tries  
 5988 to relock a normal mutex that it already owns. Attempting to unlock a mutex locked by  
 5989 another thread, or unlocking an unlocked mutex, results in undefined behavior. Normal  
 5990 mutexes will usually be the fastest type of mutex available on a platform but provide the  
 5991 least error checking.

5992 *Recursive* mutexes are useful for converting old code where it is difficult to establish clear  
 5993 boundaries of synchronization. A thread can relock a recursive mutex without first unlocking  
 5994 it. The relocking deadlock which can occur with normal mutexes cannot occur with this type  
 5995 of mutex. However, multiple locks of a recursive mutex require the same number of unlocks

5996 to release the mutex before another thread can acquire the mutex. Furthermore, this type of  
5997 mutex maintains the concept of an owner. Thus, a thread attempting to unlock a recursive  
5998 mutex which another thread has locked returns with an error. A thread attempting to unlock  
5999 a recursive mutex that is not locked shall return with an error. Never use a recursive mutex  
6000 with condition variables because the implicit unlock performed by `pthread_cond_wait()` or  
6001 `pthread_cond_timedwait()` will not actually release the mutex if it had been locked multiple  
6002 times.

6003 *Errorcheck* mutexes provide error checking and are useful primarily as a debugging aid. A  
6004 thread attempting to relock an errorcheck mutex without first unlocking it returns with an  
6005 error. Again, this type of mutex maintains the concept of an owner. Thus, a thread  
6006 attempting to unlock an errorcheck mutex which another thread has locked returns with an  
6007 error. A thread attempting to unlock an errorcheck mutex that is not locked also returns with  
6008 an error. It should be noted that errorcheck mutexes will almost always be much slower than  
6009 normal mutexes due to the extra state checks performed.

6010 The *default* mutex type provides implementation-defined error checking. The default mutex  
6011 may be mapped to one of the other defined types or may be something entirely different.  
6012 This enables each vendor to provide the mutex semantics which the vendor feels will be  
6013 most useful to their target users. Most vendors will probably choose to make normal  
6014 mutexes the default so as to give applications the benefit of the fastest type of mutexes  
6015 available on their platform. Check your implementation's documentation.

6016 An application developer can use any of the mutex types almost interchangeably as long as  
6017 the application does not depend upon the implementation detecting (or failing to detect) any  
6018 particular errors. Note that a recursive mutex can be used with condition variable waits as  
6019 long as the application never recursively locks the mutex.

6020 Two functions are provided for manipulating the *type* attribute of a mutex attributes object.  
6021 This attribute is set or returned in the *type* parameter of these functions. The  
6022 `pthread_mutexattr_settype()` function is used to set a specific type value while  
6023 `pthread_mutexattr_gettype()` is used to return the type of the mutex. Setting the *type* attribute  
6024 of a mutex attributes object affects only mutexes initialized using that mutex attributes  
6025 object. Changing the *type* attribute does not affect mutexes previously initialized using that  
6026 mutex attributes object.

#### 6027 • Read-Write Locks and Attributes

6028 The read-write locks introduced have been harmonized with those in IEEE Std. 1003.1j-2000;  
6029 see also Section B.2.9.6 (on page 3472).

6030 Read-write locks (also known as reader-writer locks) allow a thread to exclusively lock some  
6031 shared data while updating that data, or allow any number of threads to have simultaneous  
6032 read-only access to the data.

6033 Unlike a mutex, a read-write lock distinguishes between reading data and writing data. A  
6034 mutex excludes all other threads. A read-write lock allows other threads access to the data,  
6035 providing no thread is modifying the data. Thus, a read-write lock is less primitive than  
6036 either a mutex-condition variable pair or a semaphore.

6037 Application developers should consider using a read-write lock rather than a mutex to  
6038 protect data that is frequently referenced but seldom modified. Most threads (readers) will be  
6039 able to read the data without waiting and will only have to block when some other thread (a  
6040 writer) is in the process of modifying the data. Conversely a thread that wants to change the  
6041 data is forced to wait until there are no readers. This type of lock is often used to facilitate  
6042 parallel access to data on multi-processor platforms or to avoid context switches on single  
6043 processor platforms where multiple threads access the same data.

6044 If a read-write lock becomes unlocked and there are multiple threads waiting to acquire the  
6045 write lock, the implementation's scheduling policy determines which thread shall acquire the  
6046 read-write lock for writing. If there are multiple threads blocked on a read-write lock for both  
6047 read locks and write locks, it is unspecified whether the readers or a writer acquire the lock  
6048 first. However, for performance reasons, implementations often favor writers over readers to  
6049 avoid potential writer starvation.

6050 A read-write lock object is an implementation-defined opaque object of type  
6051 **pthread\_rwlock\_t** as defined in `<pthread.h>`. There are two different sorts of locks  
6052 associated with a read-write lock: a *read lock* and a *write lock*.

6053 The `pthread_rwlockattr_init()` function initializes a read-write lock attributes object with the  
6054 default value for all the attributes defined in the implementation. After a read-write lock  
6055 attributes object has been used to initialize one or more read-write locks, changes to the  
6056 read-write lock attributes object, including destruction, do not affect previously initialized  
6057 read-write locks.

6058 Implementations must provide at least the read-write lock attribute *process-shared*. This  
6059 attribute can have the following values:

6060 PTHREAD\_PROCESS\_SHARED

6061 Any thread of any process that has access to the memory where the read-write lock  
6062 resides can manipulate the read-write lock.

6063 PTHREAD\_PROCESS\_PRIVATE

6064 Only threads created within the same process as the thread that initialized the read-  
6065 write lock can manipulate the read-write lock. This is the default value.

6066 The `pthread_rwlockattr_setpshared()` function is used to set the *process-shared* attribute of an  
6067 initialized read-write lock attributes object while the function `pthread_rwlockattr_getpshared()`  
6068 obtains the current value of the *process-shared* attribute.

6069 A read-write lock attributes object is destroyed using the `pthread_rwlockattr_destroy()`  
6070 function. The effect of subsequent use of the read-write lock attributes object is undefined.

6071 A thread creates a read-write lock using the `pthread_rwlock_init()` function. The attributes of  
6072 the read-write lock can be specified by the application developer; otherwise, the default  
6073 implementation-defined read-write lock attributes are used if the pointer to the read-write  
6074 lock attributes object is NULL. In cases where the default attributes are appropriate, the  
6075 PTHREAD\_RWLOCK\_INITIALIZER macro can be used to initialize statically allocated  
6076 read-write locks.

6077 A thread which wants to apply a read lock to the read-write lock can use either  
6078 `pthread_rwlock_rdlock()` or `pthread_rwlock_tryrdlock()`. If `pthread_rwlock_rdlock()` is used, the  
6079 thread acquires a read lock if a writer does not hold the write lock and there are no writers  
6080 blocked on the write lock. If a read lock is not acquired, the calling thread blocks until it can  
6081 acquire a lock. However, if `pthread_rwlock_tryrdlock()` is used, the function returns  
6082 immediately with the error [EBUSY] if any thread holds a write lock or there are blocked  
6083 writers waiting for the write lock.

6084 A thread which wants to apply a write lock to the read-write lock can use either of two  
6085 functions: `pthread_rwlock_wrlock()` or `pthread_rwlock_trywrlock()`. If `pthread_rwlock_wrlock()`  
6086 is used, the thread acquires the write lock if no other reader or writer threads hold the read-  
6087 write lock. If the write lock is not acquired, the thread blocks until it can acquire the write  
6088 lock. However, if `pthread_rwlock_trywrlock()` is used, the function returns immediately with  
6089 the error [EBUSY] if any thread is holding either a read or a write lock.

6090 The `pthread_rwlock_unlock()` function is used to unlock a read-write lock object held by the  
6091 calling thread. Results are undefined if the read-write lock is not held by the calling thread. If  
6092 there are other read locks currently held on the read-write lock object, the read-write lock  
6093 object shall remain in the read locked state but without the current thread as one of its  
6094 owners. If this function releases the last read lock for this read-write lock object, the read-  
6095 write lock object shall be put in the unlocked read state. If this function is called to release a  
6096 write lock for this read-write lock object, the read-write lock object shall be put in the  
6097 unlocked state.

6098 • Thread Concurrency Level

6099 On threads implementations that multiplex user threads onto a smaller set of kernel  
6100 execution entities, the system attempts to create a reasonable number of kernel execution  
6101 entities for the application upon application startup.

6102 On some implementations, these kernel entities are retained by user threads that block in the  
6103 kernel. Other implementations do not *timeslice* user threads so that multiple compute-bound  
6104 user threads can share a kernel thread. On such implementations, some applications may use  
6105 up all the available kernel execution entities before its user-space threads are used up. The  
6106 process may be left with user threads capable of doing work for the application but with no  
6107 way to schedule them.

6108 The `pthread_setconcurrency()` function enables an application to request more kernel entities;  
6109 that is, specify a desired concurrency level. However, this function merely provides a hint to  
6110 the implementation. The implementation is free to ignore this request or to provide some  
6111 other number of kernel entities. If an implementation does not multiplex user threads onto a  
6112 smaller number of kernel execution entities, the `pthread_setconcurrency()` function has no  
6113 effect.

6114 The `pthread_setconcurrency()` function may also have an effect on implementations where the  
6115 kernel mode and user mode schedulers cooperate to ensure that ready user threads are not  
6116 prevented from running by other threads blocked in the kernel.

6117 The `pthread_getconcurrency()` function always returns the value set by a previous call to  
6118 `pthread_setconcurrency()`. However, if `pthread_setconcurrency()` was not previously called, this  
6119 function shall return zero to indicate that the threads implementation is maintaining the  
6120 concurrency level.

6121 • Thread Stack Guard Size

6122 DCE threads introduced the concept of a *thread stack guard size*. Most thread  
6123 implementations add a region of protected memory to a thread's stack, commonly known as  
6124 a *guard region*, as a safety measure to prevent stack pointer overflow in one thread from  
6125 corrupting the contents of another thread's stack. The default size of the guard regions  
6126 attribute is {PAGESIZE} bytes and is implementation-defined.

6127 Some application developers may wish to change the stack guard size. When an application  
6128 creates a large number of threads, the extra page allocated for each stack may strain system  
6129 resources. In addition to the extra page of memory, the kernel's memory manager has to keep  
6130 track of the different protections on adjoining pages. When this is a problem, the application  
6131 developer may request a guard size of 0 bytes to conserve system resources by eliminating  
6132 stack overflow protection.

6133 Conversely an application that allocates large data structures such as arrays on the stack may  
6134 wish to increase the default guard size in order to detect stack overflow. If a thread allocates  
6135 two pages for a data array, a single guard page provides little protection against thread stack  
6136 overflows since the thread can corrupt adjoining memory beyond the guard page.

6137 The System Interfaces volume of IEEE Std. 1003.1-200x defines a new attribute of a thread  
6138 attributes object; that is, the *guardsize* attribute which allows applications to specify the size  
6139 of the guard region of a thread's stack.

6140 Two functions are provided for manipulating a thread's stack guard size. The  
6141 *pthread\_attr\_setguardsize()* function sets the thread *guardsize* attribute, and the  
6142 *pthread\_attr\_getguardsize()* function retrieves the current value.

6143 An implementation may round up the requested guard size to a multiple of the configurable  
6144 system variable {PAGESIZE}. In this case, *pthread\_attr\_getguardsize()* returns the guard size  
6145 specified by the previous *pthread\_attr\_setguardsize()* function call and not the rounded up  
6146 value.

6147 If an application is managing its own thread stacks using the *stackaddr* attribute, the *guardsize*  
6148 attribute is ignored and no stack overflow protection is provided. In this case, it is the  
6149 responsibility of the application to manage stack overflow along with stack allocation.

#### 6150 • Parallel I/O

6151 Many I/O intensive applications, such as database engines, attempt to improve performance  
6152 through the use of parallel I/O. However, POSIX.1 does not support parallel I/O very well  
6153 because the current offset of a file is an attribute of the file descriptor.

6154 Suppose two or more threads independently issue read requests on the same file. To read  
6155 specific data from a file, a thread must first call *lseek()* to seek to the proper offset in the file,  
6156 and then call *read()* to retrieve the required data. If more than one thread does this at the  
6157 same time, the first thread may complete its seek call, but before it gets a chance to issue its  
6158 read call a second thread may complete its seek call, resulting in the first thread accessing  
6159 incorrect data when it issues its read call. One workaround is to lock the file descriptor while  
6160 seeking and reading or writing, but this reduces parallelism and adds overhead.

6161 Instead, the System Interfaces volume of IEEE Std. 1003.1-200x provides two functions to  
6162 make seek/read and seek/write operations atomic. The file descriptor's current offset is  
6163 unchanged, thus allowing multiple read and write operations to proceed in parallel. This  
6164 improves the I/O performance of threaded applications. The *pread()* function is used to do  
6165 an atomic read of data from a file into a buffer. Conversely, the *pwrite()* function does an  
6166 atomic write of data from a buffer to a file.

#### 6167 B.2.9.1 Thread-Safety

6168 All functions required by IEEE Std. 1003.1-200x need to be thread-safe. Implementations have to  
6169 provide internal synchronization when necessary in order to achieve this goal. In certain  
6170 cases—for example, most floating-point implementations—context switch code may have to  
6171 manage the writable shared state.

6172 It is not required that all functions provided by IEEE Std. 1003.1-200x be either async-cancel-safe  
6173 or async-signal-safe.

6174 As it turns out, some functions are inherently not thread-safe; that is, their interface  
6175 specifications preclude reentrancy. For example, some functions (such as *asctime()*) return a  
6176 pointer to a result stored in memory space allocated by the function on a per-process basis. Such  
6177 a function is not thread-safe, because its result can be overwritten by successive invocations.  
6178 Other functions, while not inherently non-thread-safe, may be implemented in ways that lead to  
6179 them not being thread-safe. For example, some functions (such as *rand()*) store state information  
6180 (such as a seed value, which survives multiple function invocations) in memory space allocated  
6181 by the function on a per-process basis. The implementation of such a function is not thread-safe  
6182 if the implementation fails to synchronize invocations of the function and thus fails to protect



6183 the state information. The problem is that when the state information is not protected,  
6184 concurrent invocations can interfere with one another (for example, see the same seed value).

#### 6185 *Thread-Safety and Locking of Existing Functions*

6186 Originally, POSIX.1 was not designed to work in a multi-threaded environment, and some  
6187 implementations of some existing functions will not work properly when executed concurrently.  
6188 To provide routines that will work correctly in an environment with threads (“thread-safe”), two  
6189 problems need to be solved:

- 6190 1. Routines that maintain or return pointers to static areas internal to the routine (which may  
6191 now be shared) need to be modified. The routines *ttyname()* and *localtime()* are examples.
- 6192 2. Routines that access data space shared by more than one thread need to be modified. The  
6193 *malloc()* function and the *stdio* family routines are examples.

6194 There are a variety of constraints on these changes. The first is compatibility with the existing  
6195 versions of these functions—non-thread-safe functions will continue to be in use for some time,  
6196 as the original interfaces are used by existing code. Another is that the new thread-safe versions  
6197 of these functions represent as small a change as possible over the familiar interfaces provided  
6198 by the existing non-thread-safe versions. The new interfaces should be independent of any  
6199 particular threads implementation. In particular, they should be thread-safe without depending  
6200 on explicit thread-specific memory. Finally, there should be minimal performance penalty due to  
6201 the changes made to the functions.

6202 It is intended that the list of functions from POSIX.1 that cannot be made thread-safe and for  
6203 which corrected versions are provided be complete.

#### 6204 *Thread-Safety and Locking Solutions*

6205 Many of the POSIX.1 functions were thread-safe and did not change at all. However, some  
6206 functions (for example, the math functions typically found in **libm**) are not thread-safe because  
6207 of writable shared global state. For instance, in IEEE Std. 754-1985 floating-point  
6208 implementations, the computation modes and flags are global and shared.

6209 Some functions are not thread-safe because a particular implementation is not reentrant,  
6210 typically because of a non-essential use of static storage. These require only a new  
6211 implementation.

6212 Thread-safe libraries are useful in a wide range of parallel (and asynchronous) programming  
6213 environments, not just within pthreads. In order to be used outside the context of pthreads,  
6214 however, such libraries still have to use some synchronization method. These could either be  
6215 independent of the pthread synchronization operations, or they could be a subset of the pthread  
6216 interfaces. Either method results in thread-safe library implementations that can be used without  
6217 the rest of pthreads.

6218 Some functions, such as the *stdio* family interface and dynamic memory allocation functions  
6219 such as *malloc()*, are interdependent routines that share resources (for example, buffers) across  
6220 related calls. These require synchronization to work correctly, but they do not require any  
6221 change to their external (user-visible) interfaces.

6222 In some cases, such as *getc()* and *putc()*, adding synchronization is likely to create an  
6223 unacceptable performance impact. In this case, slower thread-safe synchronized functions are to  
6224 be provided, but the original, faster (but unsafe) functions (which may be implemented as  
6225 macros) are retained under new names. Some additional special-purpose synchronization  
6226 facilities are necessary for these macros to be usable in multi-threaded programs. This also  
6227 requires changes in `<stdio.h>`.

6228 The other common reason that functions are unsafe is that they return a pointer to static storage,  
6229 making the functions non-thread-safe. This has to be changed, and there are three natural  
6230 choices:

6231 1. Return a pointer to thread-specific storage

6232 This could incur a severe performance penalty on those architectures with a costly  
6233 implementation of the thread-specific data interface.

6234 A variation on this technique is to use *malloc()* to allocate storage for the function output  
6235 and return a pointer to this storage. This technique may also have an undesirable  
6236 performance impact, however, and a simplistic implementation requires that the user  
6237 program explicitly free the storage object when it is no longer needed. This technique is  
6238 used by some existing POSIX.1 functions. With careful implementation for infrequently  
6239 used functions, there may be little or no performance or storage penalty, and the  
6240 maintenance of already-standardized interfaces is a significant benefit.

6241 2. Return the actual value computed by the function

6242 This technique can only be used with functions that return pointers to structures—routines  
6243 that return character strings would have to wrap their output in an enclosing structure in  
6244 order to return the output on the stack. There is also a negative performance impact  
6245 inherent in this solution in that the output value has to be copied twice before it can be  
6246 used by the calling function: once from the called routine's local buffers to the top of the  
6247 stack, then from the top of the stack to the assignment target. Finally, many older  
6248 compilers cannot support this technique due to a historical tendency to use internal static  
6249 buffers to deliver the results of structure-valued functions.

6250 3. Have the caller pass the address of a buffer to contain the computed value

6251 The only disadvantage of this approach is that extra arguments have to be provided by the  
6252 calling program. It represents the most efficient solution to the problem, however, and,  
6253 unlike the *malloc()* technique, it is semantically clear.

6254 There are some routines (often groups of related routines) whose interfaces are inherently non-  
6255 thread-safe because they communicate across multiple function invocations by means of static  
6256 memory locations. The solution is to redesign the calls so that they are thread-safe, typically by  
6257 passing the needed data as extra parameters. Unfortunately, this may require major changes to  
6258 the interface as well.

6259 A floating-point implementation using IEEE Std. 754-1985 is a case in point. A less problematic  
6260 example is the *rand48* family of pseudo-random number generators. The functions *getgrgid()*,  
6261 *getgrnam()*, *getpwnam()*, and *getpwuid()* are another such case.

6262 The problems with *errno* are discussed in **Alternative Solutions for Per-Thread *errno*** (on page  
6263 3390).

6264 Some functions can be thread-safe or not, depending on their arguments. These include the  
6265 *tmpnam()* and *ctermid()* functions. These functions have pointers to character strings as  
6266 arguments. If the pointers are not NULL, the functions store their results in the character string;  
6267 however, if the pointers are NULL, the functions store their results in an area that may be static  
6268 and thus subject to overwriting by successive calls. These should only be called by multi-thread  
6269 applications when their arguments are non-NULL.

6270 *Asynchronous Safety and Thread-Safety*

6271 A floating-point implementation has many modes that effect rounding and other aspects of  
6272 computation. Functions in some math library implementations may change the computation  
6273 modes for the duration of a function call. If such a function call is interrupted by a signal or

- 6274 cancelation, the floating-point state is not required to be protected.
- 6275 There is a significant cost to make floating-point operations async-cancel-safe or async-signal-  
6276 safe; accordingly, neither form of async safety is required.
- 6277 *Functions Returning Pointers to Static Storage*
- 6278 For those functions that are not thread-safe because they return values in fixed size statically  
6279 allocated structures, alternate “\_r” forms are provided that pass a pointer to an explicit result  
6280 structure. Those that return pointers into library-allocated buffers have forms provided with  
6281 explicit buffer and length parameters.
- 6282 For functions that return pointers to library-allocated buffers, it makes sense to provide “\_r”  
6283 versions that allow the application control over allocation of the storage in which results are  
6284 returned. This allows the state used by these functions to be managed on an application-specific  
6285 basis, supporting per-thread, per-process, or other application-specific sharing relationships.
- 6286 Early proposals had provided “\_r” versions for functions that returned pointers to variable-size  
6287 buffers without providing a means for determining the required buffer size. This would have  
6288 made using such functions exceedingly clumsy, potentially requiring iteratively calling them  
6289 with increasingly larger guesses for the amount of storage required. Hence, *sysconf()* variables  
6290 have been provided for such functions that return the maximum required buffer size.
- 6291 Thus, the rule that has been followed by IEEE Std. 1003.1-200x when adapting single-threaded  
6292 non-thread-safe library functions is as follows: all functions returning pointers to library-  
6293 allocated storage should have “\_r” versions provided, allowing the application control over the  
6294 storage allocation. Those with variable-sized return values accept both a buffer address and a  
6295 length parameter. The *sysconf()* variables are provided to supply the appropriate buffer sizes  
6296 when required. Implementors are encouraged to apply the same rule when adapting their own  
6297 existing functions to a pthreads environment.
- 6298 **B.2.9.2 Thread IDs**
- 6299 Separate programs should communicate through well-defined interfaces and should not depend  
6300 on each other's implementation. For example, if a programmer decides to rewrite the *sort*  
6301 program using multiple threads, it should be easy to do this so that the interface to the *sort*  
6302 program does not change. Consider that if the user causes SIGINT to be generated while the *sort*  
6303 program is running, keeping the same interface means that the entire sort program is killed, not  
6304 just one of its threads. As another example, consider a realtime program that manages a reactor.  
6305 Such a program may wish to allow other programs to control the priority at which it watches the  
6306 control rods. One technique to accomplish this is to write the ID of the thread watching the  
6307 control rods into a file and allow other programs to change the priority of that thread as they see  
6308 fit. A simpler technique is to have the reactor process accept IPCs (Inter-Process Communication  
6309 messages) from other processes, telling it at a semantic level what priority the program should  
6310 assign to watching the control rods. This allows the programmer greater flexibility in the  
6311 implementation. For example, the programmer can change the implementation from having one  
6312 thread per rod to having one thread watching all of the rods without changing the interface.  
6313 Having threads live inside the process means that the implementation of a process is invisible to  
6314 outside processes (excepting debuggers and system management tools).
- 6315 Threads do not provide a protection boundary. Every thread model allows threads to share  
6316 memory with other threads and encourages this sharing to be widespread. This means that one  
6317 thread can wipe out memory that is needed for the correct functioning of other threads that are  
6318 sharing its memory. Consequently, providing each thread with its own user and/or group IDs  
6319 would not provide a protection boundary between threads sharing memory.

6320 *B.2.9.3 Thread Mutexes*

6321 There is no additional rationale for this section.

6322 *B.2.9.4 Thread Scheduling*

## 6323 • Scheduling Implementation Models

6324 The following scheduling implementation models are presented in terms of threads and  
6325 “kernel entities”. This is to simplify exposition of the models, and it does not imply that an  
6326 implementation actually has an identifiable “kernel entity”.

6327 A kernel entity is not defined beyond the fact that it has scheduling attributes that are used to  
6328 resolve contention with other kernel entities for execution resources. A kernel entity may be  
6329 thought of as an envelope that holds a thread or a separate kernel thread. It is not a  
6330 conventional process, although it shares with the process the attribute that it has a single  
6331 thread of control; it does not necessarily imply an address space, open files, and so on. It is  
6332 better thought of as a primitive facility upon which conventional processes and threads may  
6333 be constructed.

## 6334 — System Thread Scheduling Model

6335 This model consists of one thread per kernel entity. The kernel entity is solely responsible  
6336 for scheduling thread execution on one or more processors. This model schedules all  
6337 threads against all other threads in the system using the scheduling attributes of the  
6338 thread.

## 6339 — Process Scheduling Model

6340 A generalized process scheduling model consists of two levels of scheduling. A threads  
6341 library creates a pool of kernel entities, as required, and schedules threads to run on them  
6342 using the scheduling attributes of the threads. Typically, the size of the pool is a function  
6343 of the simultaneously runnable threads, not the total number of threads. The kernel then  
6344 schedules the kernel entities onto processors according to their scheduling attributes,  
6345 which are managed by the threads library. This set model potentially allows a wide range  
6346 of mappings between threads and kernel entities.

## 6347 • System and Process Scheduling Model Performance

6348 There are a number of important implications on the performance of applications using these  
6349 scheduling models. The process scheduling model potentially provides lower overhead for  
6350 making scheduling decisions, since there is no need to access kernel-level information or  
6351 functions and the set of schedulable entities is smaller (only the threads within the process).

6352 On the other hand, since the kernel is also making scheduling decisions regarding the system  
6353 resources under its control (for example, CPU(s), I/O devices, memory), decisions that do  
6354 not take thread scheduling parameters into account can result in indeterminate delays for  
6355 realtime application threads, causing them to miss maximum response time limits.

## 6356 • Rate Monotonic Scheduling

6357 Rate monotonic scheduling was considered, but rejected for standardization in the context of  
6358 pthreads. A sporadic server policy is included.

## 6359 • Scheduling Options

6360 In IEEE Std. 1003.1-200x, the basic thread scheduling functions are defined under the Threads  
6361 option, so that they are required of all threads implementations. However, there are no  
6362 specific scheduling policies required by this option to allow for conforming thread  
6363 implementations that are not targeted to realtime applications.

6364 Specific standard scheduling policies are defined to be under the Thread Execution  
6365 Scheduling option, and they are specifically designed to support realtime applications by  
6366 providing predictable resource sharing sequences. The name of this option was chosen to  
6367 emphasize that this functionality is defined as appropriate for realtime applications that  
6368 require simple priority-based scheduling.

6369 It is recognized that these policies are not necessarily satisfactory for some multi-processor  
6370 implementations, and work is ongoing to address a wider range of scheduling behaviors. The  
6371 interfaces have been chosen to create abundant opportunity for future scheduling policies to  
6372 be implemented and standardized based on this interface. In order to standardize a new  
6373 scheduling policy, all that is required (from the standpoint of thread scheduling attributes) is  
6374 to define a new policy name, new members of the thread attributes object, and functions to  
6375 set these members when the scheduling policy is equal to the new value.

### 6376 **Scheduling Contention Scope**

6377 In order to accommodate the requirement for realtime response, each thread has a scheduling  
6378 contention scope attribute. Threads with a system scheduling contention scope have to be  
6379 scheduled with respect to all other threads in the system. These threads are usually bound to a  
6380 single kernel entity that reflects their scheduling attributes and are directly scheduled by the  
6381 kernel.

6382 Threads with a process scheduling contention scope need be scheduled only with respect to the  
6383 other threads in the process. These threads may be scheduled within the process onto a pool of  
6384 kernel entities. The implementation is also free to bind these threads directly to kernel entities  
6385 and let them be scheduled by the kernel. Process scheduling contention scope allows the  
6386 implementation the most flexibility and is the default if both contention scopes are supported  
6387 and none is specified.

6388 Thus, the choice by implementors to provide one or the other (or both) of these scheduling  
6389 models is driven by the need of their supported application domains for worst-case (that is,  
6390 realtime) response, or average-case (non-realtime) response.

### 6391 **Scheduling Allocation Domain**

6392 The SCHED\_FIFO and SCHED\_RR scheduling policies take on different characteristics on a  
6393 multi-processor. Other scheduling policies are also subject to changed behavior when executed  
6394 on a multi-processor. The concept of scheduling allocation domain determines the set of  
6395 processors on which the threads of an application may run. By considering the application's  
6396 processor scheduling allocation domain for its threads, scheduling policies can be defined in  
6397 terms of their behavior for varying processor scheduling allocation domain values. It is  
6398 conceivable that not all scheduling allocation domain sizes make sense for all scheduling  
6399 policies on all implementations. The concept of scheduling allocation domain, however, is a  
6400 useful tool for the description of multi-processor scheduling policies.

6401 The "process control" approach to scheduling obtains significant performance advantages from  
6402 dynamic scheduling allocation domain sizes when it is applicable.

6403 Non-Uniform Memory Access (NUMA) multi-processors may use a system scheduling structure  
6404 that involves reassignment of threads among scheduling allocation domains. In NUMA  
6405 machines, a natural model of scheduling is to match scheduling allocation domains to clusters of  
6406 processors. Load balancing in such an environment requires changing the scheduling allocation  
6407 domain to which a thread is assigned.

**6408 Scheduling Documentation**

6409 Implementation-provided scheduling policies need to be completely documented in order to be  
6410 useful. This documentation includes a description of the attributes required for the policy, the  
6411 scheduling interaction of threads running under this policy and all other supported policies, and  
6412 the effects of all possible values for processor scheduling allocation domain. Note that for the  
6413 implementor wishing to be minimally-compliant, it is (minimally) acceptable to define the  
6414 behavior as undefined.

**6415 Scheduling Contention Scope Attribute**

6416 The scheduling contention scope defines how threads compete for resources. Within  
6417 IEEE Std. 1003.1-200x, scheduling contention scope is used to describe only how threads are  
6418 scheduled in relation to one another in the system. That is, either they are scheduled against all  
6419 other threads in the system (“system scope”) or only against those threads in the process  
6420 (“process scope”). In fact, scheduling contention scope may apply to additional resources,  
6421 including virtual timers and profiling, which are not currently considered by  
6422 IEEE Std. 1003.1-200x.

**6423 Mixed Scopes**

6424 If only one scheduling contention scope is supported, the scheduling decision is straightforward.  
6425 To perform the processor scheduling decision in a mixed scope environment, it is necessary to  
6426 map the scheduling attributes of the thread with process-wide contention scope to the same  
6427 attribute space as the thread with system-wide contention scope.

6428 Since a conforming implementation has to support one and may support both scopes, it is useful  
6429 to discuss the effects of such choices with respect to example applications. If an implementation  
6430 supports both scopes, mixing scopes provides a means of better managing system-level (that is,  
6431 kernel-level) and library-level resources. In general, threads with system scope will require the  
6432 resources of a separate kernel entity in order to guarantee the scheduling semantics. On the  
6433 other hand, threads with process scope can share the resources of a kernel entity while  
6434 maintaining the scheduling semantics.

6435 The application is free to create threads with dedicated kernel resources, and other threads that  
6436 multiplex kernel resources. Consider the example of a window server. The server allocates two  
6437 threads per widget: one thread manages the widget user interface (including drawing), while the  
6438 other thread takes any required application action. This allows the widget to be “active” while  
6439 the application is computing. A screen image may be built from thousands of widgets. If each of  
6440 these threads had been created with system scope, then most of the kernel-level resources might  
6441 be wasted, since only a few widgets are active at any one time. In addition, mixed scope is  
6442 particularly useful in a window server where one thread with high priority and system scope  
6443 handles the mouse so that it tracks well. As another example, consider a database server. For  
6444 each of the hundreds or thousands of clients supported by a large server, an equivalent number  
6445 of threads will have to be created. If each of these threads were system, the consequences would  
6446 be the same as for the window server example above. However, the server could be constructed  
6447 so that actual retrieval of data is done by several dedicated threads. Dedicated threads that do  
6448 work for all clients frequently justify the added expense of system scope. If it were not  
6449 permissible to mix system and process threads in the same process, this type of solution would  
6450 not be possible.

**6451 Dynamic Thread Scheduling Parameters Access**

6452 In many time-constrained applications, there is no need to change the scheduling attributes  
6453 dynamically during thread or process execution, since the general use of these attributes is to  
6454 reflect directly the time constraints of the application. Since these time constraints are generally  
6455 imposed to meet higher-level system requirements, such as accuracy or availability, they  
6456 frequently should remain unchanged during application execution.

6457 However, there are important situations in which the scheduling attributes should be changed.  
6458 Generally, this will occur when external environmental conditions exist in which the time  
6459 constraints change. Consider, for example, a space vehicle major mode change, such as the  
6460 change from ascent to descent mode, or the change from the space environment to the  
6461 atmospheric environment. In such cases, the frequency with which many of the sensors or  
6462 acutators need to be read or written will change, which will necessitate a priority change. In  
6463 other cases, even the existence of a time constraint might be temporary, necessitating not just a  
6464 priority change, but also a policy change for ongoing threads or processes. For this reason, it is  
6465 critical that the interface should provide functions to change the scheduling parameters  
6466 dynamically, but, as with many of the other realtime functions, it is important that applications  
6467 use them properly to avoid the possibility of unnecessarily degrading performance.

6468 In providing functions for dynamically changing the scheduling behavior of threads, there were  
6469 two options: provide functions to get and set the individual scheduling parameters of threads, or  
6470 provide a single interface to get and set all the scheduling parameters for a given thread  
6471 simultaneously. Both approaches have merit. Access functions for individual parameters allow  
6472 simpler control of thread scheduling for simple thread scheduling parameters. However, a single  
6473 function for setting all the parameters for a given scheduling policy is required when first setting  
6474 that scheduling policy. Since the single all-encompassing functions are required, it was decided  
6475 to leave the interface as minimal as possible. Note that simpler functions (such as  
6476 *pthread\_setprio()* for threads running under the priority-based schedulers) can be easily defined  
6477 in terms of the all-encompassing functions.

6478 If the *pthread\_setschedparam()* function executes successfully, it will have set all of the scheduling  
6479 parameter values indicated in *param*; otherwise, none of the scheduling parameters will have  
6480 been modified. This is necessary to ensure that the scheduling of this and all other threads  
6481 continues to be consistent in the presence of an erroneous scheduling parameter.

6482 The [EPERM] error value is included in the list of possible *pthread\_setschedparam()* error returns  
6483 as a reflection of the fact that the ability to change scheduling parameters increases risks to the  
6484 implementation and application performance if the scheduling parameters are changed  
6485 improperly. For this reason, and based on some existing practice, it was felt that some  
6486 implementations would probably choose to define specific permissions for changing either a  
6487 thread's own or another thread's scheduling parameters. IEEE Std. 1003.1-200x does not include  
6488 portable methods for setting or retrieving permissions, so any such use of permissions is  
6489 completely unspecified .

**6490 Mutex Initialization Scheduling Attributes**

6491 In a priority-driven environment, a direct use of traditional primitives like mutexes and  
6492 condition variables can lead to unbounded priority inversion, where a higher priority thread can  
6493 be blocked by a lower priority thread, or set of threads, for an unbounded duration of time. As a  
6494 result, it becomes impossible to guarantee thread deadlines. Priority inversion can be bounded  
6495 and minimized by the use of priority inheritance protocols. This allows thread deadlines to be  
6496 guaranteed even in the presence of synchronization requirements.

6497 Two useful but simple members of the family of priority inheritance protocols are the basic  
6498 priority inheritance protocol and the priority ceiling protocol emulation. Under the Basic Priority

6499 Inheritance protocol (governed by the Threads Priority Inheritance option), a thread that is  
6500 blocking higher priority threads executes at the priority of the highest priority thread that it  
6501 blocks. This simple mechanism allows priority inversion to be bounded by the duration of  
6502 critical sections and makes timing analysis possible.

6503 Under the Priority Ceiling Protocol Emulation protocol (governed by the Thread Priority  
6504 Protection option), each mutex has a priority ceiling, usually defined as the priority of the  
6505 highest priority thread that can lock the mutex. When a thread is executing inside critical  
6506 sections, its priority is unconditionally increased to the highest of the priority ceilings of all the  
6507 mutexes owned by the thread. This protocol has two very desirable properties in uni-processor  
6508 systems. First, a thread can be blocked by a lower priority thread for at most the duration of one  
6509 single critical section. Furthermore, when the protocol is correctly used in a single processor, and  
6510 if threads do not become blocked while owning mutexes, mutual deadlocks are prevented.

6511 The priority ceiling emulation can be extended to multiple processor environments, in which  
6512 case the values of the priority ceilings will be assigned depending on the kind of mutex that is  
6513 being used: local to only one processor, or global, shared by several processors. Local priority  
6514 ceilings will be assigned the usual way, equal to the priority of the highest priority thread that  
6515 may lock that mutex. Global priority ceilings will usually be assigned a priority level higher than  
6516 all the priorities assigned to any of the threads that reside in the involved processors to avoid the  
6517 effect called remote blocking.

#### 6518 **Change the Priority Ceiling of a Mutex**

6519 In order for the priority protect protocol to exhibit its desired properties of bounding priority  
6520 inversion and avoidance of deadlock, it is critical that the ceiling priority of a mutex be the same  
6521 as the priority of the highest thread that can ever hold it, or higher. Thus, if the priorities of the  
6522 threads using such mutexes never change dynamically, there is no need ever to change the  
6523 priority ceiling of a mutex.

6524 However, if a major system mode change results in an altered response time requirement for one  
6525 or more application threads, their priority has to change to reflect it. It will occasionally be the  
6526 case that the priority ceilings of mutexes held also need to change. While changing priority  
6527 ceilings should generally be avoided, it is important that IEEE Std. 1003.1-200x provide these  
6528 interfaces for those cases in which it is necessary.

#### 6529 *B.2.9.5 Thread Cancellation*

6530 Many existing threads packages have facilities for canceling an operation or canceling a thread.  
6531 These facilities are used for implementing user requests (such as the CANCEL button in a  
6532 window-based application), for implementing OR parallelism (for example, telling the other  
6533 threads to stop working once one thread has found a forced mate in a parallel chess program), or  
6534 for implementing the ABORT mechanism in Ada.

6535 POSIX programs traditionally have used the signal mechanism combined with either *longjmp()*  
6536 or polling to cancel operations. Many POSIX programmers have trouble using these facilities to  
6537 solve their problems efficiently in a single-threaded process. With the introduction of threads,  
6538 these solutions become even more difficult to use.

6539 The main issues with implementing a cancellation facility are specifying the operation to be  
6540 canceled, cleanly releasing any resources allocated to that operation, controlling when the target  
6541 notices that it has been canceled, and defining the interaction between asynchronous signals and  
6542 cancellation.



**6543 Specifying the Operation to Cancel**

6544 Consider a thread that calls through five distinct levels of program abstraction and then, inside  
6545 the lowest-level abstraction, calls a function that suspends the thread. (An abstraction boundary  
6546 is a layer at which the client of the abstraction sees only the service being provided and can  
6547 remain ignorant of the implementation. Abstractions are often layered, each level of abstraction  
6548 being a client of the lower-level abstraction and implementing a higher-level abstraction.)  
6549 Depending on the semantics of each abstraction, one could imagine wanting to cancel only the  
6550 call that causes suspension, only the bottom two levels, or the operation being done by the entire  
6551 thread. Canceling operations at a finer grain than the entire thread is difficult because threads  
6552 are active and they may be run in parallel on a multi-processor. By the time one thread can make  
6553 a request to cancel an operation, the thread performing the operation may have completed that  
6554 operation and gone on to start another operation whose cancelation is not desired. Thread IDs  
6555 are not reused until the thread has exited, and either it was created with the *Attr detachstate*  
6556 attribute set to *PTHREAD\_CREATE\_DETACHED* or the *pthread\_join()* or *pthread\_detach()*  
6557 function has been called for that thread. Consequently, a thread cancelation will never be  
6558 misdirected when the thread terminates. For these reasons, the canceling of operations is done at  
6559 the granularity of the thread. Threads are designed to be inexpensive enough so that a separate  
6560 thread may be created to perform each separately cancelable operation; for example, each  
6561 possibly long running user request.

6562 For cancelation to be used in existing code, cancelation scopes and handlers will have to be  
6563 established for code that needs to release resources upon cancelation, so that it follows the  
6564 programming discipline described in the text.

**6565 A Special Signal Versus a Special Interface**

6566 Two different mechanisms were considered for providing the cancelation interfaces. The first  
6567 was to provide an interface to direct signals at a thread and then to define a special signal that  
6568 had the required semantics. The other alternative was to use a special interface that delivered the  
6569 correct semantics to the target thread.

6570 The solution using signals produced a number of problems. It required the implementation to  
6571 provide cancelation in terms of signals whereas a perfectly valid (and possibly more efficient)  
6572 implementation could have both layered on a low-level set of primitives. There were so many  
6573 exceptions to the special signal (it cannot be used with kill, no POSIX.1 interfaces can be used  
6574 with it) that it was clearly not a valid signal. Its semantics on delivery were also completely  
6575 different from any existing POSIX.1 signal. As such, a special interface that did not mandate the  
6576 implementation and did not confuse the semantics of signals and cancelation was felt to be the  
6577 better solution.

**6578 Races Between Cancelation and Resuming Execution**

6579 Due to the nature of cancelation, there is generally no synchronization between the thread  
6580 requesting the cancelation of a blocked thread and events that may cause that thread to resume  
6581 execution. For this reason, and because excess serialization hurts performance, when both an  
6582 event that a thread is waiting for has occurred and a cancelation request has been made and  
6583 cancelation is enabled, IEEE Std. 1003.1-200x explicitly allows the implementation to choose  
6584 between returning from the blocking call or acting on the cancelation request.

6585 **Interaction of Cancellation with Asynchronous Signals**

6586 A typical use of cancellation is to acquire a lock on some resource and to establish a cancellation  
6587 cleanup handler for releasing the resource when and if the thread is canceled.

6588 A correct and complete implementation of cancellation in the presence of asynchronous signals  
6589 requires considerable care. An implementation has to push a cancellation cleanup handler on the  
6590 cancellation cleanup stack while maintaining the integrity of the stack data structure. If an  
6591 asynchronously generated signal is posted to the thread during a stack operation, the signal  
6592 handler cannot manipulate the cancellation cleanup stack. As a consequence, asynchronous  
6593 signal handlers may not cancel threads or otherwise manipulate the cancellation state of a thread.  
6594 Threads may, of course, be canceled by another thread that used a *sigwait()* function to wait  
6595 synchronously for an asynchronous signal.

6596 In order for cancellation to function correctly, it is required that asynchronous signal handlers not  
6597 change the cancellation state. This requires that some elements of existing practice, such as using  
6598 *longjmp()* to exit from an asynchronous signal handler implicitly, be prohibited in cases where  
6599 the integrity of the cancellation state of the interrupt thread cannot be ensured.

6600 **Thread Cancellation Overview**

## 6601 • Cancellability States

6602 The three possible cancellability states (disabled, deferred, and asynchronous) are encoded  
6603 into two separate bits ((disable, enable) and (deferred, asynchronous)) to allow them to be  
6604 changed and restored independently. For instance, short code sequences that will not block  
6605 sometimes disable cancellability on entry and restore the previous state upon exit. Likewise,  
6606 long or unbounded code sequences containing no convenient explicit cancellation points will  
6607 sometimes set the cancellability type to asynchronous on entry and restore the previous value  
6608 upon exit.

## 6609 • Cancellation Points

6610 Cancellation points are points inside of certain functions where a thread has to act on any  
6611 pending cancellation request when cancellability is enabled, if the function would block. As  
6612 with checking for signals, operations need only check for pending cancellation requests when  
6613 the operation is about to block indefinitely.

6614 The idea was considered of allowing implementations to define whether blocking calls such  
6615 as *read()* should be cancellation points. It was decided that it would adversely affect the  
6616 design of portable applications if blocking calls were not cancellation points because threads  
6617 could be left blocked in an uncancelable state.

6618 There are several important blocking routines that are specifically not made cancellation  
6619 points:

6620 — *pthread\_mutex\_lock()*

6621 If *pthread\_mutex\_lock()* were a cancellation point, every routine that called it would also  
6622 become a cancellation point (that is, any routine that touched shared state would  
6623 automatically become a cancellation point). For example, *malloc()*, *free()*, and *rand()*  
6624 would become cancellation points under this scheme. Having too many cancellation points  
6625 makes programming very difficult, leading to either much disabling and restoring of  
6626 cancellability or much difficulty in trying to arrange for reliable cleanup at every possible  
6627 place.

6628 Since *pthread\_mutex\_lock()* is not a cancellation point, threads could result in being  
6629 blocked uninterruptibly for long periods of time if mutexes were used as a general

6630 synchronization mechanism. As this is normally not acceptable, mutexes should only be  
6631 used to protect resources that are held for small fixed lengths of time where not being  
6632 able to be canceled will not be a problem. Resources that need to be held exclusively for  
6633 long periods of time should be protected with condition variables.

6634 — *barrier\_wait()*

6635 Canceling a barrier wait will render a barrier unusable. Similar to a barrier timeout (which  
6636 the standard developers rejected), there is no way to guarantee the consistency of a  
6637 barrier's internal data structures if a barrier wait is canceled.

6638 — *pthread\_spin\_lock()*

6639 As with mutexes, spin locks should only be used to protect resources that are held for  
6640 small fixed lengths of time where not being cancelable will not be a problem.

6641 Every library routine should specify whether or not it includes any cancellation points.  
6642 Typically, only those routines that may block or compute indefinitely need to include  
6643 cancellation points.

6644 Correctly coded routines only reach cancellation points after having set up a cancellation  
6645 cleanup handler to restore invariants if the thread is canceled at that point. Being cancelable  
6646 only at specified cancellation points allows programmers to keep track of actions needed in a  
6647 cancellation cleanup handler more easily. A thread should only be made asynchronously  
6648 cancelable when it is not in the process of acquiring or releasing resources or otherwise in a  
6649 state from which it would be difficult or impossible to recover.

#### 6650 • Thread Cancellation Cleanup Handlers

6651 The cancellation cleanup handlers provide a portable mechanism, easy to implement, for  
6652 releasing resources and restoring invariants. They are easier to use than signal handlers  
6653 because they provide a stack of cancellation cleanup handlers rather than a single handler,  
6654 and because they have an argument that can be used to pass context information to the  
6655 handler.

6656 The alternative to providing these simple cancellation cleanup handlers (whose only use is for  
6657 cleaning up when a thread is canceled) is to define a general exception package that could be  
6658 used for handling and cleaning up after hardware traps and software detected errors. This  
6659 was too far removed from the charter of providing threads to handle asynchrony. However,  
6660 it is an explicit goal of IEEE Std. 1003.1-200x to be compatible with existing exception  
6661 facilities and languages having exceptions.

6662 The interaction of this facility and other procedure-based or language-level exception  
6663 facilities is unspecified in this version of IEEE Std. 1003.1-200x. However, it is intended that it  
6664 be possible for an implementation to define the relationship between these cancellation  
6665 cleanup handlers and Ada, C++, or other language-level exception handling facilities.

6666 It was suggested that the cancellation cleanup handlers should also be called when the  
6667 process exits or calls the *exec* function. This was rejected partly due to the performance  
6668 problem caused by having to call the cancellation cleanup handlers of every thread before the  
6669 operation could continue. The other reason was that the only state expected to be cleaned up  
6670 by the cancellation cleanup handlers would be the intraprocess state. Any handlers that are to  
6671 clean up the interprocess state would be registered with *atexit()*. There is the orthogonal  
6672 problem that the *exec* functions do not honor the *atexit()* handlers, but resolving this is  
6673 beyond the scope of IEEE Std. 1003.1-200x.

#### 6674 • Async-Cancel Safety

6675 A function is said to be *async-cancel safe* if it is written in such a way that entering the function  
 6676 with asynchronous cancelability enabled will not cause any invariants to be violated, even if  
 6677 a cancelation request is delivered at any arbitrary instruction. Functions that are *async-*  
 6678 *cancel-safe* are often written in such a way that they need to acquire no resources for their  
 6679 operation and the visible variables that they may write are strictly limited.

6680 Any routine that gets a resource as a side-effect cannot be made *async-cancel-safe* (for  
 6681 example, *malloc()*). If such a routine were called with asynchronous cancelability enabled, it  
 6682 might acquire the resource successfully, but as it was returning to the client, it could act on a  
 6683 cancelation request. In such a case, the application would have no way of knowing whether  
 6684 the resource was acquired or not.

6685 Indeed, because many interesting routines cannot be made *async-cancel-safe*, most library  
 6686 routines in general are not *async-cancel-safe*. Every library routine should specify whether or  
 6687 not it is *async-cancel safe* so that programmers know which routines can be called from code  
 6688 that is asynchronously cancelable.

### 6689 B.2.9.6 Thread Read-Write Locks

#### 6690 Background

6691 Read-write locks are often used to allow parallel access to data on multi-processors, to avoid  
 6692 context switches on uni-processors when multiple threads access the same data, and to protect  
 6693 data structures that are frequently accessed (that is, read) but rarely updated (that is, written).  
 6694 The in-core representation of a file system directory is a good example of such a data structure.  
 6695 One would like to achieve as much concurrency as possible when searching directories, but limit  
 6696 concurrent access when adding or deleting files.

6697 Although read-write locks can be implemented with mutexes and condition variables, such  
 6698 implementations are significantly less efficient than is possible. Therefore, this synchronization  
 6699 primitive is included in IEEE Std. 1003.1-200x for the purpose of allowing more efficient  
 6700 implementations in multi-processor systems.

#### 6701 Queuing of Waiting Threads

6702 The *pthread\_rwlock\_unlock()* function description states that one writer or one or more readers  
 6703 shall acquire the lock if it is no longer held by any thread as a result of the call. However, the  
 6704 function does not specify which thread(s) acquire the lock, unless the Thread Execution  
 6705 Scheduling option is supported.

6706 The standard developers considered the issue of scheduling with respect to the queuing of  
 6707 threads blocked on a read-write lock. The question turned out to be whether  
 6708 IEEE Std. 1003.1-200x should require priority scheduling of read-write locks for threads whose  
 6709 execution scheduling policy is priority-based (for example, *SCHED\_FIFO* or *SCHED\_RR*). There  
 6710 are tradeoffs between priority scheduling, the amount of concurrency achievable among readers,  
 6711 and the prevention of writer and/or reader starvation.

6712 For example, suppose one or more readers hold a read-write lock and the following threads  
 6713 request the lock in the listed order:

```
6714 pthread_rwlock_wrlock() - Low priority thread writer_a
6715 pthread_rwlock_rdlock() - High priority thread reader_a
6716 pthread_rwlock_rdlock() - High priority thread reader_b
6717 pthread_rwlock_rdlock() - High priority thread reader_c
```

6718 When the lock becomes available, should *writer\_a* block the high priority readers? Or, suppose a  
6719 read-write lock becomes available and the following are queued:

6720 pthread\_rwlock\_rdlock() - Low priority thread reader\_a  
6721 pthread\_rwlock\_rdlock() - Low priority thread reader\_b  
6722 pthread\_rwlock\_rdlock() - Low priority thread reader\_c  
6723 pthread\_rwlock\_wrlock() - Medium priority thread writer\_a  
6724 pthread\_rwlock\_rdlock() - High priority thread reader\_d

6725 If priority scheduling is applied then *reader\_d* would acquire the lock and *writer\_a* would block  
6726 the remaining readers. But should the remaining readers also acquire the lock to increase  
6727 concurrency? The solution adopted takes into account that when the Thread Execution  
6728 Scheduling option is supported, high priority threads may in fact starve low priority threads (the  
6729 application developer is responsible in this case to design the system in such a way that this  
6730 starvation is avoided). Therefore, IEEE Std. 1003.1-200x specifies that high priority readers take  
6731 precedence over lower priority writers. However, to prevent writer starvation from threads of  
6732 the same or lower priority, writers take precedence over readers of the same or lower priority.

6733 Priority inheritance mechanisms are non-trivial in the context of read-write locks. When a high  
6734 priority writer is forced to wait for multiple readers, for example, it is not clear which subset of  
6735 the readers should inherit the writer's priority. Furthermore, the internal data structures that  
6736 record the inheritance must be accessible to all readers, and this implies some sort of  
6737 serialization that could negate any gain in parallelism achieved through the use of multiple  
6738 readers in the first place. Finally, existing practice does not support the use of priority  
6739 inheritance for read-write locks. Therefore, no specification of priority inheritance or priority  
6740 ceiling is attempted. If reliable priority-scheduled synchronization is absolutely required, it can  
6741 always be obtained through the use of mutexes.

#### 6742 **Comparison to *fcntl()* Locks**

6743 The read-write locks and the *fcntl()* locks in IEEE Std. 1003.1-200x share a common goal:  
6744 increasing concurrency among readers, thus increasing throughput and decreasing delay.

6745 However, the read-write locks have two features not present in the *fcntl()* locks. First, under  
6746 priority scheduling, read-write locks are granted in priority order. Second, also under priority  
6747 scheduling, writer starvation is prevented by giving writers preference over readers of equal or  
6748 lower priority.

6749 Also, read-write locks can be used in systems lacking a file system, such as those conforming to  
6750 the minimal realtime system profile of IEEE Std. 1003.13-1998.

#### 6751 **History of Resolution Issues**

6752 Based upon some balloting objections, the draft specified the behavior of threads waiting on a  
6753 read-write lock during the execution of a signal handler, as if the thread had not called the lock  
6754 operation. However, this specified behavior would require implementations to establish  
6755 internal signal handlers even though this situation would be rare, or never happen for many  
6756 programs. This would introduce an unacceptable performance hit in comparison to the little  
6757 additional functionality gained. Therefore, the behavior of read-write locks and signals was  
6758 reverted back to its previous mutex-like specification.

6759 *B.2.9.7 Thread Interactions with Regular File Operations*

6760 There is no additional rationale for this section.

6761 **B.2.10 Sockets**

6762 There is no additional rationale for this section.

6763 *B.2.10.1 Protocol Families*

6764 There is no additional rationale for this section.

6765 *B.2.10.2 Protocols*

6766 There is no additional rationale for this section.

6767 *B.2.10.3 Addressing*

6768 There is no additional rationale for this section.

6769 *B.2.10.4 Routing*

6770 There is no additional rationale for this section.

6771 *B.2.10.5 Interfaces*

6772 There is no additional rationale for this section.

6773 *B.2.10.6 Socket Types*

6774 There is no additional rationale for this section.

6775 *B.2.10.7 Socket I/O Mode*

6776 There is no additional rationale for this section.

6777 *B.2.10.8 Socket Owner*

6778 There is no additional rationale for this section.

6779 *B.2.10.9 Socket Queue Limits*

6780 There is no additional rationale for this section.

6781 *B.2.10.10 Pending Error*

6782 There is no additional rationale for this section.

6783 *B.2.10.11 Socket Receive Queue*

6784 There is no additional rationale for this section.

6785 *B.2.10.12 Socket Out-of-Band Data State*

6786 There is no additional rationale for this section.

6787 *B.2.10.13 Connection Indication Queue*

6788           There is no additional rationale for this section.

6789 *B.2.10.14 Signals*

6790           There is no additional rationale for this section.

6791 *B.2.10.15 Asynchronous Errors*

6792           There is no additional rationale for this section.

6793 *B.2.10.16 Use of Options*

6794           There is no additional rationale for this section.

6795 *B.2.10.17 Use of Sockets for Local UNIX Connections*

6796           There is no additional rationale for this section.

6797 *B.2.10.18 Use of Sockets over Internet Protocols Based on IPv4*

6798           There is no additional rationale for this section.

6799 *B.2.10.19 Use of Sockets over Internet Protocols Based on IPv6*

6800           There is no additional rationale for this section.

6801 **B.2.11 Tracing**

6802           The organization of the tracing rationale differs from the traditional rationale in that this tracing  
6803           rationale text is written against the trace interface as a whole, rather than against the individual  
6804           components of the trace interface or the normative section in which those components are  
6805           defined. Therefore the sections below do not parallel the sections of normative text in  
6806           IEEE Std. 1003.1-200x.

6807 *B.2.11.1 Objectives*

6808           The intended uses of tracing are application-system debugging during system development, as a  
6809           “flight recorder” for maintenance of fielded systems, and as a performance measurement tool. In  
6810           all of these intended uses, the vendor-supplied computer system and its software are, for this  
6811           discussion, assumed error-free; the intent being to debug the user-written and/or third-party  
6812           application code, and their interactions. Clearly, problems with the vendor-supplied system and  
6813           its software will be uncovered from time to time, but this is a byproduct of the primary activity,  
6814           debugging user code.

6815           Another need for defining a trace interface in POSIX stems from the objective to provide an  
6816           efficient portable way to perform benchmarks. Existing practice shows that such interfaces are  
6817           commonly used in a variety of systems but with little commonality. As part of the benchmarking  
6818           needs, we must consider two aspects within the trace interface.

6819           The first, and perhaps more important one, is the qualitative aspect.

6820           The second is the quantitative aspect.

## 6821           • Qualitative Aspect

6822           To better understand this aspect, let us consider an example. Suppose that you want to  
6823           organize a number of actions to be performed during the day. Some of these actions are

6824 known at the beginning of the day. Some others, which may be more or less important, will  
 6825 be triggered by reading your mail. During the day you will make some phone calls and  
 6826 synchronously receive some more information. Finally you will receive asynchronous phone  
 6827 calls that also will trigger actions. If you, or somebody else, examines your day at work, you,  
 6828 or he, can discover that you have not efficiently organized your work. For instance, relative  
 6829 to the phone calls you made, would it be preferable to make some of these early in the  
 6830 morning? Or to delay some others until the end of the day? Relative to the phone calls you  
 6831 have received, you might find that somebody you called in the morning has called you 10  
 6832 times while you were performing some important work. To examine, afterwards, your day at  
 6833 work, you record in sequence all the trace events relative to your work. This should give you  
 6834 a chance of organizing your next day at work.

6835 This is the qualitative aspect of the trace interface. The user of a system needs to keep a trace  
 6836 of particular points the application passes through, so that he can eventually make some  
 6837 changes in the application and/or system configuration, to give the application a chance of  
 6838 running more efficiently.

6839 • Quantitative Aspect

6840 This aspect concerns primarily realtime applications, where missed deadlines can be  
 6841 undesirable. Although there are, in POSIX.1b and POSIX.1c/POSIX.1d/POSIX.1j, some  
 6842 interfaces useful for such applications (timeouts, execution time monitoring, and so on),  
 6843 there are no APIs to aid in the tuning of a realtime application's behavior (**timespec** in  
 6844 timeouts, length of message queues, duration of driver interrupt service routine, and so on).  
 6845 The tuning of an application needs a means of recording timestamped important trace events  
 6846 during execution in order to analyze offline, and eventually, to tune some realtime features  
 6847 (redesign the system with less functionalities, readjust timeouts, redesign driver interrupts,  
 6848 and so on).

6849 **Detailed Objectives**

6850 Objectives were defined to build the trace interface and are kept for historical interest. Although  
 6851 some objectives are not fully respected in this trace interface, the concept of the POSIX trace  
 6852 interface assumes the following points:

- 6853 1. It shall be possible to trace both system and user trace events concurrently.
- 6854 2. It must be possible to trace per-process trace events and also to trace system trace events  
 6855 which are unrelated to any particular process. A per-process trace event is either user-  
 6856 initiated or system-initiated.
- 6857 3. It must be possible to control tracing on a per process basis from either inside or outside  
 6858 the process.
- 6859 4. It must be possible to control tracing on a per-thread basis from inside the enclosing  
 6860 process.
- 6861 5. Trace points shall be controllable by trace event type ID from inside and outside of the  
 6862 process. Multiple trace points can have the same trace event type ID, and will be controlled  
 6863 jointly.
- 6864 6. Recording of trace events is dependent on both trace event type ID and the  
 6865 process/thread. Both must be enabled in order to record trace events. System trace events  
 6866 may or may not be handled differently.
- 6867 7. The API shall not mandate the ability to control tracing for more than one process at the  
 6868 same time.



- 6869 8. There is no objective for trace control on anything bigger than a process; for example,  
6870 group or session.
- 6871 9. Trace propagation and control:
- 6872 a. Trace propagation across fork is optional; the default is to not trace a child process.
- 6873 b. Trace control shall span *thread\_create* operations; that is, if a process is being traced,  
6874 any thread will be traced as well if this thread allows tracing. The default is to allow  
6875 tracing.
- 6876 10. Trace control shall not span *exec* or *spawn* operations.
- 6877 11. A triggering API is not required. The triggering API is the ability to command or stop  
6878 tracing based on the occurrence of specific trace event other than a `POSIX_TRACE_START`  
6879 trace event or a `POSIX_TRACE_STOP` trace event.
- 6880 12. Trace log entries shall have timestamps of implementation-defined resolution.  
6881 Implementations are exhorted to support at least microsecond resolution. When a trace log  
6882 entry is retrieved, it shall have timestamp, PC address, PID, and TID of the entity that  
6883 generated the trace event.
- 6884 13. Independently developed code should be able to use trace facilities without coordination  
6885 and without conflict.
- 6886 14. Even if the trace points in the trace calls are not unique, the trace log entries (after any  
6887 processing) shall be uniquely identified as to trace point.
- 6888 15. There shall be a standard API to read the trace stream.
- 6889 16. The format of the trace stream and the trace log is opaque and unspecified.
- 6890 17. It shall be possible to read a completed trace, if recorded on some suitable non-volatile  
6891 storage, even subsequent to a power cycle or subsequent cold boot of the system.
- 6892 18. Support of analysis of a trace log while it is being formed is implementation-defined.
- 6893 19. The API shall allow the application to write trace stream identification information into the  
6894 trace stream and to be able to retrieve it, without it being overwritten by trace entries, even  
6895 if the trace stream is full.
- 6896 20. It must be possible to specify the destination of trace data produced by trace events.
- 6897 21. It must be possible to have different trace streams, and for the tracing enabled by one trace  
6898 stream to be completely independent of the tracing of another trace stream.
- 6899 22. It must be possible to trace events from threads in different CPUs.
- 6900 23. The API shall support one or more trace streams per-system, and one or more trace  
6901 streams per-process, up to an implementation-defined set of per-system and per-process  
6902 maximums.
- 6903 24. It shall be possible to determine the order in which the trace events happened, without  
6904 necessarily depending on the clock, up to an implementation-defined time resolution.
- 6905 25. For performance reasons, the trace event point call(s) shall be implementable as a macro  
6906 (see the ISO POSIX-1: 1996 standard, Subclause 1.3.4, Statement 2).
- 6907 26. IEEE Std. 1003.1-200x must not define the trace points which a conforming system must  
6908 implement, except for trace points used in the control of tracing.
- 6909 27. The APIs shall be thread-safe, and trace points should be lock-free (that is shall not require  
6910 a lock to gain exclusive access to some resource).

- 6911 28. The user-provided information associated with a trace event is variable-sized, up to some  
6912 maximum size.
- 6913 29. Bounds on record and trace stream sizes:
- 6914 a. The API must permit the application to declare the upper bounds on the length of an  
6915 application data record. The system shall return the limit it used. The limit used may  
6916 be smaller than requested.
- 6917 b. The API must permit the application to declare the upper bounds on the size of trace  
6918 streams. The system shall return the limit it used. The limit used may be different,  
6919 either larger or smaller, than requested.
- 6920 30. The API must be able to pass any fundamental data type, and a structured data type  
6921 composed only of fundamental types. The API must be able to pass data by reference,  
6922 given only as an address and a length. Fundamental types are the POSIX.1 types (see the  
6923 ISO POSIX-1:1996 standard, Subclause 2.5, Table 2-1) plus those defined in the ISO C  
6924 standard.
- 6925 31. The API shall apply the POSIX notions of ownership and permission to recorded trace  
6926 data, corresponding to the sources of that data.

### 6927 **Comments on Objectives**

- 6928 **Note:** In the following comments, numbers in square brackets refer to the above objectives.
- 6929 It is necessary to be able to obtain a trace stream for a complete activity. This means we need to  
6930 be able to trace both application and system trace events. A per-process trace event is either  
6931 user-initiated, like the *write()* POSIX call, or system-initiated, like a timer expiration. We also  
6932 need to be able to trace an entire process's activity even when it has threads in multiple CPUs. To  
6933 avoid excess trace activity, it is necessary to be able to control tracing on a trace event type basis.  
6934 [Objectives 1,2,5,22]
- 6935 We need to be able to control tracing on a per-process basis, both from inside and outside the  
6936 process; that is, a process can start a trace activity on itself or any other process. We also see the  
6937 need to allow the definition of a maximum number trace streams per system.  
6938 [Objectives 3,23]
- 6939 From within a process, it is necessary to be able to control tracing on a per-thread basis. This  
6940 provides an additional filtering capability to keep the amount of traced data to a minimum. It  
6941 also allows for less ambiguity as to the origin of trace events. It is recognized that thread-level  
6942 control is only valid from within the process itself. It is also desirable to know the maximum  
6943 number of trace streams per process that can be started. We do not want the API to require  
6944 thread synchronization or to mandate priority inversions that would cause the thread to block.  
6945 However, the API must be thread-safe.  
6946 [Objectives 4,23,24,27]
- 6947 We see no objective to control tracing on anything larger than a process; for example, a group or  
6948 session. Also, the ability to start or stop a trace activity on multiple processes atomically may be  
6949 very difficult or cumbersome in some implementations.  
6950 [Objectives 6,8]
- 6951 It is also necessary to be able to control tracing by trace event type identifier, sometimes called a  
6952 trace hook ID. However, there is no mandated set of system trace events, since such trace points  
6953 are very system-dependent. The API must not require from the operating system facilities that  
6954 are not standard (POSIX).  
6955 [Objectives 6,26]

6956 Trace control must span *fork()* and *pthread\_create()*. If not, there will be no way to ensure that a  
6957 program's activity is entirely traced. The newly forked child would not be able to turn on its  
6958 tracing until after it obtained control after the fork, and trace control externally would be even  
6959 more problematic.  
6960 [Objective 9]

6961 Since *exec()* and *spawn()* represent a complete change in the execution of a task (a new  
6962 program), trace control need not persist over an *exec()* or *spawn()*.  
6963 [Objective 10]

6964 Where trace activities are started on multiple processes, these trace activities should not interfere  
6965 with each other.  
6966 [Objective 21]

6967 There is no need for a triggering objective, primarily for performance reasons; see also Section  
6968 B.2.11.8 (on page 3498), rationale on triggering.  
6969 [Objective 11]

6970 It must be possible to determine the origin of each traced event. We need the process and thread  
6971 identifiers for each trace event. We also saw the need for a user-specifiable origin, but felt this  
6972 would create too much overhead.  
6973 [Objectives 12,14]

6974 We must allow for trace points to come embedded in software components from several  
6975 different sources and vendors without requiring coordination.  
6976 [Objective 13]

6977 We need to be able to uniquely identify trace points that may have the same trace stream  
6978 identifier. We only need to be able to do this when a trace report is produced.  
6979 [Objectives 12,14]

6980 Tracing is a very performance-sensitive activity, and will therefore likely be implemented at a  
6981 low level within the system. Hence the interface shall not mandate any particular buffering or  
6982 storage method. Therefore, we will need a standard API to read a trace stream. Also the interface  
6983 shall not mandate the format of the trace data, and the interface shall not assume a trace storage  
6984 method. Due to the possibility of a monolithic kernel and the possible presence of multiple  
6985 processes capable of running trace activities, the two kinds of trace events may be stored in two  
6986 separate streams for performance reasons. A mandatory dump mechanism, common in some  
6987 existing practice, has been avoided to allow the implementation of this set of functions on small  
6988 realtime profiles for which the concept of a file system is not defined. The trace API calls should  
6989 be implemented as macros.  
6990 [Objectives 15,16,25,30]

6991 Since a trace facility is a valuable service tool, the output (or log) of a completed trace stream  
6992 that is written to permanent storage must be readable on other systems of the type that  
6993 produced the trace log. Note that there is no objective to be able to interpret a trace log that was  
6994 not successfully completed.  
6995 [Objectives 17,18,19]

6996 For trace streams written to permanent storage, a way to specify the destination of the trace  
6997 stream is needed.  
6998 [Objective 20]

6999 We need to be able to depend on the ordering of trace events up to some system-defined time  
7000 interval. For example, we need to know the time period which, if trace events are closer together,  
7001 their ordering is indeterminate. Events that occur within an interval smaller than this resolution  
7002 may or may not be read back in the correct order.

- 7003 [Objective 24]
- 7004 The application should be able to know how much data can be traced. When trace event types
- 7005 can be filtered, the application should be able to specify the approximate maximum amount of
- 7006 data that will be traced in a trace event so resources can be more efficiently allocated.
- 7007 [Objectives 28,29]
- 7008 Users should not be able to trace data to which they would not normally have access to. System
- 7009 trace events corresponding to a process/thread should be associated with the ownership of that
- 7010 process/thread.
- 7011 [Objective 31]

7012 *B.2.11.2 Trace Model*

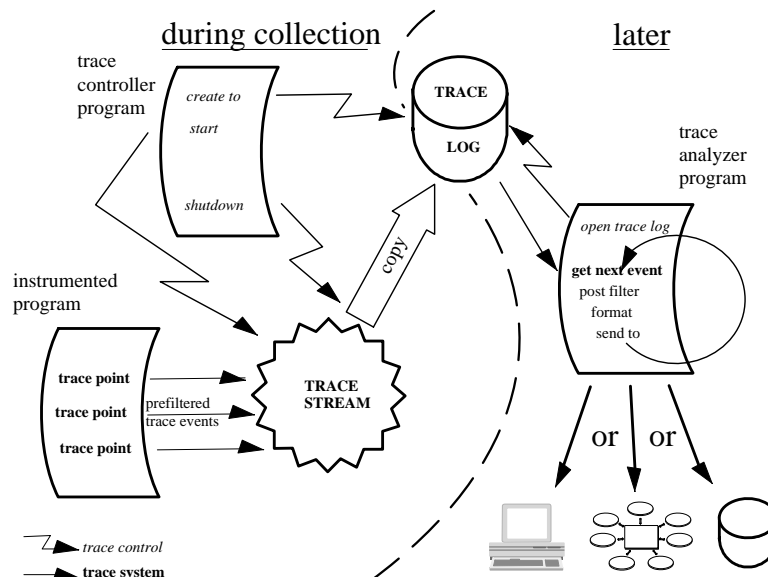
7013 **Introduction**

7014 The model is based on two base entities: the “Trace Stream” and the “Trace Log” , and a  
 7015 recorded unit called the “Trace Event”. The possibility of using Trace Streams and Trace Logs  
 7016 separately gives us two use dimensions and solves both the performance issue and the full-  
 7017 information system issue. In the case of a trace stream without log, specific information,  
 7018 although reduced in quantity, is required to be registered, in a possibly small realtime system,  
 7019 with as little overhead as possible. The Trace Log option has been added for small realtime  
 7020 systems. In the case of a trace stream with log, considerable complex application-specific  
 7021 information needs to be collected.

7022 **Trace Model Description**

7023 The trace model can be examined for three different subfunctions: Application Instrumentation,  
 7024 Trace Operation Control, and Trace Analysis.

7025



7026 **Figure B-2** Trace System Overview: for Offline Analysis

7027 Each of these subfunctions requires specific characteristics of the trace mechanism API.

## 7028 • Application Instrumentation

7029 When instrumenting an application, the programmer has no concern about the future  
 7030 utilization of the trace events in trace stream or trace log, the full policy of trace stream, or  
 7031 the eventual pre-filtering of trace events. But he is concerned about the correct determination  
 7032 of specific trace event type identifier, regardless of how many independent libraries are used  
 7033 in the same user application; see Figure B-2 and Figure B-3 (on page 3482).

7034 This trace API shall provide the necessary operations to accomplish this subfunction. This is  
 7035 done by providing functions to associate a programmer-defined name with an  
 7036 implementation-defined trace event type identifier; see the *posix\_trace\_eventid\_open()*  
 7037 function), and to send this trace event into a potential trace stream (see the  
 7038 *posix\_trace\_event()* function).

## 7039 • Trace Operation Control

7040 When controlling the recording of trace events in a trace stream, the programmer is  
 7041 concerned with the correct initialization of the trace mechanism (that is, the sizing of the  
 7042 trace stream), the correct retention of trace events in a permanent storage, the correct  
 7043 dynamic recording of trace events, and so on.

7044 This trace API shall provide the necessary material to permit this efficiently. This is done by  
 7045 providing functions to initialize a new trace stream, and optionally a trace log:

- 7046 — Trace Stream Attributes Object Initialization (see *posix\_trace\_attr\_init()*)
- 7047 — Functions to Retrieve or Set Information About a Trace Stream (see  
 7048 *posix\_trace\_attr\_getgenversion()*)
- 7049 — Functions to Retrieve or Set the Behavior of a Trace Stream (see  
 7050 *posix\_trace\_attr\_getinherited()*)
- 7051 — Functions to Retrieve or Set Trace Stream Size Attributes (see  
 7052 *posix\_trace\_attr\_getmaxuseventsizesize()*)
- 7053 — Trace Stream Initialization, Flush, and Shutdown from a Process (see *posix\_trace\_create()*)
- 7054 — Clear Trace Stream and Trace Log (see *posix\_trace\_clear()*)

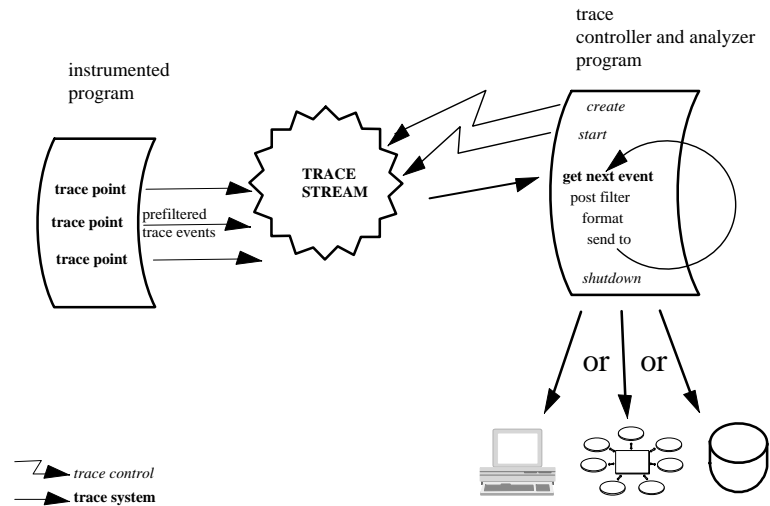
7055 To select the trace event types that are to be traced:

- 7056 — Manipulate Trace Event Type Identifier (see *posix\_trace\_trid\_eventid\_open()*)
- 7057 — Iterate over a Mapping of Trace Event Type (see *posix\_trace\_eventtypelist\_getnext\_id()*)
- 7058 — Manipulate Trace Event Type Sets (see *posix\_trace\_eventset\_empty()*)
- 7059 — Set Filter of an Initialized Trace Stream (see *posix\_trace\_set\_filter()*)

7060 To control the execution of an active trace stream:

- 7061 — Trace Start and Stop (see *posix\_trace\_start()*)
- 7062 — Functions to Retrieve the Trace Attributes or Trace Statuses (see *posix\_trace\_get\_attr()*)

7063



7064

**Figure B-3** Trace System Overview: for Online Analysis

7065

- Trace Analysis

7066

7067

7068

Once correctly recorded, on permanent storage or not, an ultimate activity consists of the analysis of the recorded information. If the recorded data is on permanent storage, a specific open operation is required to associate a trace stream to a trace log.

7069

7070

7071

7072

7073

7074

The first intent of the group was to request the presence of a system identification structure in the trace stream attribute. This was, for the application, to allow some portable way to process the recorded information. However, there is no requirement that the **utsname** structure, on which this system identification was based, be portable from one machine to another, so the contents of the attribute cannot be interpreted correctly by an application conforming to IEEE Std. 1003.1-200x.

7075

7076

7077

7078

7079

Draft 6 incorporates this modification and requests that some unspecified information be recorded in the trace log in order to fail opening it if the analysis process and the controller process were running in different types of machine, but does not request that this information be accessible to the application. This modification has implied a modification in the `posix_trace_open()` function error code returns.

7080

This trace API shall provide functions to:

7081

- Extract trace stream identification attributes (see `posix_trace_attr_getgenversion()`)

7082

- Extract trace stream behavior attributes (see `posix_trace_attr_getinherited()`)

7083

7084

- Extract trace event, stream, and log size attributes (see `posix_trace_attr_getmaxusereventsize()`)

7085

- Look up trace event type names (see `posix_trace_eventid_get_name()`)

- 7086 — Iterate over trace event type identifiers (see *posix\_trace\_eventtypelist\_getnext\_id()*)
- 7087 — Open, rewind, and close a trace log (see *posix\_trace\_open()*)
- 7088 — Read trace stream attributes and status (see *posix\_trace\_get\_attr()*)
- 7089 — Read trace events (see *posix\_trace\_getnext\_event()*)

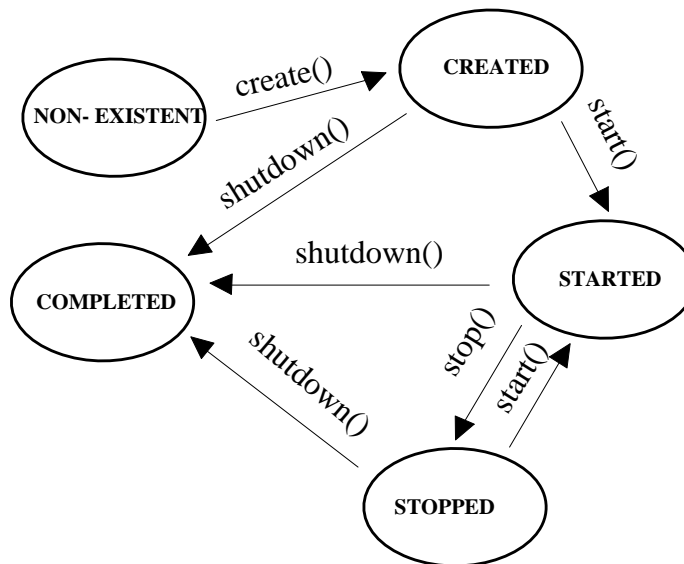
7090 Due to the following two reasons:

- 7091 1. The requirement that the trace system must not add unacceptable overhead to the traced process and so that the trace event point execution must be fast
- 7092
- 7093 2. The traced application does not care about tracing errors

7094 the trace system cannot return any internal error to the application. Internal error conditions can range from unrecoverable errors that will force the active trace stream to abort, to small errors that can affect the quality of tracing without aborting the trace stream. The group decided to define a system trace event to report to the analysis process such internal errors. It is not the intention of IEEE Std. 1003.1-200x to require an implementation to report an internal error that corrupts or terminates tracing operation. The implementor is free to decide which internal documented errors, if any, the trace system is able to report.

7101 **States of a Trace Stream**

7102



7103 **Figure B-4** Trace System Overview: States of a Trace Stream

7104 Figure B-4 shows the different states an active trace stream passes through. After the  
 7105 *posix\_trace\_create()* function call, a trace stream becomes CREATED and a trace stream is  
 7106 associated for the future collection of trace events. The status of the trace stream is  
 7107 POSIX\_TRACE\_SUSPENDED. The state becomes STARTED after a call to the *posix\_trace\_start()*  
 7108 function, and the status becomes POSIX\_TRACE\_RUNNING. In this state, all trace events that  
 7109 are not filtered out shall be stored into the trace stream. After a call to *posix\_trace\_stop()*, the

7110 trace stream becomes STOPPED (and the status POSIX\_TRACE\_SUSPENDED). In this state, no  
 7111 new trace events will be recorded in the trace stream, but previously recorded trace events may  
 7112 continue to be read.

7113 After a call to *posix\_trace\_shutdown()*, the trace stream is in the state COMPLETED. The trace  
 7114 stream no longer exists but, if the Trace Log option is supported, all the information contained in  
 7115 it has been logged. If a log object has not been associated with the trace stream at the creation, it  
 7116 is the responsibility of the trace controller process to not shut the trace stream down while trace  
 7117 events remain to be read in the stream.

### 7118 **Tracing All Processes**

7119 Some implementations have a tracing subsystem with the ability to trace all processes. This is  
 7120 useful to debug some types of device drivers such as those for ATM or X25 adapters. These types  
 7121 of adapters are used by several independent processes, that are not issued from the same  
 7122 process.

7123 The POSIX trace interface does not define any constant or option to create a trace stream tracing  
 7124 all processes. But the POSIX trace interface does not prevent this type of implementation and the  
 7125 implementor is free to add this capability. Nevertheless, the POSIX trace interface allows to trace  
 7126 all the system trace events and all the processes issued from the same process.

7127 If such a tracing system capability has to be implemented, when a trace stream is created, it is  
 7128 recommended that a constant named POSIX\_TRACE\_ALLPROC be used instead of the process  
 7129 identifier in the argument of the function *posix\_trace\_create()* or *posix\_trace\_create\_withlog()*. A  
 7130 possible value for POSIX\_TRACE\_ALLPROC may be -1 instead of a real process identifier.

7131 The implementor has to be aware that there is some impact on the tracing behavior as defined in  
 7132 the POSIX trace interface. For example:

- 7133 • If the default value for the inheritance attribute is to set to  
 7134 POSIX\_TRACE\_CLOSE\_FOR\_CHILD, the implementation has to stop tracing for the child  
 7135 process.
- 7136 • The trace controller which is creating this type of trace stream must have the appropriate  
 7137 privilege to trace all the processes.

### 7138 **Trace Storage**

7139 The model is based on two types of trace events: system trace events and user-defined trace  
 7140 events. The internal representation of trace events is implementation-defined, and so the  
 7141 implementor is free to choose the more suitable, practical, and efficient way to design the  
 7142 internal management of trace events. For the timestamping operation, the model does not  
 7143 impose the CLOCK\_REALTIME or any other clock. The buffering allocation and operation  
 7144 follow the same principle. The implementor is free to use one or more buffers to record trace  
 7145 events; the interface assumes only a logical trace stream of sequentially recorded trace events.  
 7146 Regarding flushing of trace events, the interface allows the definition of a trace log object which  
 7147 typically can be a file. But the group was also aware of defining functions to permit the use of  
 7148 this interface in small realtime systems, which may not have general file system capabilities. For  
 7149 instance, the three functions *posix\_trace\_getnext\_event()* (blocking),  
 7150 *posix\_trace\_timedgetnext\_event()* (blocking with timeout), and *posix\_trace\_trygetnext\_event()*  
 7151 (non-blocking) are proposed to read the recorded trace events.

7152 The policy to be used when the trace stream becomes full also relies on common practice:

- 7153 • For an active trace stream, the POSIX\_TRACE\_LOOP trace stream policy permits automatic  
 7154 overrun (overwrite of oldest trace events) while waiting for some user-defined condition to



7155 cause tracing to stop. By contrast, the POSIX\_TRACE\_UNTIL\_FULL trace stream policy  
 7156 requires the system to stop tracing when the trace stream is full. However, if the trace stream  
 7157 that is full is at least partially emptied by a call to the *posix\_trace\_flush()* function or by calls  
 7158 to *posix\_trace\_getnext\_event()* function, the trace system will automatically resume tracing.

7159 If the Trace Log option is supported the operation of the POSIX\_TRACE\_FLUSH policy is an  
 7160 extension of the POSIX\_TRACE\_UNTIL\_FULL policy. The automatic free operation (by  
 7161 flushing to the associated trace log) is added.

7162 • If a log is associated with the trace stream and this log is a regular file, these policies also  
 7163 apply for the log. One more policy, POSIX\_TRACE\_APPEND, is defined to allow indefinite  
 7164 extension of the log. Since the log destination can be any device or pseudo-device, the  
 7165 implementation may not be able to manipulate the destination as required by  
 7166 IEEE Std. 1003.1-200x. For this reason, the behavior of the log full policy may be unspecified  
 7167 depending of the trace log type.

7168 The current trace interface does not define a service to preallocate space for a trace log file,  
 7169 because this space can be preallocated by means of a call to the *posix\_fallocate()* function. This  
 7170 function could be called after the file has been opened, but before the trace stream is created.  
 7171 The *posix\_fallocate()* function ensures that any required storage for regular file data is  
 7172 allocated on the file system storage media. If *posix\_fallocate()* returns successfully,  
 7173 subsequent writes to the specified file data shall not fail due to the lack of free space on the  
 7174 file system storage media. Besides trace events, a trace stream also includes trace attributes  
 7175 and the mapping from trace event names to trace event type identifiers. The implementor is  
 7176 free to choose how to store the trace attributes and the trace event type map, but must ensure  
 7177 that this information is not lost when a trace stream overrun occurs.

### 7178 B.2.11.3 Trace Programming Examples

7179 Several programming examples are presented to show the code of the different possible  
 7180 subfunctions using a trace subsystem. All these programs need to include the `<trace.h>` header.  
 7181 In the examples shown, error checking is omitted for more simplicity.

### 7182 Trace Operation Control

7183 These examples show the creation of a trace stream for another process; one which is already  
 7184 trace instrumented. All the default trace stream attributes are used to simplify programming in  
 7185 the first example. The second example shows more possibilities.

### 7186 First Example

```
7187 /* Caution. Error checks omitted */
7188 {
7189 trace_attr_t attr;
7190 pid_t pid = traced_process_pid;
7191 int fd;
7192 trace_id_t trid;
7193 - - - - -
7194 /* Initialize trace stream attributes */
7195 posix_trace_attr_init(&attr);
7196 /* Open a trace log */
7197 fd=open("/tmp/mytracelog",...);
7198 /*
7199 * Create a new trace associated with a log
7200 * and with default attributes
```

```

7201 */
7202 posix_trace_create_withlog(pid, &attr, fd, &trid);
7203 /* Trace attribute structure can now be destroyed */
7204 posix_trace_attr_destroy(&attr);
7205 /* Start of trace event recording */
7206 posix_trace_start(trid);
7207 - - - - -
7208 - - - - -
7209 /* Duration of tracing */
7210 - - - - -
7211 - - - - -
7212 /* Stop and shutdown of trace activity */
7213 posix_trace_shutdown(trid);
7214 - - - - -
7215 }

```

## 7216 Second Example

7217 Between the initialization of the trace stream attributes and the creation of the trace stream,  
7218 these trace stream attributes may be modified; see **Trace Stream Attribute Manipulation** (on  
7219 page 3490) for specific programming example. Between the creation and the start of the trace  
7220 stream, the event filter may be set; after the trace stream is started, the event filter may be  
7221 changed. The setting of an event set and the change of a filter is shown in **Create a Trace Event**  
7222 **Type Set and Change the Trace Event Type Filter** (on page 3490).

```

7223 /* Caution. Error checks omitted */
7224 {
7225 trace_attr_t attr;
7226 pid_t pid = traced_process_pid;
7227 int fd;
7228 trace_id_t trid;
7229 - - - - -
7230 /* Initialize trace stream attributes */
7231 posix_trace_attr_init(&attr);
7232 /* Attr default may be changed at this place; see example */
7233 - - - - -
7234 /* Create and open a trace log with R/W user access */
7235 fd=open("/tmp/mytracelog",O_WRONLY|O_CREAT,S_IRUSR|S_IWUSR);
7236 /* Create a new trace associated with a log */
7237 posix_trace_create_withlog(pid, &attr, fd, &trid);
7238 /*
7239 * If the Trace Filter option is supported
7240 * trace event type filter default may be changed at this place;
7241 * see example about changing the trace event type filter
7242 */
7243 posix_trace_start(trid);
7244 - - - - -
7245
7246 /*
7247 * If you have an uninteresting part of the application
7248 * you can stop temporarily.
7249 */

```

```

7249 * posix_trace_stop(trid);
7250 * - - - - -
7251 * - - - - -
7252 * posix_trace_start(trid);
7253 */
7254 - - - - -
7255 /*
7256 * If the Trace Filter option is supported
7257 * the current trace event type filter can be changed
7258 * at any time (see example about how to set
7259 * a trace event type filter
7260 */
7261 - - - - -
7262 /* Stop the recording of trace events */
7263 posix_trace_stop(trid);
7264 /* Shutdown the trace stream */
7265 posix_trace_shutdown(trid);
7266 /*
7267 * Destroy trace stream attributes; attr structure may have
7268 * been used during tracing to fetch the attributes
7269 */
7270 posix_trace_attr_destroy(&attr);
7271 - - - - -
7272 }

```

### 7273 **Application Instrumentation**

7274 This example shows an instrumented application. The code is included in a block of instructions,  
7275 perhaps a function from a library. Possibly in an initialization part of the instrumented  
7276 application, two user trace event names are mapped to two trace event type identifiers  
7277 (function *posix\_trace\_eventid\_open()*). Then two trace points are programmed.

```

7278 /* Caution. Error checks omitted */
7279 {
7280 trace_eventid_t eventid1, eventid2;
7281 - - - - -
7282 /* Initialization of two trace event type ids */
7283 posix_trace_eventid_open("my_first_event",&eventid1);
7284 posix_trace_eventid_open("my_second_event",&eventid2);
7285 - - - - -
7286 - - - - -
7287 - - - - -
7288 /* Trace point */
7289 posix_trace_event(eventid1,NULL,0);
7290 - - - - -
7291 /* Trace point */
7292 posix_trace_event(eventid2,NULL,0);
7293 - - - - -
7294 }

```

7295 **Trace Analyzer**

7296 This example shows the manipulation of a trace log resulting from the dumping of a completed  
 7297 trace stream. All the default attributes are used to simplify programming, and data associated  
 7298 with a trace event are not shown in the first example. The second example shows more  
 7299 possibilities.

7300 **First Example**

```

7301 /* Caution. Error checks omitted */
7302 {
7303 int fd;
7304 trace_id_t trid;
7305 posix_trace_event_info trace_event;
7306 char trace_event_name[TRACE_EVENT_NAME_MAX];
7307 int return_value;
7308 size_t returndatasize;
7309 int lost_event_number;
7310
7311 - - - - -
7312
7313 /* Open an existing trace log */
7314 fd=open("/tmp/tracelog", O_RDONLY);
7315 /* Open a trace stream on the open log */
7316 posix_trace_open(fd, &trid);
7317 /* Read a trace event */
7318 posix_trace_getnext_event(trid, &trace_event,
7319 NULL, 0, &returndatasize,&return_value);
7320
7321 /* Read and print all trace event names out in a loop */
7322 while (return_value == NULL)
7323 {
7324 /*
7325 * Get the name of the trace event associated
7326 * with trid trace ID
7327 */
7328 posix_trace_eventid_get_name(trid, trace_event.event_id,
7329 trace_event_name);
7330 /* Print the trace event name out */
7331 printf("%s\n",trace_event_name);
7332 /* Read a trace event */
7333 posix_trace_getnext_event(trid, &trace_event,
7334 NULL, 0, &returndatasize,&return_value);
7335 }
7336
7337 /* Close the trace stream */
7338 posix_trace_close(trid);
7339 /* Close the trace log */
7340 close(fd);
7341 }

```

7338       **Second Example**

7339       The complete example includes the two other examples in **Retrieve Information from a Trace Log** (on page 3491) and in **Retrieve the List of Trace Event Types Used in a Trace Log** (on page 3492). For example, the *maxdatasize* variable is set in **Retrieve the List of Trace Event Types Used in a Trace Log** (on page 3492).

```

7343 /* Caution. Error checks omitted */
7344 {
7345 int fd;
7346 trace_id_t trid;
7347 posix_trace_event_info trace_event;
7348 char trace_event_name[TRACE_EVENT_NAME_MAX];
7349 char * data;
7350 size_t maxdatasize=1024, returndatasize;
7351 int return_value;
7352 - - - - -
7353 /* Open an existing trace log */
7354 fd=open("/tmp/tracelog", O_RDONLY);
7355 /* Open a trace stream on the open log */
7356 posix_trace_open(fd, &trid);
7357 /*
7358 * Retrieve information about the trace stream which
7359 * was dumped in this trace log (see example)
7360 */
7361 - - - - -
7362 /* Allocate a buffer for trace event data */
7363 data=(char *)malloc(maxdatasize);
7364 /*
7365 * Retrieve the list of trace event used in this
7366 * trace log (see example)
7367 */
7368 - - - - -
7369 /* Read and print all trace event names and data out in a loop */
7370 while (1)
7371 {
7372 posix_trace_getnext_event(trid, &trace_event,
7373 data, maxdatasize, &returndatasize,&return_value);
7374 if (return_value != NULL) break;
7375 /*
7376 * Get the name of the trace event type associated
7377 * with trid trace ID
7378 */
7379 posix_trace_eventid_get_name(trid, trace_event.event_id,
7380 trace_event_name);
7381 {
7382 int i;
7383 /* Print the trace event name out */
7384 printf("%s: ", trace_event_name);
7385 /* Print the trace event data out */
7386 for (i=0; i<returndatasize, i++) printf("%02.2X",

```

```

7387 (unsigned char)data[i]);
7388 printf("\n");
7389 }
7390 }
7391 /* Close the trace stream */
7392 posix_trace_close(trid);
7393 /* The buffer data is deallocated */
7394 free(data);
7395 /* Now the file can be closed */
7396 close(fd);
7397 }

```

### 7398 **Several Programming Manipulations**

7399 The following examples show some typical sets of operations needed in some contexts.

#### 7400 **Trace Stream Attribute Manipulation**

7401 This example shows the manipulation of a trace stream attribute object in order to change the  
7402 default value provided by a previous *posix\_trace\_attr\_init()* call.

```

7403 /* Caution. Error checks omitted */
7404 {
7405 trace_attr_t attr;
7406 size_t logsize=100000;
7407 - - - - -
7408 /* Initialize trace stream attributes */
7409 posix_trace_attr_init(&attr);
7410 /* Set the trace name in the attributes structure */
7411 posix_trace_attr_setname(&attr, "my_trace");
7412 /* Set the trace full policy */
7413 posix_trace_attr_setstreamfullpolicy(&attr, POSIX_TRACE_LOOP);
7414 /* Set the trace log size */
7415 posix_trace_attr_setlogsize(&attr, logsize);
7416 - - - - -
7417 }

```

#### 7418 **Create a Trace Event Type Set and Change the Trace Event Type Filter**

7419 This example is valid only if the Trace Event Filter option is supported. This example shows the  
7420 manipulation of a trace event type set in order to change the trace event type filter for an existing  
7421 active trace stream, which may be just-created, running, or suspended. Some sets of trace event  
7422 types are well-known, such as the set of trace event types not associated with a process, some  
7423 trace event types are just-built trace event types for this trace stream; one trace event type is the  
7424 predefined trace event error type which is deleted from the trace event type set.

```

7425 /* Caution. Error checks omitted */
7426 {
7427 trace_id_t trid = existing_trace;
7428 trace_event_set_t set;
7429 trace_event_id_t trace_event1, trace_event2;
7430 - - - - -
7431 /* Initialize to an empty set of trace event types */

```

```

7432 posix_trace_eventset_emptyset(&set);
7433 /*
7434 * Fill the set with all system trace events
7435 * not associated with a process
7436 */
7437 posix_trace_eventset_fill(&set, POSIX_TRACE_WOPID_EVENTS);
7438 /*
7439 * Get the trace event type identifier of the known trace event name
7440 * my_first_event for the trid trace stream
7441 */
7442 posix_trace_trid_eventid_open(trid, "my_first_event", &trace_event1);
7443 /* Add the set with this trace event type identifier */
7444 posix_trace_eventset_add_event(trace_event1, &set);
7445 /*
7446 * Get the trace event type identifier of the known trace event name
7447 * my_second_event for the trid trace stream
7448 */
7449
7450 posix_trace_trid_eventid_open(trid, "my_second_event", &trace_event2);
7451 /* Add the set with this trace event type identifier */
7452 posix_trace_eventset_add_event(trace_event2, &set);
7453 - - - - -
7454 /* Delete the system trace event POSIX_TRACE_ERROR from the set */
7455 posix_trace_eventset_del_event(POSIX_TRACE_ERROR, &set);
7456 - - - - -
7457 /* Modify the trace stream filter making it equal to the new set */
7458 posix_trace_set_filter(trid, &set, POSIX_TRACE_SET_EVENTSET);
7459 - - - - -
7460 /*
7461 * Now trace_event1, trace_event2, and all system trace event types
7462 * not associated with a process, except for the POSIX_TRACE_ERROR
7463 * system trace event type, are filtered out of (not recorded in) the
7464 * existing trace stream.
7465 */
7466 }

```

#### 7466 **Retrieve Information from a Trace Log**

7467 This example shows how to extract information from a trace log, the dump of a trace stream.  
7468 This code:

```

7469 • Asks if the trace stream has lost trace events
7470 • Extracts the information about the version of the trace subsystem which generated this trace
7471 log
7472 • Retrieves the maximum size of trace event data; this may be used to dynamically allocate an
7473 array for extracting trace event data from the trace log without overflow
7474 /* Caution. Error checks omitted */
7475 {
7476 struct posix_trace_status_info statusinfo;
7477 trace_attr_t attr;
7478 trace_id_t trid = existing_trace;

```

```

7479 size_t maxdatasize;
7480 char genversion[TRACE_NAME_MAX];
7481 - - - - -
7482 /* Get the trace stream status */
7483 posix_trace_get_status(trid, &statusinfo);
7484 /* Detect an overrun condition */
7485 if (statusinfo.posix_stream_overrun_status == POSIX_TRACE_OVERRUN)
7486 printf("trace events have been lost\n");
7487
7488 /* Get attributes from the trid trace stream */
7489 posix_trace_get_attr(trid, &attr);
7490 /* Get the trace generation version from the attributes */
7491 posix_trace_attr_getgenversion(&attr, genversion);
7492 /* Print the trace generation version out */
7493 printf("Information about Trace Generator:%s\n",genversion);
7494
7495 /* Get the trace event max data size from the attributes */
7496 posix_trace_attr_getmaxdatasize(&attr, &maxdatasize);
7497 /* Print the trace event max data size out */
7498 printf("Maximum size of associated data:%d\n",maxdatasize);
7499 /* Destroy the trace stream attributes */
7500 posix_trace_attr_destroy(&attr);
7501 }

```

### Retrieve the List of Trace Event Types Used in a Trace Log

This example shows the retrieval of a trace stream's trace event type list. This operation may be very useful if you are interested only in tracking the type of trace events in a trace log.

```

7503 /* Caution. Error checks omitted */
7504 {
7505 trace_id_t trid = existing_trace;
7506 trace_event_id_t event_id;
7507 char event_name[TRACE_EVENT_NAME_MAX];
7508 int return_value;
7509 - - - - -
7510
7511 /*
7512 * In a loop print all existing trace event names out
7513 * for the trid trace stream
7514 */
7515 while (1)
7516 {
7517 posix_trace_eventtypelist_getnext_id(trid, &event_id
7518 &return_value);
7519 if (return_value != NULL) break;
7520 /*
7521 * Get the name of the trace event associated
7522 * with trid trace ID
7523 */
7524 posix_trace_eventid_get_name(trid, event_id, event_name);
7525 /* Print the name out */
7526 printf("%s\n", event_name);
7527 }

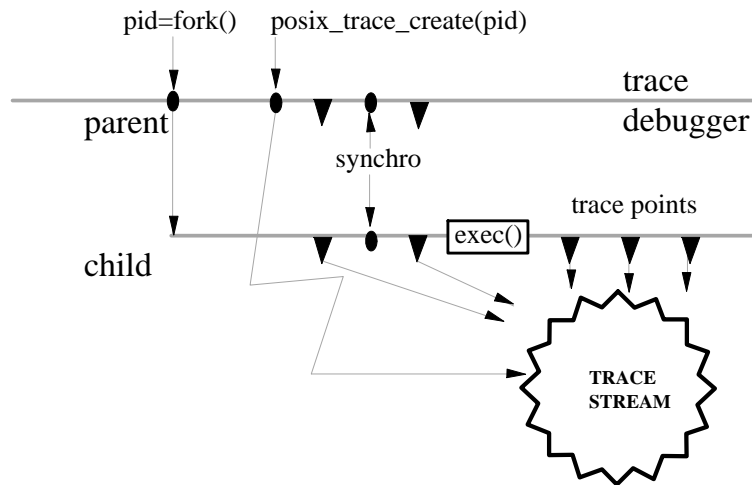
```



7527 }

7528 *B.2.11.4 Rationale on Trace for Debugging*

7529



7530

**Figure B-5** Trace Another Process

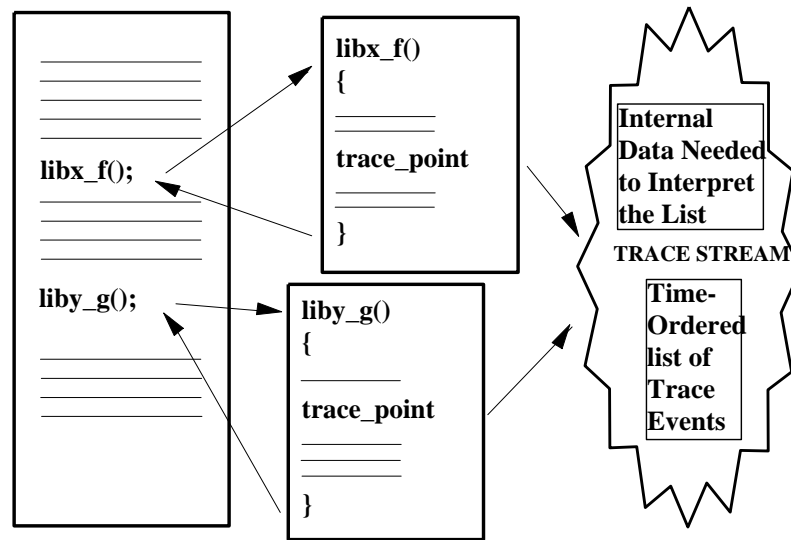
7531 Among the different possibilities offered by the trace interface defined in IEEE Std. 1003.1-200x,  
 7532 the debugging of an application is the most interesting one. Typical operations in the controlling  
 7533 debugger process are to filter trace event types, to get trace events from the trace stream, to stop  
 7534 the trace stream when the debugged process is executing uninteresting code, to start the trace  
 7535 stream when some interesting point is reached, and so on. The interface defined in  
 7536 IEEE Std. 1003.1-200x should define all the necessary base functions to allow this dynamic debug  
 7537 handling.

7538 Figure B-5 shows an example in which the trace stream is created after the call to the *fork()*  
 7539 function. If the user does not want to lose trace events some synchronization mechanism  
 7540 (represented in the figure) may be needed before calling the *exec()* function, to give the parent a  
 7541 chance to create the trace stream before the child begins the execution of its trace points.

7542 *B.2.11.5 Rationale on Trace Event Type Name Space*

7543 At first, the working group was in favor of the representation of a trace event type by an integer  
 7544 (*event\_name*). It seems that existing practice shows the weakness of such a representation. The  
 7545 collision of trace event types is the main problem that cannot be simply resolved using this sort  
 7546 of representation. Suppose, for example, that a third party designs an instrumented library. The  
 7547 user does not have the source of this library and wants to trace his application which uses in  
 7548 some part the third-party library. There is no means for him to know what are the trace event  
 7549 types used in the instrumented library so he has some chance of duplicating some of them and  
 7550 thus to obtain a contaminated tracing of his application.

7551



7552

**Figure B-6** Trace Name Space Overview: With Third-Party Library

7553 We have requirements to allow program images containing pieces from various vendors to be  
 7554 traced without also requiring those or any other vendors to coordinate their uses of the trace  
 7555 facility, and especially the naming of their various trace event types and trace point IDs. The  
 7556 chosen solution is to provide a very large name space, large enough so that the individual  
 7557 vendors can give their trace types and tracepoint IDs sufficiently long and descriptive names  
 7558 making the occurrence of collisions quite unlikely. The probability of collision is thus made  
 7559 sufficiently low so that the problem may, as a practical matter, be ignored. By requirement, the  
 7560 consequence of collisions will be a slight ambiguity in the trace streams; tracing will continue in  
 7561 spite of collisions and ambiguities. “The show must go on”. The *posix\_prog\_address* member of  
 7562 the **posix\_trace\_event\_info** structure is used to allow trace streams to be unambiguously  
 7563 interpreted, despite the fact that trace event types and trace event names need not be unique.

7564 The *posix\_trace\_eventid\_open()* function is required to allow the instrumented third-party library  
 7565 to get a valid trace event type identifier for its trace event names. This operation is, somehow,  
 7566 an allocation, and the group was aware of proposing some deallocation mechanism which the  
 7567 instrumented application could use to recover the resources used by a trace event type identifier.  
 7568 This would have given the instrumented application the benefit of being capable of reusing a  
 7569 possible minimum set of trace event type identifiers, but also the inconvenience to have,  
 7570 possibly in the same trace stream, one trace event type identifier identifying two different trace  
 7571 event types. After some discussions the group decided to not define such a function which  
 7572 would make this API thicker for little benefit, the user having always the possibility of adding  
 7573 identification information in the data member of the trace event structure.

7574 The set of the trace event type identifiers the controlling process wants to filter out is initialized  
 7575 in the trace mechanism using the function *posix\_trace\_set\_filter()*, setting the arguments  
 7576 according to the definitions explained in *posix\_trace\_set\_filter()*. This operation can be done  
 7577 statically (when the trace is in the STOPPED state) or dynamically (when the trace is in the  
 7578 STARTED state). The preparation of the filter is normally done using the function defined in

7579 *posix\_trace\_eventtypelist\_getnext\_id()* and eventually the function  
7580 *posix\_trace\_eventtypelist\_rewind()* in order to know (before the recording) the list of the potential  
7581 set of trace event types that can be recorded. In the case of an active trace stream, this list may  
7582 not be exhaustive. Actually, the target process may not have yet called the function  
7583 *posix\_trace\_eventid\_open()*. But it is a common practice, for a controlling process, to prepare the  
7584 filtering of a future trace stream before its start. Therefore the user must have a way to get the  
7585 trace event type identifier corresponding to a well-known trace event name before its future  
7586 association by the pre-cited function. This is done by calling the *posix\_trace\_trid\_eventid\_open()*  
7587 function, given the trace stream identifier and the trace name, and described hereafter. Because  
7588 this trace event type identifier is associated with a trace stream identifier, where a unique  
7589 process has initialized two or more traces, the implementation is expected to return the same  
7590 trace event type identifier for successive calls to *posix\_trace\_trid\_eventid\_open()* with different  
7591 trace stream identifiers. The *posix\_trace\_eventid\_get\_name()* function is used by the controller  
7592 process to identify, by the name, the trace event type returned by a call to the  
7593 *posix\_trace\_eventtypelist\_getnext\_id()* function.

7594 Afterwards, the set of trace event types is constructed using the functions defined in  
7595 *posix\_trace\_eventset\_empty()*, *posix\_trace\_eventset\_fill()*, *posix\_trace\_eventset\_add()*, and  
7596 *posix\_trace\_eventset\_del()*.

7597 A set of functions is provided devoted to the manipulation of the trace event type identifier and  
7598 names for an active trace stream. All these functions require the trace stream identifier argument  
7599 as the first parameter. The opacity of the trace event type identifier implies that the user cannot  
7600 associate directly its well-known trace event name with the system associated trace event type  
7601 identifier.

7602 The *posix\_trace\_trid\_eventid\_open()* function allows the application to get the system trace event  
7603 type identifier back from the system, given its well-known trace event name. One possible use of  
7604 this function is to qualify a filter.

7605 The *posix\_trace\_eventid\_get\_name()* function allows the application to obtain a trace event name  
7606 given its trace event type identifier. One possible use of this function is to identify the type of a  
7607 trace event retrieved from the trace stream, and print it. The easiest way to implement this  
7608 requirement, is to use a single trace event type map for all the processes whose maps are  
7609 required to be identical. A more difficult way is to attempt to keep multiple maps identical at  
7610 every call to *posix\_trace\_eventid\_open()* and *posix\_trace\_trid\_eventid\_open()*.

#### 7611 *B.2.11.6 Rationale on Trace Events Type Filtering*

7612 The most basic rationale for runtime and pre-registration filtering (selection/rejection) of trace  
7613 event types is to prevent choking of the trace collection facility, and/or overloading of the  
7614 computer system. Any worthwhile trace facility can bring even the largest computer to its  
7615 knees. Otherwise, we would record everything, and filter after the fact; it would be much  
7616 simpler, but impractical.

7617 To achieve debugging, measurement, or whatever the purpose of tracing, the filtering of trace  
7618 event types is an important part of trace analysis. Due to the fact that the trace events are put  
7619 into a trace stream and probably logged afterwards into a file, different levels of filtering—that  
7620 is, rejection of trace event types—are possible.

7621 **Filtering of Trace Event Types Before Tracing**

7622 This function, represented by the *posix\_trace\_set\_filter()* function in IEEE Std. 1003.1-200x (see  
 7623 *posix\_trace\_set\_filter()*), selects, before or during tracing, the set of trace event types to be filtered  
 7624 out. It should be possible also (as OSF suggested in their ETAP trace specifications) to select the  
 7625 kernel trace event types to be traced in a system-wide fashion. These two functionalities are  
 7626 called the pre-filtering of trace event types.

7627 The restriction on the actual type used for the **trace\_event\_set\_t** type is intended to guarantee  
 7628 that these objects can always be assigned, have their address taken, and be passed by value as  
 7629 parameters. It is not intended that this type be a structure including pointers to other data  
 7630 structures, as that could impact the portability of applications performing such operations. A  
 7631 reasonable implementation could be a structure containing an array of integer types.

7632 **Filtering of Trace Event Types at Runtime**

7633 Using this API, this functionality may be built, a privileged process or a privileged thread can  
 7634 get trace events from the trace stream of another process or thread, and thus specify the type of  
 7635 trace events to record into a file, using methods and interfaces out of the scope of  
 7636 IEEE Std. 1003.1-200x. This functionality, called inline filtering of trace event types, is used for  
 7637 runtime analysis of trace streams.

7638 **Post-Mortem Filtering of Trace Event Types**

7639 The word *post-mortem* is used here to indicate that some unanticipated situation occurs during  
 7640 execution that does not permit a pre or inline filtering of trace events and that it is necessary to  
 7641 record all trace event types, to have a chance to discover the problem afterwards. When the  
 7642 program stops, all the trace events recorded previously can be analyzed in order to find the  
 7643 solution. This functionality could be named the post-filtering of trace event types.

7644 **Discussions about Trace Event Type-Filtering**

7645 After long discussions with the parties involved in the process of defining the trace interface, it  
 7646 seems that the sensitivity to the filtering problem is different, but everybody agrees that the level  
 7647 of the overhead introduced during the tracing operation depends on the filtering method  
 7648 elected. If the time that it takes the trace event to be recorded can be neglected, the overhead  
 7649 introduced by the filtering process can be classified as follows:

7650 Pre-filtering      System and process/thread-level overhead

7651 Inline-filtering    Process/thread-level overhead

7652 Post-filtering     No overhead; done offline

7653 The pre-filtering could be named *critical realtime* filtering in the sense that the filtering of trace  
 7654 event type is manageable at the user level so the user can lower to a minimum the filtering  
 7655 overhead at some user selected level of priority for the inline filtering, or delay the filtering to  
 7656 after execution for the post-filtering. The counterpart of this solution is that the size of the trace  
 7657 stream must be sufficient to record all the trace events. The advantage of the pre-filtering is that  
 7658 the utilization of the trace stream is optimized.

7659 Only pre-filtering is defined by IEEE Std. 1003.1-200x. However, great care must be taken in  
 7660 specifying pre-filtering, so that it does not impose unacceptable overhead. Moreover, it is  
 7661 necessary to isolate all the functionality relative to the pre-filtering.

7662 The result of this rationale is to define a new option, the Trace Event Filter option, not  
 7663 necessarily implemented in small realtime systems, where system overhead is minimized to the  
 7664 extent possible.

7665 *B.2.11.7 Tracing, pthread API*

7666 The objective to be able to control tracing for individual threads may be in conflict with the  
 7667 efficiency expected in threads with a `contentionscope` attribute of  
 7668 `PTHREAD_SCOPE_PROCESS`. For these threads, context switches from one thread that has  
 7669 tracing enabled to another thread that has tracing disabled may require a kernel call to inform  
 7670 the kernel whether it has to trace system events executed by that thread or not. For this reason, it  
 7671 was proposed that the ability to enable or disable tracing for `PTHREAD_SCOPE_PROCESS`  
 7672 threads be made optional, through the introduction of a Trace Scope Process option. A trace  
 7673 implementation which did not implement the Trace Scope Process option would not honor the  
 7674 tracing-state attribute of a thread with `PTHREAD_SCOPE_PROCESS`; it would, however, honor  
 7675 the tracing-state attribute of a thread with `PTHREAD_SCOPE_SYSTEM`. This proposal was  
 7676 rejected as:

- 7677 1. Removing desired functionality (per-thread trace control)
- 7678 2. Introducing counter-intuitive behavior for the tracing-state attribute
- 7679 3. Mixing logically orthogonal ideas (thread scheduling and thread tracing)
- 7680 [Objective 4]

7681 Finally, to solve this complex issue, this API does not provide `pthread_gettracingstate()`,  
 7682 `pthread_settracingstate()`, `pthread_attr_gettracingstate()`, and `pthread_attr_settracingstate()`  
 7683 interfaces. These interfaces force the thread implementation to add to the weight of the thread  
 7684 and cause a revision of the threads libraries, just to support tracing. Worse yet,  
 7685 `posix_trace_userevent()` must always test this per-thread variable even in the common case where  
 7686 it is not used at all. Per-thread tracing is easy to implement using existing interfaces where  
 7687 necessary; see the following example.

7688 **Example**

```
7689 /* Caution. Error checks omitted */
7690 static pthread_key_t my_key;
7691 static trace_event_id_t my_event_id;
7692 static pthread_once_t my_once = PTHREAD_ONCE_INIT;
7693
7694 void my_init(void)
7695 {
7696 (void) pthread_key_create(&my_key, NULL);
7697 (void) posix_trace_eventid_open("my", &my_event_id);
7698 }
7699
7700 int get_trace_flag(void)
7701 {
7702 pthread_once(&my_once, my_init);
7703 return (pthread_getspecific(my_key) != NULL);
7704 }
7705
7706 void set_trace_flag(int f)
7707 {
7708 pthread_once(&my_once, my_init);
7709 pthread_setspecific(my_key, f? &my_event_id: NULL);
7710 }
7711
7712 fn()
7713 {
7714 if (get_trace_flag())
```

```
7711 posix_trace_event(my_event_id, ...)
7712 }
```

7713 The above example does not implement third-party state setting, but it is also implementable  
7714 with some more work, yet the extra functionality is rarely needed.

7715 Lastly, per-thread tracing works poorly for threads with PTHREAD\_SCOPE\_PROCESS  
7716 contention scope. These “library” threads have minimal interaction with the kernel and would  
7717 have to explicitly set the attributes whenever they are context switched to a new kernel thread in  
7718 order to trace system events. Such state was explicitly avoided in POSIX threads to keep  
7719 PTHREAD\_SCOPE\_PROCESS threads lightweight.

7720 The reason that keeping PTHREAD\_SCOPE\_PROCESS threads lightweight is important is that  
7721 such threads can be used not just for simple multi-processors but also for coroutine style  
7722 programming (such as discrete event simulation) without inventing a new threads paradigm.  
7723 Adding extra runtime cost to thread context switches will make using POSIX threads less  
7724 attractive in these situations.

#### 7725 *B.2.11.8 Rationale on Triggering*

7726 The ability to start or stop tracing based on the occurrence of specific trace event types has been  
7727 proposed as a parallel to similar functionality appearing in logic analyzers. Such triggering, in  
7728 order to be very useful, should be based not only on the trace event type, but on trace event-  
7729 specific data, including tests of user-specified fields for matching or threshold values.

7730 Such a facility is unnecessary where the buffering of the stream is not a constraint, since such  
7731 checks can be performed offline during post-mortem analysis.

7732 For example, a large system could incorporate a daemon utility to collect the trace records from  
7733 memory buffers and spool them to secondary storage for later analysis. In the instances where  
7734 resources are truly limited, such as embedded applications, the application incorporation of  
7735 application code to test the circumstances of a trace event and call the trace point only if needed  
7736 is usually straightforward.

7737 For performance reasons, the *posix\_trace\_event()* function should be implemented using a macro,  
7738 so if the trace is inactive, the trace event point calls are latent code and must cost no more than a  
7739 scalar test.

7740 The API proposed in IEEE Std. 1003.1-200x does not include any triggering functionality.

#### 7741 *B.2.11.9 Rationale on Timestamp Clock*

7742 It has been suggested that the tracing mechanism should include the possibility of specifying the  
7743 clock to be used in timestamping the trace events. When application trace events must be  
7744 correlated to remote trace events, such a facility could provide a global time reference not  
7745 available from a local clock. Further, the application may be driven by timers based on a clock  
7746 different from that used for the timestamp, and the correlation of the trace to those untraced  
7747 timer activities could be an important part of the analysis of the application.

7748 However, the tracing mechanism needs to be fast and just the provision of such an option can  
7749 materially affect its performance. Leaving aside the performance costs of reading some clocks,  
7750 this notion is also ill-defined when kernel trace events are to be traced by two applications  
7751 making use of different tracing clocks. This can even happen within a single application where  
7752 different parts of the application are served by different clocks. Another complication can occur  
7753 when a clock is maintained strictly at the user level and is unavailable at the kernel level.

7754 It is felt that the benefits of a selectable trace clock do not match its costs. Applications that wish  
7755 to correlate clocks other than the default tracing clock can include trace events with sample

7756 values of those other clocks, allowing correlation of timestamps from the various independent  
7757 clocks. In any case, such a technique would be required when applications are sensitive to  
7758 multiple clocks.

#### 7759 *B.2.11.10 Rationale on Different Overrun Conditions*

7760 The analysis of the dynamic behavior of the trace mechanism shows that different overrun  
7761 conditions may occur. The API must provide a means to manage such conditions in a portable  
7762 way.

#### 7763 **Overrun in Trace Streams Initialized with POSIX\_TRACE\_LOOP Policy**

7764 In this case, the user of the trace mechanism is interested in using the trace stream with  
7765 POSIX\_TRACE\_LOOP policy to record trace events continuously, but ideally without losing any  
7766 trace events. The online analyzer process must get the trace events at a mean speed equivalent to  
7767 the recording speed. Should the trace stream become full, a trace stream overrun occurs. This  
7768 condition is detected by getting the status of the active trace stream (function  
7769 *posix\_trace\_get\_status()*) and looking at the member *posix\_stream\_overrun\_status* of the read  
7770 **posix\_stream\_status** structure. In addition, two predefined trace event types are defined:

- 7771 1. The beginning of a trace overflow, to locate the beginning of an overflow when reading a  
7772 trace stream
- 7773 2. The end of a trace overflow, to locate the end of an overflow, when reading a trace stream

7774 As a timestamp is associated with these predefined trace events, it is possible to know the  
7775 duration of the overflow.

#### 7776 **Overrun in Dumping Trace Streams into Trace Logs**

7777 The user lets the trace mechanism dump the trace stream initialized with  
7778 POSIX\_TRACE\_FLUSH policy automatically into a trace log. If the dump operation is slower  
7779 than the recording of trace events, the trace stream can overrun. This condition is detected by  
7780 getting the status of the active trace stream (function *posix\_trace\_get\_status()*) and looking at the  
7781 member *posix\_log\_overrun\_status* of the read **posix\_stream\_status** structure. This overrun  
7782 indicates that the trace mechanism is not able to operate in this mode at this speed. It is the  
7783 responsibility of the user to modify one of the trace parameters (the stream size or the trace  
7784 event type filter, for instance) to avoid such overrun conditions, if overruns are to be prevented.  
7785 The same already predefined trace event types (see **Overrun in Trace Streams Initialized with**  
7786 **POSIX\_TRACE\_LOOP Policy**) are used to detect and to know the duration of an overflow.

#### 7787 **Reading an Active Trace Stream**

7788 Although this trace API allows one to read an active trace stream with log while it is tracing, this  
7789 feature can lead to false overflow origin interpretation: the trace log or the reader of the trace  
7790 stream. Reading from an active trace stream with log is thus non-portable, and has been left  
7791 unspecified.

7792 **B.2.12 Data Types**

7793 The requirement that additional types defined in this section end in “\_t” was prompted by the  
 7794 problem of name space pollution. It is difficult to define a type (where that type is not one  
 7795 defined by IEEE Std. 1003.1-200x) in one header file and use it in another without adding  
 7796 symbols to the name space of the program. To allow implementors to provide their own types,  
 7797 all conforming applications are required to avoid symbols ending in “\_t”, which permits the  
 7798 implementor to provide additional types. Because a major use of types is in the definition of  
 7799 structure members, which can (and in many cases must) be added to the structures defined in  
 7800 IEEE Std. 1003.1-200x, the need for additional types is compelling.

7801 The types, such as **ushort** and **ulong**, which are in common usage, are not defined in  
 7802 IEEE Std. 1003.1-200x (although **ushort\_t** would be permitted as an extension). They can be  
 7803 added to `<sys/types.h>` using a feature test macro (see Section B.2.2.1 (on page 3384)). A  
 7804 suggested symbol for these is `_SYSIII`. Similarly, the types like **u\_short** would probably be best  
 7805 controlled by `_BSD`.

7806 Some of these symbols may appear in other headers; see Section B.2.2.2 (on page 3384).

7807 **dev\_t** This type may be made large enough to accommodate host-locality considerations  
 7808 of networked systems.

7809 This type must be arithmetic. Earlier proposals allowed this to be non-arithmetic  
 7810 (such as a structure) and provided a `samefile()` function for comparison.

7811 **gid\_t** Some implementations had separated **gid\_t** from **uid\_t** before POSIX.1 was  
 7812 completed. It would be difficult for them to coalesce them when it was  
 7813 unnecessary. Additionally, it is quite possible that user IDs might be different than  
 7814 group IDs because the user ID might wish to span a heterogeneous network,  
 7815 where the group ID might not.

7816 For current implementations, the cost of having a separate **gid\_t** will be only  
 7817 lexical.

7818 **mode\_t** This type was chosen so that implementations could choose the appropriate  
 7819 integral type, and for compatibility with the ISO C standard. 4.3 BSD uses  
 7820 **unsigned short** and the SVID uses **ushort**, which is the same. Historically, only the  
 7821 low-order sixteen bits are significant.

7822 **nlink\_t** This type was introduced in place of **short** for `st_nlink` (see the `<sys/stat.h>` header)  
 7823 in response to an objection that **short** was too small.

7824 **off\_t** This type is used only in `lseek()`, `fcntl()`, and `<sys/stat.h>`. Many implementations  
 7825 would have difficulties if it were defined as anything other than **long**. Requiring  
 7826 an integral type limits the capabilities of `lseek()` to four gigabytes. The ISO C  
 7827 standard supplies routines that use larger types; see `fgetpos()` and `fsetpos()`. XSI-  
 7828 conformant systems provide the `fseeko()` and `lseeko()` functions that use larger  
 7829 types.

7830 **pid\_t** The inclusion of this symbol was controversial because it is tied to the issue of the  
 7831 representation of a process ID as a number. From the point of view of a portable  
 7832 application, process IDs should be “magic cookies”<sup>1</sup> that are produced by calls

7833 \_\_\_\_\_

7834 1. An historical term meaning: “An opaque object, or token, of determinate size, whose significance is known only to the entity  
 7835 which created it. An entity receiving such a token from the generating entity may only make such use of the ‘cookie’ as is defined  
 7836 and permitted by the supplying entity.”



7837 such as *fork()*, used by calls such as *waitpid()* or *kill()*, and not otherwise analyzed  
7838 (except that the sign is used as a flag for certain operations).

7839 The concept of a {PID\_MAX} value interacted with this in early proposals. Treating  
7840 process IDs as an opaque type both removes the requirement for {PID\_MAX} and  
7841 allows systems to be more flexible in providing process IDs that span a large range  
7842 of values, or a small one.

7843 Since the values in **uid\_t**, **gid\_t**, and **pid\_t** will be numbers generally, and  
7844 potentially both large in magnitude and sparse, applications that are based on  
7845 arrays of objects of this type are unlikely to be fully portable in any case. Solutions  
7846 that treat them as magic cookies will be portable.

7847 {CHILD\_MAX} precludes the possibility of a “toy implementation”, where there  
7848 would only be one process.

7849 **ssize\_t** This is intended to be a signed analog of **size\_t**. The wording is such that an  
7850 implementation may either choose to use a longer type or simply to use the signed  
7851 version of the type that underlies **size\_t**. All functions that return **ssize\_t** (*read()*  
7852 and *write()*) describe as “implementation-defined” the result of an input exceeding  
7853 {SSIZE\_MAX}. It is recognized that some implementations might have **ints** that  
7854 are smaller than **size\_t**. A portable application would be constrained not to  
7855 perform I/O in pieces larger than {SSIZE\_MAX}, but a portable application using  
7856 extensions would be able to use the full range if the implementation provided an  
7857 extended range, while still having a single type-compatible interface.

7858 The symbols **size\_t** and **ssize\_t** are also required in **<unistd.h>** to minimize the  
7859 changes needed for calls to *read()* and *write()*. Implementors are reminded that it  
7860 must be possible to include both **<sys/types.h>** and **<unistd.h>** in the same  
7861 program (in either order) without error.

7862 **uid\_t** Before the addition of this type, the data types used to represent these values  
7863 varied throughout early proposals. The **<sys/stat.h>** header defined these values as  
7864 type **short**, the **<passwd.h>** file (now **<pwd.h>** and **<grp.h>**) used an **int**, and  
7865 *getuid()* returned an **int**. In response to a strong objection to the inconsistent  
7866 definitions, all the types were switched to **uid\_t**.

7867 In practice, those historical implementations that use varying types of this sort can  
7868 typedef **uid\_t** to **short** with no serious consequences.

7869 The problem associated with this change concerns object compatibility after  
7870 structure size changes. Since most implementations will define **uid\_t** as a short, the  
7871 only substantive change will be a reduction in the size of the **passwd** structure.  
7872 Consequently, implementations with an overriding concern for object  
7873 compatibility can pad the structure back to its current size. For that reason, this  
7874 problem was not considered critical enough to warrant the addition of a separate  
7875 type to POSIX.1.

7876 The types **uid\_t** and **gid\_t** are magic cookies. There is no {UID\_MAX} defined by  
7877 POSIX.1, and no structure imposed on **uid\_t** and **gid\_t** other than that they be  
7878 positive arithmetic types. (In fact, they could be **unsigned char**.) There is no  
7879 maximum or minimum specified for the number of distinct user or group IDs.

7880 **B.3 System Interfaces**

7881 See the RATIONALE sections on the individual reference pages.

7882 **B.3.1 Examples for Spawn**7883 The following long examples are provided in the Rationale (Informative) volume of  
7884 IEEE Std. 1003.1-200x as a supplement to the reference page for *spawn()*.7885 **Example Library Implementation of Spawn**7886 The *posix\_spawn()* or *posix\_spawnp()* functions provide the following:

- 7887 • Simply start a process executing a process image. This is the simplest application for process  
7888 creation, and it may cover most executions of POSIX *fork()*.
- 7889 • Support I/O redirection, including pipes.
- 7890 • Run the child under a user and group ID in the domain of the parent.
- 7891 • Run the child at any priority in the domain of the parent.

7892 The *posix\_spawn()* or *posix\_spawnp()* functions do not cover every possible use of the *fork()*  
7893 function, but they do span the common applications: typical use by a shell and a login utility.7894 The price for an application is that before it calls *posix\_spawn()* or *posix\_spawnp()*, the parent  
7895 must adjust to a state that *posix\_spawn()* or *posix\_spawnp()* can map to the desired state for the  
7896 child. Environment changes require the parent to save some of its state and restore it afterwards.  
7897 The effective behavior of a successful invocation of *posix\_spawn()* is as if the operation were  
7898 implemented with POSIX operations as follows:

```

7899 #include <sys/types.h>
7900 #include <stdlib.h>
7901 #include <stdio.h>
7902 #include <unistd.h>
7903 #include <sched.h>
7904 #include <fcntl.h>
7905 #include <signal.h>
7906 #include <errno.h>
7907 #include <string.h>
7908 #include <signal.h>

7909 /* #include <spawn.h>*/
7910 /*****
7911 /* Things that could be defined in spawn.h */
7912 *****/
7913 typedef struct
7914 {
7915 short posix_attr_flags;
7916 #define POSIX_SPAWN_SETPGROUP 0x1
7917 #define POSIX_SPAWN_SETSIGMASK 0x2
7918 #define POSIX_SPAWN_SETSIGDEF 0x4
7919 #define POSIX_SPAWN_SETSCHEDULER 0x8
7920 #define POSIX_SPAWN_SETSCHEDPARAM 0x10
7921 #define POSIX_SPAWN_RESETIDS 0x20
7922 pid_t posix_attr_pgroup;
7923 sigset_t posix_attr_sigmask;
7924 sigset_t posix_attr_sigdefault;

```

```

7925 int posix_attr_schedpolicy;
7926 struct sched_param posix_attr_schedparam;
7927 } posix_spawnattr_t;
7928
7928 typedef char *posix_spawn_file_actions_t;
7929
7929 int posix_spawn_file_actions_init(
7930 posix_spawn_file_actions_t *file_actions);
7931 int posix_spawn_file_actions_destroy(
7932 posix_spawn_file_actions_t *file_actions);
7933 int posix_spawn_file_actions_addclose(
7934 posix_spawn_file_actions_t *file_actions, int fildes);
7935 int posix_spawn_file_actions_adddup2(
7936 posix_spawn_file_actions_t *file_actions, int fildes,
7937 int newfildes);
7938 int posix_spawn_file_actions_addopen(
7939 posix_spawn_file_actions_t *file_actions, int fildes,
7940 const char *path, int oflag, mode_t mode);
7941 int posix_spawnattr_init(posix_spawnattr_t *attr);
7942 int posix_spawnattr_destroy(posix_spawnattr_t *attr);
7943 int posix_spawnattr_getflags(const posix_spawnattr_t *attr, short *lags);
7944 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
7945 int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,
7946 pid_t *pgroup);
7947 int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
7948 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,
7949 int *schedpolicy);
7950 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
7951 int schedpolicy);
7952 int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
7953 struct sched_param *schedparam);
7954 int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
7955 const struct sched_param *schedparam);
7956 int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
7957 sigset_t *sigmask);
7958 int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
7959 const sigset_t *sigmask);
7960 int posix_spawnattr_getdefault(const posix_spawnattr_t *attr,
7961 sigset_t *sigdefault);
7962 int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
7963 const sigset_t *sigdefault);
7964 int posix_spawn(pid_t *pid, const char *path,
7965 const posix_spawn_file_actions_t *file_actions,
7966 const posix_spawnattr_t *attrp, char * const argv[],
7967 char * const envp[]);
7968 int posix_spawnnp(pid_t *pid, const char *file,
7969 const posix_spawn_file_actions_t *file_actions,
7970 const posix_spawnattr_t *attrp, char * const argv[],
7971 char * const envp[]);
7972
7972 /*****
7973 /* Example posix_spawn() library routine */
7974 /*****
7975 int posix_spawn(pid_t *pid,

```

```
7976 const char *path,
7977 const posix_spawn_file_actions_t *file_actions,
7978 const posix_spawnattr_t *attrp,
7979 char * const argv[],
7980 char * const envp[])
7981 {
7982 /* Create process */
7983 if((*pid=fork()) == (pid_t)0)
7984 {
7985 /* This is the child process */
7986 /* Worry about process group */
7987 if(attrp->posix_attr_flags & POSIX_SPAWN_SETPGROUP)
7988 {
7989 /* Override inherited process group */
7990 if(setpgid(0, attrp->posix_attr_pgroup) != 0)
7991 {
7992 /* Failed */
7993 exit(127);
7994 }
7995 }
7996
7997 /* Worry about process signal mask */
7998 if(attrp->posix_attr_flags & POSIX_SPAWN_SETSIGMASK)
7999 {
8000 /* Set the signal mask (can't fail) */
8001 sigprocmask(SIG_SETMASK , &attrp->posix_attr_sigmask,
8002 NULL);
8003 }
8004
8005 /* Worry about resetting effective user and group IDs */
8006 if(attrp->posix_attr_flags & POSIX_SPAWN_RESETEIDS)
8007 {
8008 /* None of these can fail for this case. */
8009 setuid(getuid());
8010 setgid(getgid());
8011 }
8012
8013 /* Worry about defaulted signals */
8014 if(attrp->posix_attr_flags & POSIX_SPAWN_SETSIGDEF)
8015 {
8016 struct sigaction deflt;
8017 sigset_t all_signals;
8018
8019 int s;
8020
8021 /* Construct default signal action */
8022 deflt.sa_handler = SIG_DFL;
8023 deflt.sa_flags = 0;
8024
8025 /* Construct the set of all signals */
8026 sigfillset(&all_signals);
8027
8028 /* Loop for all signals */
8029 for(s=0; sigismember(&all_signals,s); s++)
8030 {
8031 /* Signal to be defaulted? */
```

```

8025 if(sigismember(&attrp->posix_attr_sigdefault,s))
8026 {
8027 /* Yes; default this signal */
8028 if(sigaction(s, &deflt, NULL) == -1)
8029 {
8030 /* Failed */
8031 exit(127);
8032 }
8033 }
8034 }
8035 }

8036 /* Worry about the fds if we are to map them */
8037 if(file_actions != NULL)
8038 {
8039 /* Loop for all actions in object file_actions */
8040 /*(implementation dives beneath abstraction)*/
8041 char *p = *file_actions;
8042 while(*p != ' ')
8043 {
8044 if(strncmp(p,"close(",6) == 0)
8045 {
8046 int fd;
8047 if(sscanf(p+6,"%d",&fd) != 1)
8048 {
8049 exit(127);
8050 }
8051 if(close(fd) == -1) exit(127);
8052 }
8053 else if(strncmp(p,"dup2(",5) == 0)
8054 {
8055 int fd,newfd;
8056 if(sscanf(p+5,"%d,%d",&fd,&newfd) != 2)
8057 {
8058 exit(127);
8059 }
8060 if(dup2(fd, newfd) == -1) exit(127);
8061 }
8062 else if(strncmp(p,"open(",5) == 0)
8063 {
8064 int fd,oflag;
8065 mode_t mode;
8066 int tempfd;
8067 char path[1000]; /* Should be dynamic */
8068 char *q;
8069 if(sscanf(p+5,"%d",&fd) != 1)
8070 {
8071 exit(127);
8072 }
8073 p = strchr(p, ' ') + 1;
8074 q = strchr(p, '*');
8075 if(q == NULL) exit(127);
8076 strncpy(path, p, q-p);

```

```

8077 path[q-p] = ' ';
8078 if(sscanf(q+1,"%o,%o",&oflag,&mode)!=2)
8079 {
8080 exit(127);
8081 }
8082 if(close(fd) == -1)
8083 {
8084 if(errno != EBADF) exit(127);
8085 }
8086 tempfd = open(path, oflag, mode);
8087 if(tempfd == -1) exit(127);
8088 if(tempfd != fd)
8089 {
8090 if(dup2(tempfd,fd) == -1)
8091 {
8092 exit(127);
8093 }
8094 if(close(tempfd) == -1)
8095 {
8096 exit(127);
8097 }
8098 }
8099 }
8100 else
8101 {
8102 exit(127);
8103 }
8104 p = strchr(p, ')') + 1;
8105 }
8106 }

8107 /* Worry about setting new scheduling policy and parameters */
8108 if(attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDULER)
8109 {
8110 if(sched_setscheduler(0, attrp->posix_attr_schedpolicy,
8111 &attrp->posix_attr_schedparam) == -1)
8112 {
8113 exit(127);
8114 }
8115 }

8116 /* Worry about setting only new scheduling parameters */
8117 if(attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDPARAM)
8118 {
8119 if(sched_setparam(0, &attrp->posix_attr_schedparam)==-1)
8120 {
8121 exit(127);
8122 }
8123 }

8124 /* Now execute the program at path */
8125 /* Any fd that still has FD_CLOEXEC set will be closed */
8126 execve(path, argv, envp);
8127 exit(127); /* exec failed */

```

```

8128 }
8129 else
8130 {
8131 /* This is the parent (calling) process */
8132 if(*pid == (pid_t)-1) return errno;
8133 return 0;
8134 }
8135 }

8136 /*****
8137 /* Here is a crude but effective implementation of the */
8138 /* file action object operators which store actions as */
8139 /* concatenated token separated strings. */
8140 /*****
8141 /* Create object with no actions. */
8142 int posix_spawn_file_actions_init(
8143 posix_spawn_file_actions_t *file_actions)
8144 {
8145 *file_actions = malloc(sizeof(char));
8146 if(*file_actions == NULL) return ENOMEM;
8147 strcpy(*file_actions, "");
8148 return 0;
8149 }

8150 /* Free object storage and make invalid. */
8151 int posix_spawn_file_actions_destroy(
8152 posix_spawn_file_actions_t *file_actions)
8153 {
8154 free(*file_actions);
8155 *file_actions = NULL;
8156 return 0;
8157 }

8158 /* Add a new action string to object. */
8159 static int add_to_file_actions(
8160 posix_spawn_file_actions_t *file_actions,
8161 char *new_action)
8162 {
8163 *file_actions = realloc
8164 (*file_actions, strlen(*file_actions)+strlen(new_action)+1);
8165 if(*file_actions == NULL) return ENOMEM;
8166 strcat(*file_actions, new_action);
8167 return 0;
8168 }

8169 /* Add a close action to object. */
8170 int posix_spawn_file_actions_addclose(
8171 posix_spawn_file_actions_t *file_actions, int fildes)
8172 {
8173 char temp[100];
8174 sprintf(temp, "close(%d)", fildes);
8175 return add_to_file_actions(file_actions, temp);
8176 }

```

```

8177 /* Add a dup2 action to object. */
8178 int posix_spawn_file_actions_adddup2(
8179 posix_spawn_file_actions_t *file_actions, int fildes,
8180 int newfildes)
8181 {
8182 char temp[100];
8183 sprintf(temp, "dup2(%d,%d)", fildes, newfildes);
8184 return add_to_file_actions(file_actions, temp);
8185 }

8186 /* Add an open action to object. */
8187 int posix_spawn_file_actions_addopen(
8188 posix_spawn_file_actions_t *file_actions, int fildes,
8189 const char *path, int oflag, mode_t mode)
8190 {
8191 char temp[100];
8192 sprintf(temp, "open(%d,%s*%o,%o)", fildes, path, oflag, mode);
8193 return add_to_file_actions(file_actions, temp);
8194 }

8195 /*****
8196 /* Here is a crude but effective implementation of the */
8197 /* spawn attributes object functions which manipulate */
8198 /* the individual attributes. */
8199 /*****
8200 /* Initialize object with default values. */
8201 int posix_spawnattr_init(
8202 posix_spawnattr_t *attr)
8203 {
8204 attr->posix_attr_flags=0;
8205 attr->posix_attr_pgroup=0;
8206 /* Default value of signal mask is the parent's signal mask; */
8207 /* other values are also allowed */
8208 sigprocmask(0,NULL,&attr->posix_attr_sigmask);
8209 sigemptyset(&attr->posix_attr_sigdefault);
8210 /* Default values of scheduling attr inherited from the parent; */
8211 /* other values are also allowed */
8212 attr->posix_attr_schedpolicy=sched_getscheduler(0);
8213 sched_getparam(0,&attr->posix_attr_schedparam);
8214 return 0;
8215 }

8216 int posix_spawnattr_destroy(posix_spawnattr_t *attr)
8217 {
8218 /* No action needed */
8219 return 0;
8220 }

8221 int posix_spawnattr_getflags(const posix_spawnattr_t *attr,
8222 short *flags)
8223 {
8224 *flags=attr->posix_attr_flags;
8225 return 0;
8226 }

```



```
8227 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags)
8228 {
8229 attr->posix_attr_flags=flags;
8230 return 0;
8231 }

8232 int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,
8233 pid_t *pgroup)
8234 {
8235 *pgroup=attr->posix_attr_pgroup;
8236 return 0;
8237 }

8238 int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup)
8239 {
8240 attr->posix_attr_pgroup=pgroup;
8241 return 0;
8242 }

8243 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,
8244 int *schedpolicy)
8245 {
8246 *schedpolicy=attr->posix_attr_schedpolicy;
8247 return 0;
8248 }

8249 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
8250 int schedpolicy)
8251 {
8252 attr->posix_attr_schedpolicy=schedpolicy;
8253 return 0;
8254 }

8255 int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
8256 struct sched_param *schedparam)
8257 {
8258 *schedparam=attr->posix_attr_schedparam;
8259 return 0;
8260 }

8261 int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
8262 const struct sched_param *schedparam)
8263 {
8264 attr->posix_attr_schedparam=*schedparam;
8265 return 0;
8266 }

8267 int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
8268 sigset_t *sigmask)
8269 {
8270 *sigmask=attr->posix_attr_sigmask;
8271 return 0;
8272 }

8273 int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
8274 const sigset_t *sigmask)
```

```

8275 {
8276 attr->posix_attr_sigmask=*sigmask;
8277 return 0;
8278 }

8279 int posix_spawnattr_getsigdefault(const posix_spawnattr_t *attr,
8280 sigset_t *sigdefault)
8281 {
8282 *sigdefault=attr->posix_attr_sigdefault;
8283 return 0;
8284 }

8285 int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
8286 const sigset_t *sigdefault)
8287 {
8288 attr->posix_attr_sigdefault=*sigdefault;
8289 return 0;
8290 }

```

### 8291 **I/O Redirection with Spawn**

8292 I/O redirection with *posix\_spawn()* or *posix\_spawnp()* is accomplished by crafting a *file\_actions*  
8293 argument to effect the desired redirection. Such a redirection follows the general outline of the  
8294 following example:

```

8295 /* To redirect new standard output (fd 1) to a file, */
8296 /* and redirect new standard input (fd 0) from my fd socket_pair[1], */
8297 /* and close my fd socket_pair[0] in the new process. */
8298 posix_spawn_file_actions_t file_actions;
8299 posix_spawn_file_actions_init(&file_actions);
8300 posix_spawn_file_actions_addopen(&file_actions, 1, "newout", ...);
8301 posix_spawn_file_actions_dup2(&file_actions, socket_pair[1], 0);
8302 posix_spawn_file_actions_close(&file_actions, socket_pair[0]);
8303 posix_spawn_file_actions_close(&file_actions, socket_pair[1]);
8304 posix_spawn(..., &file_actions, ...);
8305 posix_spawn_file_actions_destroy(&file_actions);

```

### 8306 **Spawning a Process Under a New User ID**

8307 Spawning a process under a new user ID follows the outline shown in the following example:

```

8308 Save = getuid();
8309 setuid(newid);
8310 posix_spawn(...);
8311 setuid(Save);

```

8312 / *Rationale (Informative)*

8313 **Part C:**

8314 **Shell and Utilities**

8315 *The Open Group*



# Rationale for Shell and Utilities

8316

## 8317 **C.1 Introduction**

### 8318 **C.1.1 Scope**

8319 Refer to Section A.1.1 (on page 3311).

### 8320 **C.1.2 Conformance**

8321 Refer to Section A.2 (on page 3317).

### 8322 **C.1.3 Normative References**

8323 There is no additional rationale provided for this section.

### 8324 **C.1.4 Changes from Issue 4**

8325 The change history is provided as an informative section, to track changes from previous issues  
8326 of IEEE Std. 1003.1-200x that comprised earlier versions of the Single UNIX Specification.

#### 8327 *C.1.4.1 Changes from Issue 4 to Issue 4, Version 2*

8328 There is no additional rationale provided for this section.

#### 8329 *C.1.4.2 Changes from Issue 4, Version 2 to Issue 5*

8330 There is no additional rationale provided for this section.

#### 8331 *C.1.4.3 Changes from Issue 5 to Issue 6*

8332 There is no additional rationale provided for this section.

### 8333 **C.1.5 Terminology**

8334 Refer to Section A.1.4 (on page 3313).

### 8335 **C.1.6 Definitions**

8336 Refer to Section A.3 (on page 3321).

### 8337 **C.1.7 Relationship to Other Documents**

#### 8338 *C.1.7.1 System Interfaces*

8339 It has been pointed out that the Shell and Utilities volume of IEEE Std. 1003.1-200x assumes that  
8340 a great deal of functionality from the System Interfaces volume of IEEE Std. 1003.1-200x is  
8341 present, but never states exactly how much (and strictly does not need to since both are  
8342 mandated on a conforming system). This section is an attempt to clarify the assumptions.

**8343 C.1.8 Portability**

8344 Refer to Section A.1.5 (on page 3315).

**8345 C.1.8.1 Codes**

8346 Refer to Section A.1.5.1 (on page 3315).

**8347 C.1.9 Utility Limits**

8348 This section grew out of an idea that originated with the original POSIX.1, in the tables of system  
8349 limits for the *sysconf()* and *pathconf()* functions. The idea being that a conforming application  
8350 can be written to use the most restrictive values that a minimal system can provide, but it should  
8351 not have to. The values provided represent compromises so that some vendors can use  
8352 historically limited versions of UNIX system utilities. They are the highest values that a strictly  
8353 conforming application can assume, given no other information.

8354 However, by using the *getconf* utility or the *sysconf()* function, the elegant application can be  
8355 tailored to more liberal values on some of the specific instances of specific implementations.

8356 There is no explicitly stated requirement that an implementation provide finite limits for any of  
8357 these numeric values; the implementation is free to provide essentially unbounded capabilities  
8358 (where it makes sense), stopping only at reasonable points such as {ULONG\_MAX} (from the  
8359 ISO C standard). Therefore, applications desiring to tailor themselves to the values on a  
8360 particular implementation need to be ready for possibly huge values; it may not be a good idea  
8361 to allocate blindly a buffer for an input line based on the value of {LINE\_MAX}, for instance.  
8362 However, unlike the System Interfaces volume of IEEE Std. 1003.1-200x, there is no set of limits  
8363 that return a special indication meaning “unbounded”. The implementation should always  
8364 return an actual number, even if the number is very large.

8365 The statement:

8366 “It is not guaranteed that the application ...”

8367 is an indication that many of these limits are designed to ensure that implementors design their  
8368 utilities without arbitrary constraints related to unimaginative programming. There are certainly  
8369 conditions under which combinations of options can cause failures that would not render an  
8370 implementation non-conforming. For example, {EXPR\_NEST\_MAX} and {ARG\_MAX} could  
8371 collide when expressions are large; combinations of {BC\_SCALE\_MAX} and {BC\_DIM\_MAX}  
8372 could exceed virtual memory.

8373 In the Shell and Utilities volume of IEEE Std. 1003.1-200x, the notion of a limit being guaranteed  
8374 for the process lifetime, as it is in the System Interfaces volume of IEEE Std. 1003.1-200x, is not as  
8375 useful to a shell script. The *getconf* utility is probably a process itself, so the guarantee would be  
8376 without value. Therefore, the Shell and Utilities volume of IEEE Std. 1003.1-200x requires the  
8377 guarantee to be for the session lifetime. This will mean that many vendors will either return very  
8378 conservative values or possibly implement *getconf* as a built-in.

8379 It may seem confusing to have limits that apply only to a single utility grouped into one global  
8380 section. However, the alternative, which would be to disperse them out into their utility  
8381 description sections, would cause great difficulty when *sysconf()* and *getconf* were described.  
8382 Therefore, the standard developers chose the global approach.

8383 Each language binding could provide symbol names that are slightly different than are shown  
8384 here. For example, the C-Language Binding option adds a leading underscore to the symbols as a  
8385 prefix.

8386 The following comments describe selection criteria for the symbols and their values:

8387 {ARG\_MAX}

8388 This is defined by the System Interfaces volume of IEEE Std. 1003.1-200x. Unfortunately, it

8389 is very difficult for a portable application to deal with this value, as it does not know how

8390 much of its argument space is being consumed by the environment variables of the user.

8391 {BC\_BASE\_MAX}

8392 {BC\_DIM\_MAX}

8393 {BC\_SCALE\_MAX}

8394 These were originally one value, {BC\_SCALE\_MAX}, but it was unreasonable to link all

8395 three concepts into one limit.

8396 {CHILD\_MAX}

8397 This is defined by the System Interfaces volume of IEEE Std. 1003.1-200x.

8398 {COLL\_WEIGHTS\_MAX}

8399 The weights assigned to **order** can be considered as “passes” through the collation

8400 algorithm.

8401 {EXPR\_NEST\_MAX}

8402 The value for expression nesting was borrowed from the ISO C standard.

8403 {LINE\_MAX}

8404 This is a global limit that affects all utilities, unless otherwise noted. The {MAX\_CANON}

8405 value from the System Interfaces volume of IEEE Std. 1003.1-200x may further limit input

8406 lines from terminals. The {LINE\_MAX} value was the subject of much debate and is a

8407 compromise between those who wished to have unlimited lines and those who understood

8408 that many historical utilities were written with fixed buffers. Frequently, utility writers

8409 selected the UNIX system constant {BUFSIZ} to allocate these buffers; therefore, some

8410 utilities were limited to 512 bytes for I/O lines, while others achieved 4 096 bytes or greater.

8411 It should be noted that {LINE\_MAX} applies only to input line length; there is no

8412 requirement in IEEE Std. 1003.1-200x that limits the length of output lines. Utilities such as

8413 *awk*, *sed*, and *paste* could theoretically construct lines longer than any of the input lines they

8414 received, depending on the options used or the instructions from the application. They are

8415 not required to truncate their output to {LINE\_MAX}. It is the responsibility of the

8416 application to deal with this. If the output of one of those utilities is to be piped into another

8417 of the standard utilities, line length restrictions will have to be considered; the *fold* utility,

8418 among others, could be used to ensure that only reasonable line lengths reach utilities or

8419 applications.

8420 {LINK\_MAX}

8421 This is defined by the System Interfaces volume of IEEE Std. 1003.1-200x.

8422 {MAX\_CANON}

8423 {MAX\_INPUT}

8424 {NAME\_MAX}

8425 {NGROUPS\_MAX}

8426 {OPEN\_MAX}

8427 {PATH\_MAX}

8428 {PIPE\_BUF}

8429 These limits are defined by the System Interfaces volume of IEEE Std. 1003.1-200x. Note that

8430 the byte lengths described by some of these values continue to represent bytes, even if the

8431 applicable character set uses a multi-byte encoding.

8432 {RE\_DUP\_MAX}  
 8433 The value selected is consistent with historical practice. Although the name implies that it  
 8434 applies to all REs, only BREs use the interval notation `\{m,n\}` addressed by this limit.

8435 {POSIX2\_SYMLINKS}  
 8436 The {POSIX2\_SYMLINKS} variable indicates that the underlying operating system supports  
 8437 the creation of symbolic links in specific directories. Many of the utilities defined in  
 8438 IEEE Std. 1003.1-200x that deal with symbolic links do not depend on this value. For  
 8439 example, a utility that follows symbolic links (or does not, as the case may be) will only be  
 8440 affected by a symbolic link if it encounters one. Presumably, a file system that does not  
 8441 support symbolic links will not contain any. This variable does affect such utilities as `ln -s`  
 8442 and `pax` that attempt to create symbolic links.

8443 {POSIX2\_SYMLINKS} was developed even though there is no comparable configuration  
 8444 value in the IEEE P1003.1a draft standard.

8445 There are different limits associated with command lines and input to utilities, depending on the  
 8446 method of invocation. In the case of a C program *exec*-ing a utility, {ARG\_MAX} is the  
 8447 underlying limit. In the case of the shell reading a script and *exec*-ing a utility, {LINE\_MAX}  
 8448 limits the length of lines the shell is required to process, and {ARG\_MAX} will still be a limit. If a  
 8449 user is entering a command on a terminal to the shell, requesting that it invoke the utility,  
 8450 {MAX\_INPUT} may restrict the length of the line that can be given to the shell to a value below  
 8451 {LINE\_MAX}.

8452 When an option is supported, *getconf* returns a value of 1. For example, when C development is  
 8453 supported:

```
8454 if ["$(getconf POSIX2_C_DEV)" -eq 1]; then
8455 echo C supported
8456 fi
```

8457 The *sysconf()* function in the C-Language Binding option would return 1.

8458 The following comments describe selection criteria for the symbols and their values:

8459 POSIX2\_C\_BIND  
 8460 POSIX2\_C\_DEV  
 8461 POSIX2\_FORT\_DEV  
 8462 POSIX2\_FORT\_RUN  
 8463 POSIX2\_SW\_DEV  
 8464 POSIX2\_UPE

8465 It is possible for some (usually privileged) operations to remove utilities that support these  
 8466 options or otherwise to render these options unsupported. The header files, the *sysconf()*  
 8467 function, or the *getconf* utility will not necessarily detect such actions, in which case they  
 8468 should not be considered as rendering the implementation non-conforming. A test suite  
 8469 should not attempt tests such as:

```
8470 rm /usr/bin/c89
8471 getconf POSIX2_C_DEV
```

8472 POSIX2\_LOCALEDEF  
 8473 This symbol was introduced to allow implementations to restrict supported locales to only  
 8474 those supplied by the implementation.



8475 **C.1.10 Grammar Conventions**

8476 There is no additional rationale for this section.

8477 **C.1.11 Utility Description Defaults**8478 This section is arranged with headings in the same order as all the utility descriptions. It is a  
8479 collection of related and unrelated information concerning

8480 1. The default actions of utilities

8481 2. The meanings of notations used in IEEE Std. 1003.1-200x that are specific to individual  
8482 utility sections8483 Although this material may seem out of place here, it is important that this information appear  
8484 before any of the utilities to be described later.8485 **NAME**

8486 There is no additional rationale provided for this section.

8487 **SYNOPSIS**

8488 There is no additional rationale provided for this section.

8489 **DESCRIPTION**

8490 There is no additional rationale provided for this section.

8491 **OPTIONS**8492 Although it has not always been possible, the standard developers tried to avoid repeating  
8493 information to reduce the risk that duplicate explanations could each be modified differently.8494 The need to recognize `--` is required because portable applications need to shield their operands  
8495 from any arbitrary options that the implementation may provide as an extension. For example, if  
8496 the standard utility *foo* is listed as taking no options, and the application needed to give it a path  
8497 name with a leading hyphen, it could safely do it as:8498 `foo -- -myfile`8499 and avoid any problems with `-m` used as an extension.8500 **OPERANDS**8501 The usage of `-` is never shown in the SYNOPSIS. Similarly, the usage of `--` is never shown.8502 The requirement for processing operands in command-line order is to avoid a “WeirdNIX”  
8503 utility that might choose to sort the input files alphabetically, by size, or by directory order.  
8504 Although this might be acceptable for some utilities, in general the programmer has a right to  
8505 know exactly what order will be chosen.8506 Some of the standard utilities take multiple *file* operands and act as if they were processing the  
8507 concatenation of those files. For example:8508 `asa file1 file2`

8509 and:

8510 `cat file1 file2 | asa`

8511 have similar results when questions of file access, errors, and performance are ignored. Other  
8512 utilities such as *grep* or *wc* have completely different results in these two cases. This latter type of  
8513 utility is always identified in its DESCRIPTION or OPERANDS sections, whereas the former is  
8514 not. Although it might be possible to create a general assertion about the former case, the  
8515 following points must be addressed:

- 8516 • Access times for the files might be different in the operand case *versus* the *cat* case.
- 8517 • The utility may have error messages that are cognizant of the input file name, and this added  
8518 value should not be suppressed. (As an example, *awk* sets a variable with the file name at  
8519 each file boundary.)

## 8520 STDIN

8521 There is no additional rationale provided for this section.

## 8522 INPUT FILES

8523 A conforming application cannot assume the following three commands are equivalent:

```
8524 tail -n +2 file
8525 (sed -n 1q; cat) < file
8526 cat file | (sed -n 1q; cat)
```

8527 The second command is equivalent to the first only when the file is seekable. In the third  
8528 command, if the file offset in the open file description were not unspecified, *sed* would have to be  
8529 implemented so that it read from the pipe 1 byte at a time or it would have to employ some  
8530 method to seek backwards on the pipe. Such functionality is not defined currently in POSIX.1  
8531 and does not exist on all historical systems. Other utilities, such as *head*, *read*, and *sh*, have similar  
8532 properties, so the restriction is described globally in this section.

8533 The definition of *text file* is strictly enforced for input to the standard utilities; very few of them  
8534 list exceptions to the undefined results called for here. (Of course, “undefined” here does not  
8535 mean that historical implementations necessarily have to change to start indicating error  
8536 conditions. Conforming applications cannot rely on implementations succeeding or failing when  
8537 non-text files are used.)

8538 The utilities that allow line continuation are generally those that accept input languages, rather  
8539 than pure data. It would be unusual for an input line of this type to exceed {LINE\_MAX} bytes  
8540 and unreasonable to require that the implementation allow unlimited accumulation of multiple  
8541 lines, each of which could reach {LINE\_MAX}. Thus, for a portable application the total of all  
8542 the continued lines in a set cannot exceed {LINE\_MAX}.

8543 The format description is intended to be sufficiently rigorous to allow other applications to  
8544 generate these input files. However, since <blank>s can legitimately be included in some of the  
8545 fields described by the standard utilities, particularly in locales other than the POSIX locale, this  
8546 intent is not always realized.

## 8547 ENVIRONMENT VARIABLES

8548 There is no additional rationale provided for this section.

8549 **ASYNCHRONOUS EVENTS**

8550 Because there is no language prohibiting it, a utility is permitted to catch a signal, perform some  
8551 additional processing (such as deleting temporary files), restore the default signal action (or  
8552 action inherited from the parent process), and resignal itself.

8553 **STDOUT**

8554 The format description is intended to be sufficiently rigorous to allow post-processing of output  
8555 by other programs, particularly by an *awk* or *lex* parser.

8556 **STDERR**

8557 This section does not describe error messages that refer to incorrect operation of the utility.  
8558 Consider a utility that processes program source code as its input. This section is used to  
8559 describe messages produced by a correctly operating utility that encounters an error in the  
8560 program source code on which it is processing. However, a message indicating that the utility  
8561 had insufficient memory in which to operate would not be described.

8562 Some utilities have traditionally produced warning messages without returning a non-zero exit  
8563 status; these are specifically noted in their sections. Other utilities shall not write to standard  
8564 error if they complete successfully, unless the implementation provides some sort of extension  
8565 to increase the verbosity or debugging level.

8566 The format descriptions are intended to be sufficiently rigorous to allow post-processing of  
8567 output by other programs.

8568 **OUTPUT FILES**

8569 The format description is intended to be sufficiently rigorous to allow post-processing of output  
8570 by other programs, particularly by an *awk* or *lex* parser.

8571 Receipt of the SIGQUIT signal should generally cause termination (unless in some debugging  
8572 mode) that would bypass any attempted recovery actions.

8573 **EXTENDED DESCRIPTION**

8574 There is no additional rationale provided for this section.

8575 **EXIT STATUS**

8576 Note the additional discussion of exit values in *Exit Status for Commands* in the *sh* utility. It  
8577 describes requirements for returning exit values greater than 125.

8578 A utility may list zero as a successful return, 1 as a failure for a specific reason, and greater than  
8579 1 as “an error occurred”. In this case, unspecified conditions may cause a 2 or 3, or other value,  
8580 to be returned. A strictly conforming application should be written so that it tests for successful  
8581 exit status values (zero in this case), rather than relying upon the single specific error value listed  
8582 in IEEE Std. 1003.1-200x. In that way, it will have maximum portability, even on  
8583 implementations with extensions.

8584 The standard developers are aware that the general non-enumeration of errors makes it difficult  
8585 to write test suites that test the *incorrect* operation of utilities. There are some historical  
8586 implementations that have expended effort to provide detailed status messages and a helpful  
8587 environment to bypass or explain errors, such as prompting, retrying, or ignoring unimportant  
8588 syntax errors; other implementations have not. Since there is no realistic way to mandate system  
8589 behavior in cases of undefined application actions or system problems—in a manner acceptable  
8590 to all cultures and environments—attention has been limited to the correct operation of utilities

8591 by the conforming application. Furthermore, the portable application does not need detailed  
8592 information concerning errors that it caused through incorrect usage or that it cannot correct.

8593 There is no description of defaults for this section because all of the standard utilities specify  
8594 something (or explicitly state “Unspecified”) for exit status.

#### 8595 **CONSEQUENCES OF ERRORS**

8596 Several actions are possible when a utility encounters an error condition, depending on the  
8597 severity of the error and the state of the utility. Included in the possible actions of various  
8598 utilities are: deletion of temporary or intermediate work files; deletion of incomplete files; and  
8599 validity checking of the file system or directory.

8600 The text about recursive traversing is meant to ensure that utilities such as *find* process as many  
8601 files in the hierarchy as they can. They should not abandon all of the hierarchy at the first error  
8602 and resume with the next command-line operand, but should attempt to keep going.

#### 8603 **APPLICATION USAGE**

8604 This section provides additional caveats, issues, and recommendations to the developer.

#### 8605 **EXAMPLES**

8606 This section provides sample usage.

#### 8607 **RATIONALE**

8608 There is no additional rationale provided for this section.

#### 8609 **FUTURE DIRECTIONS**

8610 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
8611 the future, and often cautions the developer to architect the code to account for a change in this  
8612 area. Note that a future directions statement should not be taken as a commitment to adopt a  
8613 feature or interface in the future.

#### 8614 **SEE ALSO**

8615 There is no additional rationale provided for this section.

#### 8616 **CHANGE HISTORY**

8617 There is no additional rationale provided for this section.

### 8618 **C.1.12 Considerations for Utilities in Support of Files of Arbitrary Size**

8619 This section is intended to clarify the requirements for utilities in support of large files.

8620 The utilities listed in this section are utilities which are used to perform administrative tasks  
8621 such as to create, move, copy, remove, change the permissions, or measure the resources of a  
8622 file. They are useful both as end-user tools and as utilities invoked by applications during  
8623 software installation and operation.

8624 The *chgrp*, *chmod*, *chown*, *ln*, and *rm* utilities probably require use of large file capable versions of  
8625 *stat()*, *lstat()*, *ftw()*, and the **stat** structure.

8626 The *cat*, *cksum*, *cmp*, *cp*, *dd*, *mv*, *sum*, and *touch* utilities probably require use of large file capable  
8627 versions of *creat()*, *open()*, and *fopen()*.

- 8628           The *cat*, *cksum*, *cmp*, *dd*, *df*, *du*, *ls*, and *sum* utilities may require writing large integer values. For  
8629           example:
- 8630           • The *cat* utility might have a `-n` option which counts `<newline>`s.
  - 8631           • The *cksum* and *ls* utilities report file sizes.
  - 8632           • The *cmp* utility reports the line number at which the first difference occurs, and also has a `-l`  
8633           option which reports file offsets.
  - 8634           • The *dd*, *df*, *du*, *ls*, and *sum* utilities report block counts.
- 8635           The *dd*, *find*, and *test* utilities may need to interpret command arguments that contain 64-bit  
8636           values. For *dd*, the arguments include `skip=n`, `seek=n`, and `count=n`. For *find*, the arguments  
8637           include `-size n`. For *test*, the arguments are those associated with algebraic comparisons.
- 8638           The *df* utility might need to access large file systems with `statvfs()`.
- 8639           The *ulimit* utility will need to use large file capable versions of `getrlimit()` and `setrlimit()` and be  
8640           able to read and write large integer values.

8641 **C.2 Shell Command Language**8642 **C.2.1 Shell Introduction**

8643 The System V shell was selected as the starting point for the Shell and Utilities volume of  
 8644 IEEE Std. 1003.1-200x. The BSD C shell was excluded from consideration for the following  
 8645 reasons:

- 8646 • Most historically portable shell scripts assume the Version 7 Bourne shell, from which the  
 8647 System V shell is derived.
- 8648 • The majority of tutorial materials on shell programming assume the System V shell.

8649 The construct "#!" is reserved for implementations wishing to provide that extension. If it were  
 8650 not reserved, the Shell and Utilities volume of IEEE Std. 1003.1-200x would disallow it by forcing  
 8651 it to be a comment. As it stands, a POSIX-conforming application must not use "#!" as the first  
 8652 two characters of the file. An XSI-conforming application can use the construct "#!", since on  
 8653 XSI-conformant systems this is defined to denote an executable script, which matches historical  
 8654 practice. Invention of new meanings or extensions to the "#!" construct were rejected since they  
 8655 are beyond the scope of IEEE Std. 1003.1-200x.

8656 **C.2.2 Quoting**

8657 There is no additional rationale for this section.

8658 *C.2.2.1 Escape Character (Backslash)*

8659 There is no additional rationale for this section.

8660 *C.2.2.2 Single-Quotes*

8661 A backslash cannot be used to escape a single-quote in a single-quoted string. An embedded  
 8662 quote can be created by writing, for example: "'a'\ 'b'", which yields "a'b". (See the Shell  
 8663 and Utilities volume of IEEE Std. 1003.1-200x, Section 2.6.5, Field Splitting for a better  
 8664 understanding of how portions of words are either split into fields or remain concatenated.) A  
 8665 single token can be made up of concatenated partial strings containing all three kinds of quoting  
 8666 or escaping, thus permitting any combination of characters.

8667 *C.2.2.3 Double-Quotes*

8668 The escaped <newline> used for line continuation is removed entirely from the input and is not  
 8669 replaced by any white space. Therefore, it cannot serve as a token separator.

8670 In double-quoting, if a backslash is immediately followed by a character that would be  
 8671 interpreted as having a special meaning, the backslash is deleted and the subsequent character is  
 8672 taken literally. If a backslash does not precede a character that would have a special meaning, it  
 8673 is left in place unmodified and the character immediately following it is also left unmodified.  
 8674 Thus, for example:

8675       "\\$" → \$

8676       "\a" → \a

8677 It would be desirable to include the statement “The characters from an enclosed "\${" to the  
 8678 matching '}' shall not be affected by the double quotes”, similar to the one for "\$()".  
 8679 However, historical practice in the System V shell prevents this.

8680 The requirement that double-quotes be matched inside "\${...}" within double-quotes and the  
 8681 rule for finding the matching '}' in the Shell and Utilities volume of IEEE Std. 1003.1-200x,  
 8682 Section 2.6.2, Parameter Expansion eliminate several subtle inconsistencies in expansion for  
 8683 historical shells in rare cases; for example:

```
8684 "${foo-bar}"
```

8685 yields **bar** when **foo** is not defined, and is an invalid substitution when **foo** is defined, in many  
 8686 historical shells. The differences in processing the "\${...}" form have led to inconsistencies  
 8687 between historical systems. A consequence of this rule is that single-quotes cannot be used to  
 8688 quote the '}' within "\${...}"; for example:

```
8689 unset bar

 8690 foo="${bar-'}'"
```

8691 is invalid because the "\${...}" substitution contains an unpaired unescaped single-quote. The  
 8692 backslash can be used to escape the '}' in this example to achieve the desired result:

```
8693 unset bar

 8694 foo="${bar-\'}'"
```

8695 The differences in processing the "\${...}" form have led to inconsistencies between the  
 8696 historical System V shell, BSD, and KornShells, and the text in the Shell and Utilities volume of  
 8697 IEEE Std. 1003.1-200x is an attempt to converge them without breaking too many applications.  
 8698 The only alternative to this compromise between shells would be to make the behavior  
 8699 unspecified whenever the literal characters ''', '{', '}', and '"' appear within "\${...}".  
 8700 To write a portable script that uses these values, a user would have to assign variables; for  
 8701 example:

```
8702 squote=\` dquote=\" lbrace='{` rbrace=}'`

 8703 ${foo-$squote$rbrace$squote}
```

8704 rather than:

```
8705 ${foo-"' }' }
```

8706 Some systems have allowed the end of the word to terminate the backquoted command  
 8707 substitution, such as in:

```
8708 "`echo hello"
```

8709 This usage is undefined; the matching backquote is required by the Shell and Utilities volume of  
 8710 IEEE Std. 1003.1-200x. The other undefined usage can be illustrated by the example:

```
8711 sh -c '` echo "foo`'
```

8712 The description of the recursive actions involving command substitution can be illustrated with  
 8713 an example. Upon recognizing the introduction of command substitution, the shell parses input  
 8714 (in a new context), gathering the source for the command substitution until an unbalanced '}'  
 8715 or '`' is located. For example, in the following:

```
8716 echo "${date; echo "

 8717 one" }"
```

8718 the double-quote following the *echo* does not terminate the first double-quote; it is part of the  
 8719 command substitution script. Similarly, in:

```
8720 echo "${echo *}"
```

8721 the asterisk is not quoted since it is inside command substitution; however:

8722           echo "\$ (echo "\*" )"

8723           is quoted (and represents the asterisk character itself).

### 8724 C.2.3 Token Recognition

8725           The "(" and ")" symbols are control operators in the KornShell, used for an alternative  
8726           syntax of an arithmetic expression command. A portable application cannot use "(" as a single  
8727           token (with the exception of the "\$ ( ( " form for shell arithmetic).

8728           The (3) rule about combining characters to form operators is not meant to preclude systems from  
8729           extending the shell language when characters are combined in otherwise invalid ways. Portable  
8730           applications cannot use invalid combinations, and test suites should not penalize systems that  
8731           take advantage of this fact. For example, the unquoted combination "|&" is not valid in a POSIX  
8732           script, but has a specific KornShell meaning.

8733           The (10) rule about '#' as the current character is the first in the sequence in which a new token  
8734           is being assembled. The '#' starts a comment only when it is at the beginning of a token. This  
8735           rule is also written to indicate that the search for the end-of-comment does not consider escaped  
8736           <newline> specially, so that a comment cannot be continued to the next line.

#### 8737 C.2.3.1 Alias Substitution

8738           The alias capability was added in the UPE because it is widely used in historical  
8739           implementations by interactive users.

8740           The definition of *alias name* precludes an alias name containing a slash character. Since the text  
8741           applies to the command words of simple commands, reserved words (in their proper places)  
8742           cannot be confused with aliases.

8743           The placement of alias substitution in token recognition makes it clear that it precedes all of the  
8744           word expansion steps.

8745           An example concerning trailing <blank> characters and reserved words follows. If the user  
8746           types:

```
8747 $ alias foo="/bin/ls "
8748 $ alias while="/"
```

8749           The effect of executing:

```
8750 $ while true
8751 > do
8752 > echo "Hello, World"
8753 > done
```

8754           is a never-ending sequence of "Hello, World" strings to the screen. However, if the user  
8755           types:

```
8756 $ foo while
```

8757           the result is an *ls* listing of /. Since the alias substitution for **foo** ends in a <space> character, the  
8758           next word is checked for alias substitution. The next word, **while**, has also been aliased, so it is  
8759           substituted as well. Since it is not in the proper position as a command word, it is not recognized  
8760           as a reserved word.

8761           If the user types:

```
8762 $ foo; while
```



8763 **while** retains its normal reserved-word properties.

#### 8764 **C.2.4 Reserved Words**

8765 All reserved words are recognized syntactically as such in the contexts described. However, note  
8766 that **in** is the only meaningful reserved word after a **case** or **for**; similarly, **in** is not meaningful as  
8767 the first word of a simple command.

8768 Reserved words are recognized only when they are delimited (that is, meet the definition of the  
8769 Base Definitions volume of IEEE Std. 1003.1-200x, Section 3.437, Word), whereas operators are  
8770 themselves delimiters. For instance, '( ' and ' ) ' are control operators, so that no <space>  
8771 character is needed in (*list*). However, '{ ' and ' } ' are reserved words in { *list*; }, so that in this  
8772 case the leading <space> character and semicolon are required.

8773 The list of unspecified reserved words is from the KornShell, so portable applications cannot use  
8774 them in places a reserved word would be recognized. This list contained **time** in early proposals,  
8775 but it was removed when the *time* utility was selected for the Shell and Utilities volume of  
8776 IEEE Std. 1003.1-200x.

8777 There was a strong argument for promoting braces to operators (instead of reserved words), so  
8778 they would be syntactically equivalent to subshell operators. Concerns about compatibility  
8779 outweighed the advantages of this approach. Nevertheless, portable applications should  
8780 consider quoting '{ ' and ' } ' when they represent themselves.

8781 The restriction on ending a name with a colon is to allow future implementations that support  
8782 named labels for flow control; see the RATIONALE for the *break* built-in utility .

8783 It is possible that a future version of the Shell and Utilities volume of IEEE Std. 1003.1-200x may  
8784 require that '{ ' and ' } ' be treated individually as control operators, although the token "{ }"  
8785 will probably be a special-case exemption from this because of the often-used *find*{ } construct.

#### 8786 **C.2.5 Parameters and Variables**

##### 8787 *C.2.5.1 Positional Parameters*

8788 There is no additional rationale for this section.

##### 8789 *C.2.5.2 Special Parameters*

8790 Most historical implementations implement subshells by forking; thus, the special parameter  
8791 '\$ ' does not necessarily represent the process ID of the shell process executing the commands  
8792 since the subshell execution environment preserves the value of '\$ '.

8793 If a subshell were to execute a background command, the value of "\$ ! " for the parent would  
8794 not change. For example:

```
8795 (
8796 date &
8797 echo $!
8798)
8799 echo $!
```

8800 would echo two different values for "\$ ! " .

8801 The "\$ - " special parameter can be used to save and restore *set* options:

```

8802 Save=$(echo $- | sed 's/[ics]//g')
8803 ...
8804 set +aCefnuvx
8805 if [-n "$Save"]; then
8806 set -$Save
8807 fi

```

8808 The three options are removed using *sed* in the example because they may appear in the value of  
 8809 "\$-" (from the *sh* command line), but are not valid options to *set*.

8810 The descriptions of parameters '\*' and '@' assume the reader is familiar with the field splitting  
 8811 discussion in the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.6.5, Field Splitting  
 8812 and understands that portions of the word remain concatenated unless there is some reason to  
 8813 split them into separate fields.

8814 Some examples of the '\*' and '@' properties, including the concatenation aspects:

```

8815 set "abc" "def ghi" "jkl"
8816 echo $* => "abc" "def" "ghi" "jkl"
8817 echo "$*" => "abc def ghi jkl"
8818 echo $@ => "abc" "def" "ghi" "jkl"

```

8819 but:

```

8820 echo "$@" => "abc" "def ghi" "jkl"
8821 echo "xx$@yy" => "xxabc" "def ghi" "jkl"yy"
8822 echo "$@$@" => "abc" "def ghi" "jklabc" "def ghi" "jkl"

```

8823 In the preceding examples, the double-quote characters that appear after the "=>" do not appear  
 8824 in the output and are used only to illustrate word boundaries.

8825 The following example illustrates the effect of setting *IFS* to a null string:

```

8826 $ IFS=''
8827 $ set foo bar bam
8828 $ echo "$@"
8829 foo bar bam
8830 $ echo "$*"
8831 foobarbam
8832 $ unset IFS
8833 $ echo "$*"
8834 foo bar bam

```

### 8835 C.2.5.3 Shell Variables

8836 See the discussion of *IFS* in Section C.2.6.5 (on page 3532).

8837 The prohibition on *LC\_CTYPE* changes affecting lexical processing protects the shell  
 8838 implementor (and the shell programmer) from the ill effects of changing the definition of  
 8839 <blank> or the set of alphabetic characters in the current environment. It would probably not be  
 8840 feasible to write a compiled version of a shell script without this rule. The rule applies only to  
 8841 the current invocation of the shell and its subshells—invoking a shell script or performing *exec sh*  
 8842 would subject the new shell to the changes in *LC\_CTYPE*.

8843 Other common environment variables used by historical shells are not specified by the Shell and  
 8844 Utilities volume of IEEE Std. 1003.1-200x, but they should be reserved for the historical uses.

|      |                |                                                                                                         |
|------|----------------|---------------------------------------------------------------------------------------------------------|
| 8845 |                | Tilde expansion for components of the <i>PATH</i> in an assignment such as:                             |
| 8846 |                | <code>PATH=~hlj/bin:~dwc/bin:\$PATH</code>                                                              |
| 8847 |                | is a feature of some historical shells and is allowed by the wording of the Shell and Utilities         |
| 8848 |                | volume of IEEE Std. 1003.1-200x, Section 2.6.1, Tilde Expansion. Note that the tildes are               |
| 8849 |                | expanded during the assignment to <i>PATH</i> , not when <i>PATH</i> is accessed during command search. |
| 8850 |                | The following entries represent additional information about variables included in the Shell and        |
| 8851 |                | Utilities volume of IEEE Std. 1003.1-200x, or rationale for common variables in use by shells that      |
| 8852 |                | have been excluded:                                                                                     |
| 8853 | —              | (Underscore.) While underscore is historical practice, its overloaded usage in                          |
| 8854 |                | the KornShell is confusing, and it has been omitted from the Shell and Utilities                        |
| 8855 |                | volume of IEEE Std. 1003.1-200x.                                                                        |
| 8856 | <i>ENV</i>     | This variable can be used to set aliases and other items local to the invocation                        |
| 8857 |                | of a shell. The file referred to by <i>ENV</i> differs from <i>\$HOME/.profile</i> in that              |
| 8858 |                | <i>.profile</i> is typically executed at session start-up, whereas the <i>ENV</i> file is               |
| 8859 |                | executed at the beginning of each shell invocation. The <i>ENV</i> value is                             |
| 8860 |                | interpreted in a manner similar to a dot script, in that the commands are                               |
| 8861 |                | executed in the current environment and the file needs to be readable, but not                          |
| 8862 |                | executable. However, unlike dot scripts, no <i>PATH</i> searching is performed.                         |
| 8863 |                | This is used as a guard against Trojan Horse security breaches.                                         |
| 8864 | <i>ERRNO</i>   | This variable was omitted from the Shell and Utilities volume of                                        |
| 8865 |                | IEEE Std. 1003.1-200x because the values of error numbers are not defined in                            |
| 8866 |                | IEEE Std. 1003.1-200x in a portable manner.                                                             |
| 8867 | <i>FCEDIT</i>  | Since this variable affects only the <i>fc</i> utility, it has been omitted from this more              |
| 8868 |                | global place. The value of <i>FCEDIT</i> does not affect the command line editing                       |
| 8869 |                | mode in the shell; see the description of <i>set -o vi</i> in the <i>set</i> built-in utility.          |
| 8870 | <i>PS1</i>     | This variable is used for interactive prompts. Historically, the “superuser”                            |
| 8871 |                | has had a prompt of ‘#’. Since privileges are not required to be monolithic, it                         |
| 8872 |                | is difficult to define which privileges should cause the alternate prompt.                              |
| 8873 |                | However, a sufficiently powerful user should be reminded of that power by                               |
| 8874 |                | having an alternate prompt.                                                                             |
| 8875 | <i>PS3</i>     | This variable is used by the KornShell for the <i>select</i> command. Since the POSIX                   |
| 8876 |                | shell does not include <i>select</i> , <i>PS3</i> was omitted.                                          |
| 8877 | <i>PS4</i>     | This variable is used for shell debugging. For example, the following script:                           |
| 8878 |                | <code>PS4=' [ \${LINENO} ]+ '</code>                                                                    |
| 8879 |                | <code>set -x</code>                                                                                     |
| 8880 |                | <code>echo Hello</code>                                                                                 |
| 8881 |                | writes the following to standard error:                                                                 |
| 8882 |                | <code>[3]+ echo Hello</code>                                                                            |
| 8883 | <i>RANDOM</i>  | This pseudo-random number generator was not seen as being useful to                                     |
| 8884 |                | interactive users.                                                                                      |
| 8885 | <i>SECONDS</i> | Although this variable is sometimes used with <i>PS1</i> to allow the display of the                    |
| 8886 |                | current time in the prompt of the user, it is not one that would be manipulated                         |
| 8887 |                | frequently enough by an interactive user to include in the Shell and Utilities                          |
| 8888 |                | volume of IEEE Std. 1003.1-200x.                                                                        |

8889 **C.2.6 Word Expansions**

8890 Step (2) refers to the “portions of fields generated by step (1)”. For example, if the word being  
 8891 expanded were "\$x+\$y" and *IFS*=+, the word would be split only if "\$x" or "\$y" contained  
 8892 '+'; the '+' in the original word was not generated by step (1).

8893 *IFS* is used for performing field splitting on the results of parameter and command substitution;  
 8894 it is not used for splitting all fields. Previous versions of the shell used it for splitting all fields  
 8895 during field splitting, but this has severe problems because the shell can no longer parse its own  
 8896 script. There are also important security implications caused by this behavior. All useful  
 8897 applications of *IFS* use it for parsing input of the *read* utility and for splitting the results of  
 8898 parameter and command substitution.

8899 The rule concerning expansion to a single field requires that if **foo=abc** and **bar=def**, that:

8900 "\$foo" "\$bar"

8901 expands to the single field:

8902 abcdef

8903 The rule concerning empty fields can be illustrated by:

```
8904 $ unset foo
8905 $ set $foo bar ' ' xyz "$foo" abc
8906 $ for i
8907 > do
8908 > echo "-$i-"
8909 > done
8910 -bar-
8911 --
8912 -xyz-
8913 --
8914 -abc-
```

8915 Step (1) indicates that parameter expansion, command substitution, and arithmetic expansion  
 8916 are all processed simultaneously as they are scanned. For example, the following is valid  
 8917 arithmetic:

```
8918 x=1
8919 echo $(($(echo 3)+$x))
```

8920 An early proposal stated that tilde expansion preceded the other steps, but this is not the case in  
 8921 known historical implementations; if it were, and if a referenced home directory contained a '\$'  
 8922 character, expansions would result within the directory name.

8923 **C.2.6.1 Tilde Expansion**

8924 Tilde expansion generally occurs only at the beginning of words, but an exception based on  
 8925 historical practice has been included:

```
8926 PATH=/posix/bin:~djk/bin
```

8927 This is eligible for tilde expansion because tilde follows a colon and none of the relevant  
 8928 characters is quoted. Consideration was given to prohibiting this behavior because any of the  
 8929 following are reasonable substitutes:

```
8930 PATH=$(printf %s ~karels/bin : ~bostic/bin)
```

```

8931 for Dir in ~maat/bin ~srb/bin ...
8932 do
8933 PATH=${PATH:+$PATH:}$Dir
8934 done

```

8935 In the first command, explicit colons are used for each directory. In all cases, the shell performs  
 8936 tilde expansion on each directory because all are separate words to the shell.

8937 Note that expressions in operands such as:

```

8938 make -k mumble LIBDIR=~chet/lib

```

8939 do not qualify as shell variable assignments, and tilde expansion is not performed (unless the  
 8940 command does so itself, which *make* does not).

8941 Because of the requirement that the word is not quoted, the following are not equivalent; only  
 8942 the last causes tilde expansion:

```

8943 \~hlj/ ~h\lj/ ~"hlj"/ ~hlj\ / ~hlj/

```

8944 In an early proposal, tilde expansion occurred following any unquoted equals sign or colon, but  
 8945 this was removed because of its complexity and to avoid breaking commands such as:

```

8946 rcp hostname:~marc/.profile .

```

8947 A suggestion was made that the special sequence "\$~" should be allowed to force tilde  
 8948 expansion anywhere. Since this is not historical practice, it has been left for future  
 8949 implementations to evaluate. (The description in the Shell and Utilities volume of  
 8950 IEEE Std. 1003.1-200x, Section 2.2, Quoting requires that a dollar sign be quoted to represent  
 8951 itself, so the "\$~" combination is already unspecified.)

8952 The results of giving tilde with an unknown login name are undefined because the KornShell  
 8953 "~+" and "~-" constructs make use of this condition, but in general it is an error to give an  
 8954 incorrect login name with tilde. The results of having *HOME* unset are unspecified because some  
 8955 historical shells treat this as an error.

### 8956 C.2.6.2 Parameter Expansion

8957 The rule for finding the closing '}' in "\${...}" is the one used in the KornShell and is  
 8958 upwardly-compatible with the Bourne shell, which does not determine the closing '}' until the  
 8959 word is expanded. The advantage of this is that incomplete expansions, such as:

```

8960 ${foo

```

8961 can be determined during tokenization, rather than during expansion.

8962 The string length and substring capabilities were included because of the demonstrated need for  
 8963 them, based on their usage in other shells, such as C shell and KornShell.

8964 Historical versions of the KornShell have not performed tilde expansion on the word part of  
 8965 parameter expansion; however, it is more consistent to do so.

### 8966 C.2.6.3 Command Substitution

8967 The "\$()" form of command substitution solves a problem of inconsistent behavior when using  
 8968 backquotes. For example:

8969  
8970  
8971  
8972  
8973

| Command              | Output |
|----------------------|--------|
| echo '\\$x'          | \\$x   |
| echo `echo '\\$x'`   | \$x    |
| echo \$(echo '\\$x') | \\$x   |

8974  
8975  
8976  
8977

Additionally, the backquoted syntax has historical restrictions on the contents of the embedded command. While the newer "\$()" form can process any kind of valid embedded script, the backquoted form cannot handle some valid scripts that include backquotes. For example, these otherwise valid embedded scripts do not work in the left column, but do work on the right:

8978  
8979  
8980  
8981  
8982  
  
8983  
8984  
8985  
  
8986  
8987  
8988

```

echo `
cat <<\eof
a here-doc with `
eof
`

echo `
echo abc # a comment with `
`

echo `
echo ` `
`

echo $(
cat <<\eof
a here-doc with)
eof
)

echo $(
echo abc # a comment with)
)

echo $(
echo ` `
)

```

8989  
8990  
8991

Because of these inconsistent behaviors, the backquoted variety of command substitution is not recommended for new applications that nest command substitutions or attempt to embed complex scripts.

8992

The KornShell feature:

8993  
8994

If *command* is of the form *<word, word* is expanded to generate a path name, and the value of the command substitution is the contents of this file with any trailing *<newline>*s deleted.

8995  
8996  
8997

was omitted from the Shell and Utilities volume of IEEE Std. 1003.1-200x because \$(*cat word*) is an appropriate substitute. However, to prevent breaking numerous scripts relying on this feature, it is unspecified to have a script within "\$()" that has only redirections.

8998  
8999

The requirement to separate "\$(" and '((' when a single subshell is command-substituted is to avoid any ambiguities with arithmetic expansion.

#### 9000 C.2.6.4 Arithmetic Expansion

9001  
9002  
9003  
9004  
9005

The "(( ))" form of KornShell arithmetic in early proposals was omitted. The standard developers concluded that there was a strong desire for some kind of arithmetic evaluator to replace *expr*, and that relating it to '\$' makes it work well with the standard shell language, and it provides access to arithmetic evaluation in places where accessing a utility would be inconvenient.

9006  
9007  
9008  
9009  
9010  
9011  
9012  
9013  
9014

The syntax and semantics for arithmetic were changed for the ISO/IEC 9945-2:1993 standard. The language is essentially a pure arithmetic evaluator of constants and operators (excluding assignment) and represents a simple subset of the previous arithmetic language (which was derived from the KornShell "(( ))" construct). The syntax was changed from that of a command denoted by ((*expression*)) to an expansion denoted by \$(*expression*). The new form is a dollar expansion ('\$') that evaluates the expression and substitutes the resulting value. Objections to the previous style of arithmetic included that it was too complicated, did not fit in well with the use of variables in the shell, and its syntax conflicted with subshells. The justification for the new syntax is that the shell is traditionally a macro language, and if a new

9015 feature is to be added, it should be accomplished by extending the capabilities presented by the  
 9016 current model of the shell, rather than by inventing a new one outside the model; adding a new  
 9017 dollar expansion was perceived to be the most intuitive and least destructive way to add such a  
 9018 new capability.

9019 In early proposals, a form  $\$(expression)$  was used. It was functionally equivalent to the " $\$(())$ "  
 9020 of the current text, but objections were lodged that the 1988 KornShell had already implemented  
 9021 " $\$(())$ " and there was no compelling reason to invent yet another syntax. Furthermore, the  
 9022 " $\$[]$ " syntax had a minor incompatibility involving the patterns in **case** statements.

9023 The portion of the ISO C standard arithmetic operations selected corresponds to the operations  
 9024 historically supported in the KornShell.

9025 It was concluded that the *test* command (**D**) was sufficient for the majority of relational arithmetic  
 9026 tests, and that tests involving complicated relational expressions within the shell are rare, yet  
 9027 could still be accommodated by testing the value of " $\$(())$ " itself. For example:

```
9028 # a complicated relational expression
9029 while [$\$((($x + $y) / ($a * $b)) < ($foo * $bar)) -ne 0]$
```

9030 or better yet, the rare script that has many complex relational expressions could define a  
 9031 function like this:

```
9032 val() {
9033 return $\$(! $1)$
9034 }
```

9035 and complicated tests would be less intimidating:

```
9036 while val $\$((($x + $y) / ($a * $b)) < ($foo * $bar))$
9037 do
9038 # some calculations
9039 done
```

9040 A suggestion that was not adopted was to modify *true* and *false* to take an optional argument,  
 9041 and *true* would exit true only if the argument was non-zero, and *false* would exit false only if the  
 9042 argument was non-zero:

```
9043 while true $\$(($x > 5 && $y <= 25))$
```

9044 There is a minor portability concern with the new syntax. The example  $\$(2+2)$  could have been  
 9045 intended to mean a command substitution of a utility named *2+2* in a subshell. The standard  
 9046 developers considered this to be obscure and isolated to some KornShell scripts (because " $\$( )$ "  
 9047 command substitution existed previously only in the KornShell). The text on command  
 9048 substitution requires that the " $\$($ " and ' $($ ' be separate tokens if this usage is needed.

9049 An example such as:

```
9050 echo $\$((echo hi); (echo there))$
```

9051 should not be misinterpreted by the shell as arithmetic because attempts to balance the  
 9052 parentheses pairs would indicate that they are subshells. However, as indicated by the Base  
 9053 Definitions volume of IEEE Std. 1003.1-200x, Section 3.115, Control Operator, a conforming  
 9054 application must separate two adjacent parentheses with white space to indicate nested  
 9055 subshells.

9056 **C.2.6.5** *Field Splitting*

9057 The operation of field splitting using *IFS*, as described in early proposals, was based on the way  
 9058 the KornShell splits words, but it is incompatible with other common versions of the shell.  
 9059 However, each has merit, and so a decision was made to allow both. If the *IFS* variable is unset  
 9060 or is `<space><tab><newline>`, the operation is equivalent to the way the System V shell splits  
 9061 words. Using characters outside the `<space><tab><newline>` set yields the KornShell behavior,  
 9062 where each of the non-`<space><tab><newline>` characters is significant. This behavior, which  
 9063 affords the most flexibility, was taken from the way the original *awk* handled field splitting.

9064 Rule (3) can be summarized as a pseudo-ERE:

9065  $(s^*ns^*|s^+)$

9066 where *s* is an *IFS* white space character and *n* is a character in the *IFS* that is not white space.  
 9067 Any string matching that ERE delimits a field, except that the *s+* form does not delimit fields at  
 9068 the beginning or the end of a line. For example, if *IFS* is `<space>/<comma>/<tab>`, the string:

9069 `<space><space>red<space><space>, <space>white<space>blue`

9070 yields the three colors as the delimited fields.

9071 **C.2.6.6** *Path Name Expansion*

9072 There is no additional rationale for this section.

9073 **C.2.6.7** *Quote Removal*

9074 There is no additional rationale for this section.

9075 **C.2.7** **Redirection**

9076 In the System Interfaces volume of IEEE Std. 1003.1-200x, file descriptors are integers in the  
 9077 range 0–(`{OPEN_MAX}`–1). The file descriptors discussed in the Shell and Utilities volume of  
 9078 IEEE Std. 1003.1-200x, Section 2.7, Redirection are that same set of small integers.

9079 Having multi-digit file descriptor numbers for I/O redirection can cause some obscure  
 9080 compatibility problems. Specifically, scripts that depend on an example command:

9081 `echo 22>/dev/null`

9082 echoing 2 to standard error or 22 to standard output are no longer portable. However, the file  
 9083 descriptor number still must be delimited from the preceding text. For example:

9084 `cat file2>foo`

9085 writes the contents of **file2**, not the contents of **file**.

9086 The "`>`" format of output redirection was adopted from the KornShell. Along with the  
 9087 *noclobber* option, *set -C*, it provides a safety feature to prevent inadvertent overwriting of  
 9088 existing files. (See the RATIONALE for the *pathchk* utility for why this step was taken.) The  
 9089 restriction on regular files is historical practice.

9090 The System V shell and the KornShell have differed historically on path name expansion of *word*;  
 9091 the former never performed it, the latter only when the result was a single field (file). As a  
 9092 compromise, it was decided that the KornShell functionality was useful, but only as a shorthand  
 9093 device for interactive users. No reasonable shell script would be written with a command such  
 9094 as:

9095 `cat foo > a*`



9096 Thus, shell scripts are prohibited from doing it, while interactive users can select the shell with  
9097 which they are most comfortable.

9098 The construct `2>&1` is often used to redirect standard error to the same file as standard output.  
9099 Since the redirections take place beginning to end, the order of redirections is significant. For  
9100 example:

```
9101 ls > foo 2>&1
```

9102 directs both standard output and standard error to file **foo**. However:

```
9103 ls 2>&1 > foo
```

9104 only directs standard output to file **foo** because standard error was duplicated as standard  
9105 output before standard output was directed to file **foo**.

9106 The "<>" operator could be useful in writing an application that worked with several terminals,  
9107 and occasionally wanted to start up a shell. That shell would in turn be unable to run  
9108 applications that run from an ordinary controlling terminal unless it could make use of "<>"  
9109 redirection. The specific example is a historical version of the pager *more*, which reads from  
9110 standard error to get its commands, so standard input and standard output are both available  
9111 for their usual usage. There is no way of saying the following in the shell without "<>":

```
9112 cat food | more - >/dev/tty03 2<>/dev/tty03
```

9113 Another example of "<>" is one that opens `/dev/tty` on file descriptor 3 for reading and writing:

```
9114 exec 3<> /dev/tty
```

9115 An example of creating a lock file for a critical code region:

```
9116 set -C
9117 until 2> /dev/null > lockfile
9118 do sleep 30
9119 done
9120 set +C
9121 perform critical function
9122 rm lockfile
```

9123 Since `/dev/null` is not a regular file, no error is generated by redirecting to it in *noclobber* mode.

9124 Tilde expansion is not performed on a here-document because the data is treated as if it were  
9125 enclosed in double quotes.

#### 9126 *C.2.7.1 Redirecting Input*

9127 There is no additional rationale for this section.

#### 9128 *C.2.7.2 Redirecting Output*

9129 There is no additional rationale for this section.

#### 9130 *C.2.7.3 Appending Redirected Output*

9131 There is no additional rationale for this section.

9132 C.2.7.4 *Here-Document*

9133 There is no additional rationale for this section.

9134 C.2.7.5 *Duplicating an Input File Descriptor*

9135 There is no additional rationale for this section.

9136 C.2.7.6 *Duplicating an Output File Descriptor*

9137 There is no additional rationale for this section.

9138 C.2.7.7 *Open File Descriptors for Reading and Writing*

9139 There is no additional rationale for this section.

9140 **C.2.8 Exit Status and Errors**9141 C.2.8.1 *Consequences of Shell Errors*

9142 There is no additional rationale for this section.

9143 C.2.8.2 *Exit Status for Commands*

9144 There is a historical difference in *sh* and *ksh* non-interactive error behavior. When a command  
9145 named in a script is not found, some implementations of *sh* exit immediately, but *ksh* continues  
9146 with the next command. Thus, the Shell and Utilities volume of IEEE Std. 1003.1-200x says that  
9147 the shell “may” exit in this case. This puts a small burden on the programmer, who has to test  
9148 for successful completion following a command if it is important that the next command not be  
9149 executed if the previous command was not found. If it is important for the command to have  
9150 been found, it was probably also important for it to complete successfully. The test for successful  
9151 completion would not need to change.

9152 Historically, shells have returned an exit status of  $128+n$ , where  $n$  represents the signal number.  
9153 Since signal numbers are not standardized, there is no portable way to determine which signal  
9154 caused the termination. Also, it is possible for a command to exit with a status in the same range  
9155 of numbers that the shell would use to report that the command was terminated by a signal.  
9156 Implementations are encouraged to choose exit values greater than 256 to indicate programs  
9157 that terminate by a signal so that the exit status cannot be confused with an exit status generated  
9158 by a normal termination.

9159 Historical shells make the distinction between “utility not found” and “utility found but cannot  
9160 execute” in their error messages. By specifying two seldomly used exit status values for these  
9161 cases, 127 and 126 respectively, this gives an application the opportunity to make use of this  
9162 distinction without having to parse an error message that would probably change from locale to  
9163 locale. The *command*, *env*, *nohup*, and *xargs* utilities in the Shell and Utilities volume of  
9164 IEEE Std. 1003.1-200x have also been specified to use this convention.

9165 When a command fails during word expansion or redirection, most historical implementations  
9166 exit with a status of 1. However, there was some sentiment that this value should probably be  
9167 much higher so that an application could distinguish this case from the more normal exit status  
9168 values. Thus, the language “greater than zero” was selected to allow either method to be  
9169 implemented.

9170 **C.2.9 Shell Commands**

9171 A description of an “empty command” was removed from an early proposal because it is only  
 9172 relevant in the cases of *sh -c " "*, *system(" ")*, or an empty shell-script file (such as the  
 9173 implementation of *true* on some historical systems). Since it is no longer mentioned in the Shell  
 9174 and Utilities volume of IEEE Std. 1003.1-200x, it falls into the silently unspecified category of  
 9175 behavior where implementations can continue to operate as they have historically, but  
 9176 conforming applications do not construct empty commands. (However, note that *sh* does  
 9177 explicitly state an exit status for an empty string or file.) In an interactive session or a script with  
 9178 other commands, extra <newline>s or semicolons, such as;

```
9179 $ false
9180 $
9181 $ echo $?
9182 1
```

9183 would not qualify as the empty command described here because they would be consumed by  
 9184 other parts of the grammar.

9185 **C.2.9.1 Simple Commands**

9186 The enumerated list is used only when the command is actually going to be executed. For  
 9187 example, in:

```
9188 true || $foo *
```

9189 no expansions are performed.

9190 The following example illustrates both how a variable assignment without a command name  
 9191 affects the current execution environment, and how an assignment with a command name only  
 9192 affects the execution environment of the command:

```
9193 $ x=red
9194 $ echo $x
9195 red
9196 $ export x
9197 $ sh -c 'echo $x'
9198 red
9199 $ x=blue sh -c 'echo $x'
9200 blue
9201 $ echo $x
9202 red
```

9203 This next example illustrates that redirections without a command name are still performed:

```
9204 $ ls foo
9205 ls: foo: no such file or directory
9206 $ > foo
9207 $ ls foo
9208 foo
```

9209 A command without a command name, but one that includes a command substitution, has an  
 9210 exit status of the last command substitution that the shell performed. For example:

```
9211 if x=$(command)
9212 then ...
9213 fi
```

9214 An example of redirections without a command name being performed in a subshell shows that  
9215 the here-document does not disrupt the standard input of the **while** loop:

```
9216 IFS=:
9217 while read a b
9218 do echo $a
9219 <<-eof
9220 Hello
9221 eof
9222 done </etc/passwd
```

9223 Some examples of commands without command names in AND-OR lists:

```
9224 > foo || {
9225 echo "error: foo cannot be created" >&2
9226 exit 1
9227 }
9228 # set saved if /vmunix.save exists
9229 test -f /vmunix.save && saved=1
```

9230 Command substitution and redirections without command names both occur in subshells, but  
9231 they are not necessarily the same ones. For example, in:

```
9232 exec 3> file
9233 var=$(echo foo >&3) 3>&1
```

9234 it is unspecified whether **foo** is echoed to the file or to standard output.

### 9235 **Command Search and Execution**

9236 This description requires that the shell can execute shell scripts directly, even if the underlying  
9237 system does not support the common "#!" interpreter convention. That is, if file **foo** contains  
9238 shell commands and is executable, the following executes **foo**:

```
9239 ./foo
```

9240 The command search shown here does not match all historical implementations. A more typical  
9241 sequence has been:

- 9242 • Any built-in (special or regular)
- 9243 • Functions
- 9244 • Path search for executable files

9245 But there are problems with this sequence. Since the programmer has no idea in advance which  
9246 utilities might have been built into the shell, a function cannot be used to override portably a  
9247 utility of the same name. (For example, a function named *cd* cannot be written for many  
9248 historical systems.) Furthermore, the *PATH* variable is partially ineffective in this case, and only  
9249 a path name with a slash can be used to ensure a specific executable file is invoked.

9250 After the *execve()* failure described, the shell normally executes the file as a shell script. Some  
9251 implementations, however, attempt to detect whether the file is actually a script and not an  
9252 executable from some other architecture. The method used by the KornShell is allowed by the  
9253 text that indicates non-text files may be bypassed.

9254 The sequence selected for the Shell and Utilities volume of IEEE Std. 1003.1-200x acknowledges  
9255 that special built-ins cannot be overridden, but gives the programmer full control over which  
9256 versions of other utilities are executed. It provides a means of suppressing function lookup (via

9257 the *command* utility) for the user's own functions and ensures that any regular built-ins or  
 9258 functions provided by the implementation are under the control of the path search. The  
 9259 mechanisms for associating built-ins or functions with executable files in the path are not  
 9260 specified by the Shell and Utilities volume of IEEE Std. 1003.1-200x, but the wording requires  
 9261 that if either is implemented, the application is not able to distinguish a function or built-in from  
 9262 an executable (other than in terms of performance, presumably). The implementation ensures  
 9263 that all effects specified by the Shell and Utilities volume of IEEE Std. 1003.1-200x resulting from  
 9264 the invocation of the regular built-in or function (interaction with the environment, variables,  
 9265 traps, and so on) are identical to those resulting from the invocation of an executable file.

### 9266 Examples

9267 Consider three versions of the *ls* utility:

- 9268 1. The application includes a shell function named *ls*.
- 9269 2. The user writes a utility named *ls* and puts it in **/fred/bin**.
- 9270 3. The example implementation provides *ls* as a regular shell built-in that is invoked (either  
 9271 by the shell or directly by *exec*) when the path search reaches the directory **/posix/bin**.

9272 If *PATH*=**/posix/bin**, various invocations yield different versions of *ls*:

9273

9274

9275

9276

9277

9278

9279

| Invocation                                             | Version of <i>ls</i> |
|--------------------------------------------------------|----------------------|
| <i>ls</i> (from within application script)             | (1) function         |
| <i>command ls</i> (from within application script)     | (3) built-in         |
| <i>ls</i> (from within makefile called by application) | (3) built-in         |
| <i>system("ls")</i>                                    | (3) built-in         |
| <i>PATH="/fred/bin:\$PATH" ls</i>                      | (2) user's version   |

### 9280 C.2.9.2 Pipelines

9281 Because pipeline assignment of standard input or standard output or both takes place before  
 9282 redirection, it can be modified by redirection. For example:

```
9283 $ command1 2>&1 | command2
```

9284 sends both the standard output and standard error of *command1* to the standard input of  
 9285 *command2*.

9286 The reserved word **!** allows more flexible testing using AND and OR lists.

9287 It was suggested that it would be better to return a non-zero value if any command in the  
 9288 pipeline terminates with non-zero status (perhaps the bitwise-inclusive OR of all return values).  
 9289 However, the choice of the last-specified command semantics are historical practice and would  
 9290 cause applications to break if changed. An example of historical behavior:

```
9291 $ sleep 5 | (exit 4)
```

```
9292 $ echo $?
```

```
9293 4
```

```
9294 $ (exit 4) | sleep 5
```

```
9295 $ echo $?
```

```
9296 0
```

9297 *C.2.9.3 Lists*

9298 The equal precedence of "&&" and "||" is historical practice. The standard developers  
 9299 evaluated the model used more frequently in high-level programming languages, such as C, to  
 9300 allow the shell logical operators to be used for complex expressions in an unambiguous way, but  
 9301 they could not allow historical scripts to break in the subtle way unequal precedence might  
 9302 cause. Some arguments were posed concerning the "{" or "(" groupings that are required  
 9303 historically. There are some disadvantages to these groupings:

- 9304 • The "(" can be expensive, as they spawn other processes on some systems. This  
 9305 performance concern is primarily an implementation issue.
- 9306 • The "{" braces are not operators (they are reserved words) and require a trailing space  
 9307 after each '{', and a semicolon before each '}'. Most programmers (and certainly  
 9308 interactive users) have avoided braces as grouping constructs because of the problematic  
 9309 syntax required. Braces were not changed to operators because that would generate  
 9310 compatibility issues even greater than the precedence question; braces appear outside the  
 9311 context of a keyword in many shell scripts.

9312 **Asynchronous Lists**

9313 The grammar treats a construct such as:

```
9314 foo & bar & bam &
```

9315 as one "asynchronous list", but since the status of each element is tracked by the shell, the term  
 9316 "element of an asynchronous list" was introduced to identify just one of the **foo**, **bar**, or **bam**  
 9317 portions of the overall list.

9318 Unless the implementation has an internal limit, such as {CHILD\_MAX}, on the retained process  
 9319 IDs, it would require unbounded memory for the following example:

```
9320 while true
9321 do foo & echo $!
9322 done
```

9323 The treatment of the signals SIGINT and SIGQUIT with asynchronous lists is described in the  
 9324 Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.12, Signals and Error Handling.

9325 Since the connection of the input to the equivalent of /dev/null is considered to occur before  
 9326 redirections, the following script would produce no output:

```
9327 exec < /etc/passwd
9328 cat <&0 &
9329 wait
```

9330 **Sequential Lists**

9331 There is no additional rationale for this section.

9332        **AND Lists**

9333        There is no additional rationale for this section.

9334        **OR Lists**

9335        There is no additional rationale for this section.

9336    *C.2.9.4 Compound Commands*9337        **Grouping Commands**9338        The semicolon shown *{compound-list;}* is an example of a control operator delimiting the } reserved word. Other delimiters are possible, as shown in the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.11, Shell Grammar; <newline> is frequently used.

9341        A proposal was made to use the &lt;do-done&gt; construct in all cases where command grouping in the current process environment is performed, identifying it as a construct for the grouping commands, as well as for shell functions. This was not included because the shell already has a grouping construct for this purpose ("{}"), and changing it would have been counter-productive.

9346        **For Loop**

9347        The format is shown with generous usage of &lt;newline&gt;s. See the grammar in the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.11, Shell Grammar for a precise description of where &lt;newline&gt;s and semicolons can be interchanged.

9350        Some historical implementations support ' { ' and ' } ' as substitutes for **do** and **done**. The standard developers chose to omit them, even as an obsolescent feature. (Note that these substitutes were only for the **for** command; the **while** and **until** commands could not use them historically because they are followed by compound-lists that may contain "{ . . . }" grouping commands themselves.)9355        The reserved word pair **do** ... **done** was selected rather than **do** ... **od** (which would have matched the spirit of **if** ... **fi** and **case** ... **esac**) because *od* is already the name of a standard utility.

9358        PASC Interpretation 1003.2 #169 has been applied changing the grammar.

9359        **Case Conditional Construct**9360        An optional left parenthesis before *pattern* was added to allow numerous historical KornShell scripts to conform. At one time, using the leading parenthesis was required if the **case** statement was to be embedded within a "\$ ( )" command substitution; this is no longer the case with the POSIX shell. Nevertheless, many historical scripts use the left parenthesis, if only because it makes matching-parenthesis searching easier in *vi* and other editors. This is a relatively simple implementation change that is upward-compatible for all scripts.9366        Consideration was given to requiring *break* inside the *compound-list* to prevent falling through to the next pattern action list. This was rejected as being nonexistent practice. An interesting undocumented feature of the KornShell is that using ";&" instead of ";;" as a terminator causes the exact opposite behavior—the flow of control continues with the next *compound-list*.9370        The pattern ' \* ', given as the last pattern in a **case** construct, is equivalent to the default case in a C-language **switch** statement.

9371

9372 The grammar shows that reserved words can be used as patterns, even if one is the first word on  
9373 a line. Obviously, the reserved word **esac** cannot be used in this manner.

#### 9374 **If Conditional Construct**

9375 The precise format for the command syntax is described in the Shell and Utilities volume of  
9376 IEEE Std. 1003.1-200x, Section 2.11, Shell Grammar.

#### 9377 **While Loop**

9378 The precise format for the command syntax is described in the Shell and Utilities volume of  
9379 IEEE Std. 1003.1-200x, Section 2.11, Shell Grammar.

#### 9380 **Until Loop**

9381 The precise format for the command syntax is described in the Shell and Utilities volume of  
9382 IEEE Std. 1003.1-200x, Section 2.11, Shell Grammar.

#### 9383 *C.2.9.5 Function Definition Command*

9384 The description of functions in an early proposal was based on the notion that functions should  
9385 behave like miniature shell scripts; that is, except for sharing variables, most elements of an  
9386 execution environment should behave as if they were a new execution environment, and  
9387 changes to these should be local to the function. For example, traps and options should be reset  
9388 on entry to the function, and any changes to them do not affect the traps or options of the caller.  
9389 There were numerous objections to this basic idea, and the opponents asserted that functions  
9390 were intended to be a convenient mechanism for grouping common commands that were to be  
9391 executed in the current execution environment, similar to the execution of the *dot* special built-  
9392 in.

9393 It was also pointed out that the functions described in that early proposal did not provide a local  
9394 scope for everything a new shell script would, such as the current working directory, or *umask*,  
9395 but instead provided a local scope for only a few select properties. The basic argument was that  
9396 if a local scope is needed for the execution environment, the mechanism already existed: the  
9397 application can put the commands in a new shell script and call that script. All historical shells  
9398 that implemented functions, other than the KornShell, have implemented functions that operate  
9399 in the current execution environment. Because of this, traps and options have a global scope  
9400 within a shell script. Local variables within a function were considered and included in another  
9401 early proposal (controlled by the special built-in *local*), but were removed because they do not fit  
9402 the simple model developed for functions and because there was some opposition to adding yet  
9403 another new special built-in that was not part of historical practice. Implementations should  
9404 reserve the identifier *local* (as well as *typeset*, as used in the KornShell) in case this local variable  
9405 mechanism is adopted in a future version of IEEE Std. 1003.1-200x.

9406 A separate issue from the execution environment of a function is the availability of that function  
9407 to child shells. A few objectors maintained that just as a variable can be shared with child shells  
9408 by exporting it, so should a function. In early proposals, the *export* command therefore had a *-f*  
9409 flag for exporting functions. Functions that were exported were to be put into the environment  
9410 as *name()=value* pairs, and upon invocation, the shell would scan the environment for these and  
9411 automatically define these functions. This facility was strongly opposed and was omitted. Some  
9412 of the arguments against exportable functions were as follows:

- 9413 • There was little historical practice. The Ninth Edition shell provided them, but there was  
9414 controversy over how well it worked.



- 9415 • There are numerous security problems associated with functions appearing in the  
9416 environment of a user and overriding standard utilities or the utilities owned by the  
9417 application.
- 9418 • There was controversy over requiring *make* to import functions, where it has historically used  
9419 an *exec* function for many of its command line executions.
- 9420 • Functions can be big and the environment is of a limited size. (The counter-argument was  
9421 that functions are no different than variables in terms of size: there can be big ones, and there  
9422 can be small ones—and just as one does not export huge variables, one does not export huge  
9423 functions. However, this might not apply to the average shell-function writer, who typically  
9424 writes much larger functions than variables.)

9425 As far as can be determined, the functions in the Shell and Utilities volume of  
9426 IEEE Std. 1003.1-200x match those in System V. Earlier versions of the KornShell had two  
9427 methods of defining functions:

```
9428 function fname { compound-list }
```

9429 and:

```
9430 fname() { compound-list }
```

9431 The latter used the same definition as the Shell and Utilities volume of IEEE Std. 1003.1-200x, but  
9432 differed in semantics, as described previously. The current edition of the KornShell aligns the  
9433 latter syntax with the Shell and Utilities volume of IEEE Std. 1003.1-200x and keeps the former as  
9434 is.

9435 The name space for functions is limited to that of a *name* because of historical practice.  
9436 Complications in defining the syntactic rules for the function definition command and in dealing  
9437 with known extensions such as the "@()" usage in the KornShell prevented the name space  
9438 from being widened to a *word*. Using functions to support synonyms such as the "!!" and '%'  
9439 usage in the C shell is thus disallowed to portable applications, but acceptable as an extension.  
9440 For interactive users, the aliasing facilities in the Shell and Utilities volume of  
9441 IEEE Std. 1003.1-200x should be adequate for this purpose. It is recognized that the name space  
9442 for utilities in the file system is wider than that currently supported for functions, if the portable  
9443 file name character set guidelines are ignored, but it did not seem useful to mandate extensions  
9444 in systems for so little benefit to portable applications.

9445 The "()" in the function definition command consists of two operators. Therefore, intermixing  
9446 <blank>s with the *fname*, '( ', and ' )' is allowed, but unnecessary.

9447 An example of how a function definition can be used wherever a simple command is allowed:

```
9448 # If variable i is equal to "yes",
9449 # define function foo to be ls -l
9450 #
9451 ["$i" = yes] && foo() {
9452 ls -l
9453 }
```

9454 **C.2.10 Executable Script**

9455 The working group did not reach consensus to adopt this as a core requirement—that is, for  
 9456 POSIX-conforming applications—however, existing practice on UNIX systems indicated that it  
 9457 should be added as an XSI extension, and this was brought into the scope of this revision by The  
 9458 Open Group Base Resolution bwg2000-004. The scope of this feature is to document existing  
 9459 practice and not to invent.

9460 Applications must not assume that the standard utilities will be available in any particular  
 9461 named directory. For example, it cannot be assumed that standard versions of *awk* and *sh* will be  
 9462 available as */bin/sh* or */bin/awk*, respectively, since implementations are permitted to provide  
 9463 non-standard versions of the utilities in these directories.

9464 It is recommended that an installation script for executable scripts use the standard *PATH*  
 9465 returned by a call to the *getconf* utility with the argument *PATH*, combined with the *command*  
 9466 utility to determine the location of a standard utility.

9467 For example, to determine the location of the standard *sh* utility:

```
9468 command -v sh
```

9469 On some systems this might return:

```
9470 /usr/xpg4/bin/sh
```

9471 Note that the installation script should ensure that the returned path name is an absolute path  
 9472 name prior to use, since a shell built-in might be returned for some utilities.

9473 **C.2.11 Shell Grammar**

9474 There are several subtle aspects of this grammar where conventional usage implies rules about  
 9475 the grammar that in fact are not true.

9476 For *compound\_list*, only the forms that end in a *separator* allow a reserved word to be recognized,  
 9477 so usually only a *separator* can be used where a compound list precedes a reserved word (such as  
 9478 **Then**, **Else**, **Do** and **Rbrace**). Explicitly requiring a separator would disallow such valid (if rare)  
 9479 statements as:

```
9480 if (false) then (echo x) else (echo y) fi
```

9481 See the Note under special grammar rule 1.

9482 Concerning the third sentence of rule (1) (“Also, if the parser ...”):

- 9483 • This sentence applies rather narrowly: when a compound list is terminated by some clear  
 9484 delimiter (such as the closing **fi** of an inner **if\_clause**) then it would apply; where the  
 9485 compound list might continue (as in after a *;*), rule (7a) (and consequently the first  
 9486 sentence of rule (1)) would apply. In many instances the two conditions are identical, but this  
 9487 part of rule (1) does not give license to treating a **WORD** as a reserved word unless it is in a  
 9488 place where a reserved word has to appear.

- 9489 • The statement is equivalent to requiring that when the LR(1) lookahead set contains exactly  
 9490 one reserved word, it must be recognized if it is present. (Here “LR(1)” refers to the  
 9491 theoretical concepts, not to any real parser generator.)

9492 For example, in the construct below, and when the parser is at the point marked with ‘^’,  
 9493 the only next legal token is **then** (this follows directly from the grammar rules):

```
9494 if if...fi then ... fi
9495 ^
```

9496 At that point, the **then** must be recognized as a reserved word.

9497 (Depending on the parser generator actually used, “extra” reserved words may be in some  
9498 lookahead sets. It does not really matter if they are recognized, or even if any possible  
9499 reserved word is recognized in that state, because if it is recognized and is not in the  
9500 (theoretical) LR(1) lookahead set, an error is ultimately detected. In the example above, if  
9501 some other reserved word (for example, **while**) is also recognized, an error occurs later.

9502 This is approximately equivalent to saying that reserved words are recognized after other  
9503 reserved words (because it is after a reserved word that this condition occurs), but avoids the  
9504 “except for ...” list that would be required for **case**, **for**, and so on. (Reserved words are of  
9505 course recognized anywhere a *simple\_command* can appear, as well. Other rules take care of  
9506 the special cases of non-recognition, such as rule (4) for **case** statements.)

9507 Note that the body of here-documents are handled by token recognition (see the Shell and  
9508 Utilities volume of IEEE Std. 1003.1-200x, Section 2.3, Token Recognition) and do not appear in  
9509 the grammar directly. (However, the here-document I/O redirection operator is handled as part  
9510 of the grammar.)

9511 The start symbol of the grammar (**complete\_command**) represents either input from the  
9512 command line or a shell script. It is repeatedly applied by the interpreter to its input and  
9513 represents a single “chunk” of that input as seen by the interpreter.

9514 *C.2.11.1 Shell Grammar Lexical Conventions*

9515 There is no additional rationale for this section.

9516 *C.2.11.2 Shell Grammar Rules*

9517 There is no additional rationale for this section.

9518 **C.2.12 Signals and Error Handling**

9519 There is no additional rationale for this section.

9520 **C.2.13 Shell Execution Environment**

9521 Some systems have implemented the last stage of a pipeline in the current environment so that  
9522 commands such as:

9523 `command | read foo`

9524 set variable **foo** in the current environment. This extension is allowed, but not required;  
9525 therefore, a shell programmer should consider a pipeline to be in a subshell environment, but  
9526 not depend on it.

9527 In early proposals, the description of execution environment failed to mention that each  
9528 command in a multiple command pipeline could be in a subshell execution environment. For  
9529 compatibility with some historical shells, the wording was phrased to allow an implementation  
9530 to place any or all commands of a pipeline in the current environment. However, this means that  
9531 a POSIX application must assume each command is in a subshell environment, but not depend  
9532 on it.

9533 The wording about shell scripts is meant to convey the fact that describing “trap actions” can  
9534 only be understood in the context of the shell command language. Outside of this context, such  
9535 as in a C-language program, signals are the operative condition, not traps.

9536 **C.2.14 Pattern Matching Notation**

9537 Pattern matching is a simpler concept and has a simpler syntax than REs, as the former is  
 9538 generally used for the manipulation of file names, which are relatively simple collections of  
 9539 characters, while the latter is generally used to manipulate arbitrary text strings of potentially  
 9540 greater complexity. However, some of the basic concepts are the same, so this section points  
 9541 liberally to the detailed descriptions in the Base Definitions volume of IEEE Std. 1003.1-200x,  
 9542 Chapter 9, Regular Expressions.

9543 *C.2.14.1 Patterns Matching a Single Character*

9544 Both quoting and escaping are described here because pattern matching must work in three  
 9545 separate circumstances:

9546 1. Calling directly upon the shell, such as in path name expansion or in a **case** statement. All  
 9547 of the following match the string or file **abc**:

9548 `abc "abc" a"b" c a\b c a[b] c a["b"] c a[\b] c a["\b"] c a?c a*c`

9549 The following do not:

9550 `"a?c" a*c a\b c`

9551 2. Calling a utility or function without going through a shell, as described for *find* and the  
 9552 *fmatch()* function defined in the System Interfaces volume of IEEE Std. 1003.1-200x.

9553 3. Calling utilities such as *find*, *cpio*, *tar*, or *pax* through the shell command line. In this case,  
 9554 shell quote removal is performed before the utility sees the argument. For example, in:

9555 `find /bin -name "e\c[\h]o" -print`

9556 after quote removal, the backslashes are presented to *find* and it treats them as escape  
 9557 characters. Both precede ordinary characters, so the *c* and *h* represent themselves and *echo*  
 9558 would be found on many historical systems (that have it in **/bin**). To find a file name that  
 9559 contained shell special characters or pattern characters, both quoting and escaping are  
 9560 required, such as:

9561 `pax -r ... "*a\(\?"`

9562 to extract a file name ending with "a(?".

9563 Conforming applications are required to quote or escape the shell special characters (sometimes  
 9564 called metacharacters). If used without this protection, syntax errors can result or  
 9565 implementation extensions can be triggered. For example, the KornShell supports a series of  
 9566 extensions based on parentheses in patterns.

9567 The restriction on a circumflex in a bracket expression is to allow implementations that support  
 9568 pattern matching using the circumflex as the negation character in addition to the exclamation  
 9569 mark. A portable application must use something like "[\^!]" to match either character.

9570 *C.2.14.2 Patterns Matching Multiple Characters*

9571 Since each asterisk matches zero or more occurrences, the patterns "a\*b" and "a\*\*b" have  
 9572 identical functionality.

9573 **Examples**

- 9574 `a[bc]` Matches the strings "ab" and "ac".
- 9575 `a*d` Matches the strings "ad", "abd", and "abcd", but not the string "abc".
- 9576 `a*d*` Matches the strings "ad", "abcd", "abcdef", "aaaad", and "adddd".
- 9577 `*a*d` Matches the strings "ad", "abcd", "efabcd", "aaaad", and "adddd".

9578 **C.2.14.3 Patterns Used for File Name Expansion**

9579 The caveat about a slash within a bracket expression is derived from historical practice. The  
 9580 pattern "a[b/c]d" does not match such path names as **abd** or **a/d**. On some systems (including  
 9581 those conforming to the Single UNIX Specification), it matched a path name of literally  
 9582 "a[b/c]d". On other systems, it produced an undefined condition (an unescaped '[' used  
 9583 outside a bracket expression). In this version, the XSI behavior is now required.

9584 File names beginning with a period historically have been specially protected from view on  
 9585 UNIX systems. A proposal to allow an explicit period in a bracket expression to match a leading  
 9586 period was considered; it is allowed as an implementation extension, but a conforming  
 9587 application cannot make use of it. If this extension becomes popular in the future, it will be  
 9588 considered for a future version of the Shell and Utilities volume of IEEE Std. 1003.1-200x.

9589 Historical systems have varied in their permissions requirements. To match **f\*/bar** has required  
 9590 read permissions on the **f\*** directories in the System V shell, but the Shell and Utilities volume of  
 9591 IEEE Std. 1003.1-200x, the C shell, and KornShell require only search permissions.

9592 **C.2.15 Special Built-In Utilities**

9593 See the RATIONALE sections on the individual reference pages.

## 9594 C.3 Batch Environment Services and Utilities

### 9595 Scope of the Batch Environment Option

9596 This section summarizes the deliberations of the IEEE P1003.15 (Batch Environment) working  
9597 group in the development of the Batch Environment option, which covers a set of services and  
9598 utilities defining a batch processing system.

9599 This informative section contains historical information concerning the contents of the  
9600 amendment and describes why features were included or discarded by the working group.

### 9601 History of Batch Systems

9602 The supercomputing technical committee began as a “Birds Of a Feather” (BOF) at the January  
9603 1987 Usenix meeting. There was enough general interest to form a supercomputing attachment  
9604 to the /usr/group working groups. Several subgroups rapidly formed. Of those subgroups, the  
9605 batch group was the most ambitious. The first early meetings were spent evaluating user needs  
9606 and existing batch implementations.

9607 To evaluate user needs, individuals from the supercomputing community came and presented  
9608 their needs. Common requests were flexibility, interoperability, control of resources, and ease-  
9609 of-use. Backwards-compatibility was not an issue. The working group then evaluated some  
9610 existing systems. The following different systems were evaluated:

- 9611 • PROD
- 9612 • Convex Distributed Batch
- 9613 • NQS
- 9614 • CTSS
- 9615 • MDQS from Ballistics Research Laboratory (BRL)

9616 Finally, NQS was chosen as a model because it satisfied not only the most user requirements, but  
9617 because it was public domain, already implemented on a variety of hardware platforms, and  
9618 networked-based.

### 9619 Historical Implementations of Batch Systems

9620 Deferred processing of work under the control of a scheduler has been a feature of most  
9621 proprietary operating systems from the earliest days of multi-user systems in order to maximize  
9622 utilization of the computer.

9623 The arrival of UNIX systems proved to be a dilemma to many hardware providers and users  
9624 because it did not include the sophisticated batch facilities offered by the proprietary systems.  
9625 This omission was rectified in 1986 by NASA Ames Research Center who developed the  
9626 Network Queuing System (NQS) as a portable UNIX application that allowed the routing and  
9627 processing of batch “jobs” in a network. To encourage its usage, the product was later put into  
9628 the public domain. It was promptly picked up by UNIX hardware providers, and ported and  
9629 developed for their respective hardware and UNIX implementations.

9630 Many major vendors, who traditionally offer a batch-dominated environment, ported the  
9631 public-domain product to their systems, customized it to support the capabilities of their  
9632 systems, and added many customer-requested features.

9633 Due to the strong hardware provider and customer acceptance of NQS, it was decided to use  
9634 NQS as the basis for the POSIX Batch Environment amendment in 1987. Other batch systems  
9635 considered at the time included CTSS, MDQS (a forerunner of NQS from the Ballistics Research

9636 Laboratory), and PROD (a Los Alamos Labs development). None were thought to have both the  
9637 functionality and acceptability of NQS.

### 9638 **NQS Differences from the at utility**

9639 The base standard *at* and *batch* utilities are not sufficient to meet the batch processing needs in a  
9640 supercomputing environment and additional functionality in the areas of resource management,  
9641 job scheduling, system management, and control of output is required.

### 9642 **Batch Environment Option Definitions**

9643 The concept of a batch job is closely related to a session with a session leader. The main  
9644 difference is that a batch job does not have a controlling terminal. There has been much debate  
9645 over whether to use the term *request* or *job*. Job was the final choice because of the historical use  
9646 of this term in the batch environment.

9647 The current definition for job identifiers is not sufficient with the model of destinations. The  
9648 current definition is:

9649 `sequence_number.originating_host`

9650 Using the model of destination, a host may include multiple batch nodes, the location of which is  
9651 identified uniquely by a name or directory service. If the current definition is used, batch nodes  
9652 running on the same host would have to coordinate their use of sequence numbers, as sequence  
9653 numbers are assigned by the originating host. The alternative is to use the originating batch node  
9654 name instead of the originating host name.

9655 The reasons for wishing to run more than one batch system per host could be the following:

9656 A test and production batch system are maintained on a single host. This is most likely in a  
9657 development facility, but could also arise when a site is moving from one version to another.  
9658 The new batch system could be installed as a test version that is completely separate from the  
9659 production batch system, so that problems can be isolated to the test system. Requiring the batch  
9660 nodes to coordinate their use of sequence numbers creates a dependency between the two  
9661 nodes, and that defeats the purpose of running two nodes.

9662 A site has multiple departments using a single host, with different management policies. An  
9663 example of contention might be in job selection algorithms. One group might want a FIFO type  
9664 of selection, while another group wishes to use a more complex algorithm based on resource  
9665 availability. Again, requiring the batch nodes to coordinate is an unnecessary binding.

9666 The proposal eventually accepted was to replace originating host with originating batch node.  
9667 This supplies sufficient granularity to ensure unique job identifiers. If more than one batch node  
9668 is on a particular host, they each have their own unique name.

9669 The queue portion of a destination is not part of the job identifier as these are not required to be  
9670 unique between batch nodes. For instance, two batch nodes may both have queues called small,  
9671 medium, and large. It is only the batch node name that is uniquely identifiable throughout the  
9672 batch system. The queue name has no additional function in this context.

9673 Assume there are three batch nodes, each of which has its own name server. On batch node one,  
9674 there are no queues. On batch node two, there are fifty queues. On batch node three, there are  
9675 forty queues. The system administrator for batch node one does not have to configure queues,  
9676 because there are none implemented. However, if a user wishes to send a job to either batch  
9677 node two or three, the system administrator for batch node one must configure a destination  
9678 that maps to the appropriate batch node and queue. If every queue is to be made accessible from  
9679 batch node one, the system administrator has to configure ninety destinations.

9680 To avoid requiring this, there should be a mechanism to allow a user to separate the destination  
9681 into a batch node name and a queue name. Then, an implementation that is configured to get to  
9682 all the batch nodes does not need any more configuration to allow a user to get to all of the  
9683 queues on all of the batch nodes. The node name is used to locate the batch node, while the  
9684 queue name is sent unchanged to that batch node.

9685 The following are requirements that a destination identifier must be capable of providing:

- 9686 • The ability to direct a job to a queue in a particular batch node.
- 9687 • The ability to direct a job to a particular batch node.
- 9688 • The ability to group at a higher level than just one queue. This includes grouping similar  
9689 queues across multiple batch nodes (this is a pipe queue today).
- 9690 • The ability to group batch nodes. This allows a user to submit a job to a group name with no  
9691 knowledge of the batch node configuration. This also provides aliasing as a special case.  
9692 Aliasing is a group containing only one batch node name. The group name is the alias.

9693 In addition, the administrator has the following requirements:

- 9694 • The ability to control access to the queues.
- 9695 • The ability to control access to the batch nodes.
- 9696 • The ability to control access to groups of queues (pipe queues).
- 9697 • The ability to configure retry time intervals and durations.

9698 The requirements of the user are met by destination as explained in the following:

9699 The user has the ability to specify a queue name, which is known only to the batch node  
9700 specified. There is no configuration of these queues required on the submitting node.

9701 The user has the ability to specify a batch node whose name is network-unique. The  
9702 configuration required is that the batch node be defined as an application, just as other  
9703 applications such as FTP are configured.

9704 Once a job reaches a queue, it can again become a user of the batch system. The batch node can  
9705 choose to send the job to another batch node or queue or both. In other words, the routing is at  
9706 an application level, and it is up to the batch system to choose where the job will be sent.  
9707 Configuration is up to the batch node where the queue resides. This provides grouping of  
9708 queues across batch nodes or within a batch node. The user submits the job to a queue, which by  
9709 definition routes the job to other queues or nodes or both.

9710 A node name may be given to a naming service, which returns multiple addresses as opposed to  
9711 just one. This provides grouping at a batch node level. This is a local issue, meaning that the  
9712 batch node must choose only one of these addresses. The list of addresses is not sent with the  
9713 job, and once the job is accepted on another node, there is no connection between the list and the  
9714 job. The requirements of the administrator are met by destination as explained in the following:

9715 The control of queues is a batch system issue, and will be done using the batch administrative  
9716 utilities.

9717 The control of nodes is a network issue, and will be done through whatever network facilities  
9718 are available.

9719 The control of access to groups of queues (pipe queues) is covered by the control of any other  
9720 queue. The fact that the job may then be sent to another destination is not relevant.

9721 The propagation of a job across more than one point-to-point connection was dropped because  
9722 of its complexity and because all of the issues arising from this capability could not be resolved.



9723 It could be provided as additional functionality at some time in the future.

9724 The addition of *network* as a defined term was done to clarify the difference between a network  
9725 of batch nodes as opposed to a network of hosts. A network of batch nodes is referred to as a  
9726 batch system. The network refers to the actual host configuration. A single host may have  
9727 multiple batch nodes.

9728 In the absence of a standard network naming convention, this option establishes its own  
9729 convention for the sake of consistency and expediency. This is subject to change, should a future  
9730 working group develop a standard naming convention for network path names.

### 9731 **C.3.1 Batch General Concepts**

9732 During the development of the Batch Environment option, a number of topics were discussed at  
9733 length which influenced the wording of the normative text but could not be included in the final  
9734 text. The following items are some of the most significant terms and concepts of those discussed:

- 9735 • Small and Consistent Command Set

9736 Often, conventional utilities from UNIX systems have a very complicated utility syntax and  
9737 usage. This can often result in confusion and errors when trying to use them. The Batch  
9738 Environment option utility set, on the other hand, has been paired to a small set of robust  
9739 utilities with an orthogonal calling sequence.

- 9740 • Checkpoint/Restart

9741 This feature permits an already executing process to checkpoint or save its contents. Some  
9742 implementations permit this at both the batch utility level; for example, checkpointing this  
9743 job upon its abnormal termination or from within the job itself via a system call. Support of  
9744 checkpoint/restart is optional. A conscious, careful effort was made to make the *qsub* and  
9745 *qmgr* utilities consistently refer to checkpoint/restart as optional functionality.

- 9746 • Rerunability

9747 When a user submits a job for batch processing, they can designate it “rerunnable” in that it  
9748 will automatically resume execution from the start of the job if the machine on which it was  
9749 executing crashes for some reason. The decision on whether the job will be rerun or not is  
9750 entirely up to the submitter of the job and no decisions will be made within the batch system.  
9751 A job that is rerunnable and has been submitted with the proper checkpoint/restart switch  
9752 will first be checkpointed and execution begun from that point. Furthermore, use of the  
9753 implementation-defined checkpoint/restart feature will be not be defined in this context.

- 9754 • Error Codes

9755 All utilities exit with error status zero (0) if successful, one (1) if a user error occurred, and  
9756 two (2) for an internal Batch Environment option error.

- 9757 • Level of Portability

9758 Portability is specified at both the user, operator, and administrator levels. A conforming  
9759 batch implementation prevents identical functionality and behavior at all these levels.  
9760 Additionally, portable batch shell scripts with embedded Batch Environment option utilities  
9761 adds an additional level of portability.

- 9762 • Resource Specification

9763 A small set of globally understood resources, such as memory and CPU time, is specified. All  
9764 conforming batch implementations are able to process them in a manner consistent with the  
9765 yet-to-be-developed resource management model. Resources not in this amendment set are  
9766 ignored and passed along as part of the argument stream of the utility.

- 9767
- Maximum of 80 Characters on Output Display
- 9768 At one time, existing displays were limited to 80 characters in length for purposes of  
9769 readability, both in the amendment and online. Current internationalization efforts  
9770 discourage specifying displays in the normative text. Instead, all suggested displays appear  
9771 in an informative annex for illustrative purposes. As before, length is limited to 80 characters  
9772 for readability purposes.
- 9773
- Queue Position
- 9774 Queue position is the place a job occupies in a queue. It is dependent on a variety of factors  
9775 such as submission time and priority. Since priority may be affected by the implementation  
9776 of fair share scheduling, the definition of queue position is implementation-defined.
- 9777
- Queue ID
- 9778 A numerical queue ID is an external requirement for purposes of accounting. The  
9779 identification number was chosen over queue name for processing convenience.
- 9780
- Job ID
- 9781 A common notion of “jobs” is a collection of processes whose process group cannot be  
9782 altered and is used for resource management and accounting. This concept is  
9783 implementation-defined and, as such, has been omitted from the batch amendment.
- 9784
- Bytes versus Words
- 9785 Except for one case, bytes are used as the standard unit for memory size. Furthermore, the  
9786 definition of a word varies from machine to machine. Therefore, bytes will be the default unit  
9787 of memory size.
- 9788
- Regular Expressions
- 9789 The standard definition of regular expressions is much too broad to be used in the batch  
9790 utility syntax. All that is needed is a simple concept of “all”; for example, delete all my jobs  
9791 from the named queue. For this reason, regular expressions have been eliminated from the  
9792 batch amendment.
- 9793
- Display Privacy
- 9794 How much data should be displayed locally through library functions? Local policy dictates  
9795 the amount of privacy. Library functions must be used to create and enforce local policy.  
9796 Network and local *qstats* must reflect the policy of the server machine.
- 9797
- Remote Host Naming Convention
- 9798 It was decided that host names would be a maximum of 255 characters in length, with at  
9799 most 15 characters being shown in displays. The 255 character limit was chosen because it is  
9800 consistent with BSD). The 15-character limit was an arbitrary decision.
- 9801
- Network Administration
- 9802 Network administration is important, but is outside the scope of the batch amendment.  
9803 Network administration could done with *rsh*. However, authentication becomes two-sided.
- 9804
- Network Administration Philosophy
- 9805 Keep it simple. Centralized management should be possible. For example, Los Alamos needs  
9806 a dumb set of CPUs to be managed by a central system versus several independently-  
9807 managed systems as is the general case for the Batch Environment option.

- 9808       • Operator Utility Defaults (that is, Default Host, User, Account, and so on)
- 9809        It was decided that usability would override orthogonality and syntactic consistency.
- 9810       • The Batch System Manager and Operator Distinction
- 9811        The distinction between manager and operator is that operators can only control the flow of
- 9812        jobs. A manager can alter the batch system configuration in addition to job flow. POSIX
- 9813        makes a distinction between user and system administrator but goes no further. The
- 9814        concepts of manager and operator privileges fall under local policy. The distinction between
- 9815        manager and operator is historical in batch environments, and the Batch Environment option
- 9816        has continued that distinction.
- 9817       • The Batch System Administrator
- 9818        An administrator is equivalent to a batch system manager.
- 9819       • Network Administration *versus* Checkpoint/Recovery
- 9820        Network administration is better left for a future revision of IEEE Std. 1003.1-200x. If
- 9821        network administration is put up against checkpoint/recovery, the argument is that there are
- 9822        possible solutions for network administration. However, checkpoint/recovery currently has
- 9823        no solution. This is another reason the issue of checkpoint/recovery should be addressed
- 9824        first.

### 9825 **C.3.2 Batch Services**

- 9826       This rationale is provided as informative rather than normative text, to avoid placing
- 9827       requirements on implementors regarding the use of symbolic constants, but at the same time to
- 9828       give implementors a preferred practice for assigning values to these constants to promote
- 9829       interoperability.
- 9830       The *Checkpoint* and *Minimum\_Cpu\_Interval* attributes induce a variety of behavior depending
- 9831       upon their values. Some jobs cannot or should not be checkpointed. Other users will simply
- 9832       need to ensure job continuation across planned downtimes; for example, scheduled preventive
- 9833       maintenance. For users consuming expensive resources, or for jobs that run longer than the
- 9834       mean time between failures, however, periodic checkpointing may be essential. However,
- 9835       system administrators must be able to set minimum checkpoint intervals on a queue-by-queue
- 9836       basis to guard against; for example, naive users specifying interval values too small on memory
- 9837       intensive jobs. Otherwise, system overhead would adversely affect performance.
- 9838       The use of symbolic constants, such as `NO_CHECKPOINT`, was introduced to lend a degree of
- 9839       formalism and portability to this option.
- 9840       Support for checkpointing is optional for servers. However, clients must provide for the `-c`
- 9841       option, since in a distributed environment the job may run on a server that does provide such
- 9842       support, even if the host of the client does not support the checkpoint feature.
- 9843       If the user does not specify the `-c` option, the default action is left unspecified by this option.
- 9844       Some implementations may wish to do checkpointing by default; others may wish to checkpoint
- 9845       only under an explicit request from the user.
- 9846       The *Priority* attribute has been made non-optional. All clients already had been required to
- 9847       support the `-p` option. The concept of prioritization is common in historical implementations.
- 9848       The default priority is left to the server to establish.
- 9849       The *Hold\_Types* attribute has been modified to allow for implementation-defined hold types to
- 9850       be passed to a batch server.

9851 It was the intent of the IEEE P1003.15 working group to mandate the support for the  
9852 *Resource\_List* attribute in this option by referring to another amendment, specifically P1003.1a.  
9853 However, during the development of P1003.1a this was excluded. As such this requirement has  
9854 been removed from the normative text.

9855 The *Shell\_Path* attribute has been modified to accept a list of shell paths that are associated with  
9856 a host. The name of the attribute has been changed to *Shell\_Path\_List*.

9857 **C.3.3 Common Behavior for Batch Environment Utilities**

9858 This section was defined to meet the goal of a “Small and Consistent Command Set” for this  
9859 option.

9860 **C.4 Utilities**

9861 See the RATIONALE sections on the individual reference pages.



9862 / *Rationale (Informative)*

9863 **Part D:**

9864 **Portability Considerations**

9865 *The Open Group*





## Portability Considerations (Informative)

9866

9867 This section contains information to satisfy various international requirements:

- 9868 • Section D.1 describes perceived user requirements.
- 9869 • Section D.2 (on page 3560) indicates how the facilities of IEEE Std. 1003.1-200x satisfy those  
9870 requirements.
- 9871 • Section D.3 (on page 3567) offers guidance to writers of profiles on how the configurable  
9872 options, limits, and optional behavior of IEEE Std. 1003.1-200x should be cited in profiles.

### 9873 D.1 User Requirements

9874 This section describes the user requirements that were perceived by the developers of  
9875 IEEE Std. 1003.1-200x. The primary source for these requirements was an analysis of historical  
9876 practice in widespread use, as typified by the base documents listed in Section A.1.1 (on page  
9877 3311).

9878 IEEE Std. 1003.1-200x addresses the needs of users requiring open systems solutions for source  
9879 code portability of applications. It currently addresses users requiring open systems solutions  
9880 for source-code portability of applications involving multi-programming and process  
9881 management (creating processes, signaling, and so on); access to files and directories in a  
9882 hierarchy of file systems (opening, reading, writing, deleting files, and so on); access to  
9883 asynchronous communications ports and other special devices; access to information about  
9884 other users of the system; facilities supporting applications requiring bounded (realtime)  
9885 response.

9886 The following users are identified for IEEE Std. 1003.1-200x:

- 9887 • Those employing applications written in high-level languages, such as C, Ada, or FORTRAN.
- 9888 • Users who desire portable applications that do not necessarily require the characteristics of  
9889 high-level languages (for example, the speed of execution of compiled languages or the  
9890 relative security of source code intellectual property inherent in the compilation process).
- 9891 • Users who desire portable applications that can be developed quickly and can be modified  
9892 readily without the use of compilers and other system components that may be unavailable  
9893 on small systems or those without special application development capabilities.
- 9894 • Users who interact with a system to achieve general-purpose time-sharing capabilities  
9895 common to most business or government offices or academic environments: editing, filing,  
9896 inter-user communications, printing, and so on.
- 9897 • Users who develop applications for POSIX-conformant systems.
- 9898 • Users who develop applications for UNIX systems.

9899 An acknowledged restriction on applicable users is that they are limited to the group of  
9900 individuals who are familiar with the style of interaction characteristic of historically-derived  
9901 systems based on one of the UNIX operating systems (as opposed to other historical systems  
9902 with different models, such as MS/DOS, Macintosh, VMS, MVS, and so on). Typical users  
9903 would include program developers, engineers, or general-purpose time-sharing users.

9904 The requirements of users of IEEE Std. 1003.1-200x can be summarized as a single goal:  
9905 *application source portability*. The requirements of the user are stated in terms of the requirements  
9906 of portability of applications. This in turn becomes a requirement for a standardized set of  
9907 syntax and semantics for operations commonly found on many operating systems.

9908 The following sections list the perceived requirements for application portability.

#### 9909 **D.1.1 Configuration Interrogation**

9910 An application must be able to determine whether and how certain optional features are  
9911 provided and to identify the system upon which it is running, so that it may appropriately adapt  
9912 to its environment.

9913 Applications must have sufficient information to adapt to varying behaviors of the system.

#### 9914 **D.1.2 Process Management**

9915 An application must be able to manage itself, either as a single process or as multiple processes.  
9916 Applications must be able to manage other processes when appropriate.

9917 Applications must be able to identify, control, create, and delete processes, and there must be  
9918 communication of information between processes and to and from the system.

9919 Applications must be able to use multiple flows of control with a process (threads) and  
9920 synchronize operations between these flows of control.

#### 9921 **D.1.3 Access to Data**

9922 Applications must be able to operate on the data stored on the system, access it, and transmit it  
9923 to other applications. Information must have protection from unauthorized or accidental access  
9924 or modification.

#### 9925 **D.1.4 Access to the Environment**

9926 Applications must be able to access the external environment to communicate their input and  
9927 results.

#### 9928 **D.1.5 Access to Determinism and Performance Enhancements**

9929 Applications must have sufficient control of resource allocation to ensure the timeliness of  
9930 interactions with external objects.

#### 9931 **D.1.6 Operating System-Dependent Profile**

9932 The capabilities of the operating system may make certain optional characteristics of the base  
9933 language in effect no longer optional, and this should be specified.

**9934 D.1.7 I/O Interaction**

9935 The interaction between the C language I/O subsystem (*stdio*) and the I/O subsystem of  
9936 IEEE Std. 1003.1-200x must be specified.

**9937 D.1.8 Internationalization Interaction**

9938 The effects of the environment of IEEE Std. 1003.1-200x on the internationalization facilities of  
9939 the C language must be specified.

**9940 D.1.9 C-Language Extensions**

9941 Certain functions in the C language must be extended to support the additional capabilities  
9942 provided by IEEE Std. 1003.1-200x.

**9943 D.1.10 Command Language**

9944 Users should be able to define procedures that combine simple tools and/or applications into  
9945 higher-level components that perform to the specific needs of the user. The user should be able  
9946 to store, recall, use, and modify these procedures. These procedures should employ a powerful  
9947 command language that is used for recurring tasks in portable applications (scripts) in the same  
9948 way that it is used interactively to accomplish one-time tasks. The language and the utilities that  
9949 it uses must be consistent between systems to reduce errors and retraining.

**9950 D.1.11 Interactive Facilities**

9951 Use the system to accomplish individual tasks at an interactive terminal. The interface should be  
9952 consistent, intuitive, and offer usability enhancements to increase the productivity of terminal  
9953 users, reduce errors, and minimize retraining costs. Online documentation or usage assistance  
9954 should be available.

**9955 D.1.12 Accomplish Multiple Tasks Simultaneously**

9956 Access applications and interactive facilities from a single terminal without requiring serial  
9957 execution: switch between multiple interactive tasks; schedule one-time or periodic background  
9958 work; display the status of all work in progress or scheduled; influence the priority scheduling of  
9959 work, when authorized.

**9960 D.1.13 Complex Data Manipulation**

9961 Manipulate data in files in complex ways: sort, merge, compare, translate, edit, format, pattern  
9962 match, select subsets (strings, columns, fields, rows, and so on). These facilities should be  
9963 available to both portable applications and interactive users.

**9964 D.1.14 File Hierarchy Manipulation**

9965 Create, delete, move/rename, copy, backup/archive, and display files and directories. These  
9966 facilities should be available to both portable applications and interactive users.

**9967 D.1.15 Locale Configuration**

9968 Customize applications and interactive sessions for the cultural and language conventions of the  
9969 user. Employ a wide variety of standard character encodings. These facilities should be available  
9970 to both portable applications and interactive users.

**9971 D.1.16 Inter-User Communication**

9972 Send messages or transfer files to other users on the same system or other systems on a network.  
9973 These facilities should be available to both portable applications and interactive users.

**9974 D.1.17 System Environment**

9975 Display information about the status of the system (activities of users and their interactive and  
9976 background work, file system utilization, system time, configuration, and presence of optional  
9977 facilities) and the environment of the user (terminal characteristics, and so on). Inform the  
9978 system operator/administrator of problems. Control access to user files and other resources.

**9979 D.1.18 Printing**

9980 Output files on a variety of output device classes, accessing devices on local or network-  
9981 connected systems. Control (or influence) the formatting, priority scheduling, and output  
9982 distribution of work. These facilities should be available to both portable applications and  
9983 interactive users.

**9984 D.1.19 Software Development**

9985 Develop (create and manage source files, compile/interpret, debug) portable open systems  
9986 applications and package them for distribution to, and updating of, other systems.

**9987 D.2 Portability Capabilities**

9988 This section describes the significant portability capabilities of IEEE Std. 1003.1-200x and  
9989 indicates how the user requirements listed in Section D.1 (on page 3557) are addressed. The  
9990 capabilities are listed in the same format as the preceding user requirements; they are  
9991 summarized below:

- 9992 • Configuration Interrogation
- 9993 • Process Management
- 9994 • Access to Data
- 9995 • Access to the Environment
- 9996 • Access to Determinism and Performance Enhancements
- 9997 • Operating System-Dependent Profile
- 9998 • I/O Interaction
- 9999 • Internationalization Interaction
- 10000 • C-Language Extensions
- 10001 • Command Language
- 10002 • Interactive Facilities

- 10003 • Accomplish Multiple Tasks Simultaneously
- 10004 • Complex Data Manipulation
- 10005 • File Hierarchy Manipulation
- 10006 • Locale Configuration
- 10007 • Inter-User Communication
- 10008 • System Environment
- 10009 • Printing
- 10010 • Software Development

### 10011 D.2.1 Configuration Interrogation

10012 The *uname()* operation provides basic identification of the system. The *sysconf()*, *pathconf()*, and  
 10013 *fpathconf()* functions and the *getconf* utility provide means to interrogate the implementation to  
 10014 determine how to adapt to the environment in which it is running. These values can be either  
 10015 static (indicating that all instances of the implementation have the same value) or dynamic  
 10016 (indicating that different instances of the implementation have the different values, or that the  
 10017 value may vary for other reasons, such as reconfiguration).

### 10018 Unsatisfied Requirements

10019 None directly. However, as new areas are added, there will be a need for additional capability in  
 10020 this area.

### 10021 D.2.2 Process Management

10022 The *fork()*, *exec* family, and *spawn()* functions provide for the creation of new processes or the  
 10023 insertion of new applications into existing processes. The *\_Exit()*, *\_exit()*, *exit()*, and *abort()*  
 10024 functions allow for the termination of a process by itself. The *wait()* and *waitpid()* functions  
 10025 allow one process to deal with the termination of another.

10026 The *times()* function allows for basic measurement of times used by a process. Various  
 10027 functions, including *fstat()*, *getegid()*, *geteuid()*, *getgid()*, *getrgid()*, *getgrnam()*, *getlogin()*,  
 10028 *getpid()*, *getppid()*, *getpwnam()*, *getpwuid()*, *getuid()*, *lstat()*, and *stat()*, provide for access to the  
 10029 identifiers of processes and the identifiers and names of owners of processes (and files).

10030 The various functions operating on environment variables provide for communication of  
 10031 information (primarily user-configurable defaults) from a parent to child processes.

10032 The operations on the current working directory control and interrogate the directory from  
 10033 which relative file name searches start. The *umask()* function controls the default protections  
 10034 applied to files created by the process.

10035 The *alarm()*, *pause()*, *sleep()*, *ualarm()*, and *usleep()* operations allow the process to suspend until  
 10036 a timer has expired or to be notified when a period of time has elapsed. The *time()* operation  
 10037 interrogates the current time and date.

10038 The signal mechanism provides for communication of events either from other processes or  
 10039 from the environment to the application, and the means for the application to control the effect  
 10040 of these events. The mechanism provides for external termination of a process and for a process  
 10041 to suspend until an event occurs. The mechanism also provides for a value to be associated with  
 10042 an event.

10043 Job control provides a means to group processes and control them as groups, and to control their  
 10044 access to the function between the user and the system (the *controlling terminal*). It also provides  
 10045 the means to suspend and resume processes.

10046 The Process Scheduling option provides control of the scheduling and priority of a process.

10047 The Message Passing option provides a means for interprocess communication involving small  
 10048 amounts of data.

10049 The Memory Management facilities provide control of memory resources and for the sharing of  
 10050 memory. This functionality is mandatory on XSI-conformant systems.

10051 The Threads facilities provide multiple flows of control with a process (threads),  
 10052 synchronization between threads, association of data with threads, and controlled cancelation of  
 10053 threads.

10054 The XSI interprocess communications functionality provide an alternate set of facilities to  
 10055 manipulate semaphores, message queues, and shared memory. These are provided on XSI-  
 10056 conformant systems to support portable applications developed to run on UNIX systems.

### 10057 **D.2.3 Access to Data**

10058 The *open()*, *close()*, *fclose()*, *fopen()*, and *pipe()* functions provide for access to files and data.  
 10059 Such files may be regular files, interprocess data channels (pipes), or devices. Additional types  
 10060 of objects in the file system are permitted and are being contemplated for standardization.

10061 The *access()*, *chmod()*, *chown()*, *dup()*, *dup2()*, *fchmod()*, *fcntl()*, *fstat()*, *ftruncate()*, *lstat()*,  
 10062 *readlink()*, *realpath()*, *stat()*, and *utime()* functions allow for control and interrogation of file and  
 10063 file-related objects, (including symbolic links) and their ownership, protections, and timestamps.

10064 The *fgetc()*, *fputc()*, *fread()*, *fseek()*, *fsetpos()*, *fwrite()*, *getc()*, *getch()*, *lseek()*, *putchar()*, *putc()*,  
 10065 *read()*, and *write()* functions provide for data transfer from the application to files (in all their  
 10066 forms).

10067 The *closedir()*, *link()*, *mkdir()*, *opendir()*, *readdir()*, *rename()*, *rmdir()*, *rewinddir()*, and *unlink()*  
 10068 functions provide for a complete set of operations on directories. Directories can arbitrarily  
 10069 contain other directories, and a single file can be mentioned in more than one directory.

10070 The file-locking mechanism provides for advisory locking (protection during transactions) of  
 10071 ranges of bytes (in effect, records) in a file.

10072 The *confstr()*, *fpathconf()*, *pathconf()*, and *sysconf()* functions provide for enquiry as to the  
 10073 behavior of the system where variability is permitted.

10074 The Synchronized Input and Output option provides for assured commitment of data to media.

10075 The Asynchronous Input and Output option provides for initiation and control of asynchronous  
 10076 data transfers.

### 10077 **D.2.4 Access to the Environment**

10078 The operations and types in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 11,  
 10079 General Terminal Interface are provided for access to asynchronous serial devices. The primary  
 10080 intended use for these is the controlling terminal for the application (the interaction point  
 10081 between the user and the system). They are general enough to be used to control any  
 10082 asynchronous serial device. The functions are also general enough to be used with many other  
 10083 device types as a user interface when some emulation is provided.

10084 Less detailed access is provided for other device types, but in many instances an application  
10085 need not know whether an object in the file system is a device or a regular file to operate  
10086 correctly.

10087 **Unsatisfied Requirements**

10088 Detailed control of common device classes, specifically magnetic tape, is not provided.

10089 **D.2.5 Bounded (Realtime) Response**

10090 The Realtime Signals Extension provides queued signals and the prioritization of the handling of  
10091 signals. The SCHED\_FIFO, SCHED\_SPORADIC, and SCHED\_RR scheduling policies provide  
10092 control over processor allocation. The Semaphores option provides high-performance  
10093 synchronization. The Memory Management functions provide memory locking for control of  
10094 memory allocation, file mapping for high-performance, and shared memory for high-  
10095 performance interprocess communication. The Message Passing option provides for interprocess  
10096 communication without being dependent on shared memory.

10097 The Timers option provides a high resolution function called *nanosleep()* with a finer resolution  
10098 than the *sleep()* function.

10099 The Typed Memory Objects option, the Monotonic Clock option, and the Timeouts option  
10100 provide further facilities for applications to use to obtain predictable bounded response.

10101 **D.2.6 Operating System-Dependent Profile**

10102 IEEE Std. 1003.1-200x makes no distinction between text and binary files. The values of  
10103 EXIT\_SUCCESS and EXIT\_FAILURE are further defined.

10104 **Unsatisfied Requirements**

10105 None known, but the ISO C standard may contain some additional options that could be  
10106 specified.

10107 **D.2.7 I/O Interaction**

10108 IEEE Std. 1003.1-200x defines how each of the ISO C standard *stdio* functions interact with the  
10109 POSIX.1 operations, typically specifying the behavior in terms of POSIX.1 operations.

10110 **Unsatisfied Requirements**

10111 None.

10112 **D.2.8 Internationalization Interaction**

10113 The IEEE Std. 1003.1-200x environment operations provide a means to define the environment  
10114 for *setlocale()* and time functions such as *ctime()*. The *tzset()* function is provided to set time  
10115 conversion information.

10116 The *nl\_langinfo()* function is provided as an XSI extension to query locale-specific cultural  
10117 settings.

10118 **Unsatisfied Requirements**

10119 None.

10120 **D.2.9 C-Language Extensions**10121 The *setjmp()* and *longjmp()* functions are not defined to be cognizant of the signal masks defined  
10122 for POSIX.1. The *sigsetjmp()* and *siglongjmp()* functions are provided to fill this gap.10123 The *\_setjmp()* and *\_longjmp()* functions are provided as XSI extensions to support historic  
10124 practice.10125 **Unsatisfied Requirements**

10126 None.

10127 **D.2.10 Command Language**10128 The shell command language, as described in Shell and Utilities volume of  
10129 IEEE Std. 1003.1-200x, Chapter 2, Shell Command Language, is a common language useful in  
10130 batch scripts, through an API to high-level languages (for the C-Language Binding option,  
10131 *system()* and *popen()*) and through an interactive terminal (see the *sh* utility). The shell language  
10132 has many of the characteristics of a high-level language, but it has been designed to be more  
10133 suitable for user terminal entry and includes interactive debugging facilities. Through the use of  
10134 pipelining, many complex commands can be constructed from combinations of data filters and  
10135 other common components. Shell scripts can be created, stored, recalled, and modified by the  
10136 user with simple editors.10137 In addition to the basic shell language, the following utilities offer features that simplify and  
10138 enhance programmatic access to the utilities and provide features normally found only in high-  
10139 level languages: *basename*, *bc*, *command*, *dirname*, *echo*, *env*, *expr*, *false*, *printf*, *read*, *sleep*, *tee*, *test*,  
10140 *time\**,<sup>2</sup> *true*, *wait*, *xargs*, and all of the special built-in utilities in the Shell and Utilities volume of  
10141 IEEE Std. 1003.1-200x, Section 2.15, Special Built-In Utilities .10142 **Unsatisfied Requirements**

10143 None.

10144 **D.2.11 Interactive Facilities**10145 The utilities offer a common style of command-line interface through conformance to the Utility  
10146 Syntax Guidelines (see the Base Definitions volume of IEEE Std. 1003.1-200x, Section 12.2, Utility  
10147 Syntax Guidelines) and the common utility defaults (see the Shell and Utilities volume of  
10148 IEEE Std. 1003.1-200x, Section 1.11, Utility Description Defaults). The *sh* utility offers an  
10149 interactive command-line history and editing facility. The following utilities in the User  
10150 Portability Utilities option have been customized for interactive use: *alias*, *ex*, *fc*, *mailx*, *more*, *talk*,  
10151 *vi*, *unalias*, and *write*; the *man* utility offers online access to system documentation.

10152 \_\_\_\_\_

10153 2. The utilities listed with an asterisk here and later in this section are present only on systems which support the User Portability  
10154 Utilities option. There may be further restrictions on the utilities offered with various configuration option combinations; see the  
10155 individual utility descriptions.



10156 **Unsatisfied Requirements**

10157 The command line interface to individual utilities is as intuitive and consistent as historical  
 10158 practice allows. Work underway based on graphical user interfaces may be more suitable for  
 10159 novice or occasional users of the system.

10160 **D.2.12 Accomplish Multiple Tasks Simultaneously**

10161 The shell command language offers background processing through the asynchronous list  
 10162 command form; see the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.9, Shell  
 10163 Commands. The *nohup* utility makes background processing more robust and usable. The *kill*  
 10164 utility can terminate background jobs. When the User Portability Utilities option is supported,  
 10165 the following utilities allow manipulation of jobs: *bg*, *fg*, and *jobs*. Also, if the User Portability  
 10166 Utilities option is supported, the following can support periodic job scheduling, control, and  
 10167 display: *at*, *batch*, *crontab*, *nice*, *ps*, and *renice*.

10168 **Unsatisfied Requirements**

10169 Terminals with multiple windows may be more suitable for some multi-tasking interactive uses  
 10170 than the job control approach in IEEE Std. 1003.1-200x. See the comments on graphical user  
 10171 interfaces in Section D.2.11 (on page 3564). The *nice* and *renice* utilities do not necessarily take  
 10172 advantage of complex system scheduling algorithms that are supported by the realtime options  
 10173 within IEEE Std. 1003.1-200x.

10174 **D.2.13 Complex Data Manipulation**

10175 The following utilities address user requirements in this area: *asa*, *awk*, *bc*, *cmp*, *comm*, *csplit\**, *cut*,  
 10176 *dd*, *diff*, *ed*, *ex\**, *expand\**, *expr*, *find*, *fold*, *grep*, *head*, *join*, *od*, *paste*, *pr*, *printf*, *sed*, *sort*, *split\**, *tabs\**, *tail*,  
 10177 *tr*, *unexpand\**, *uniq*, *uudecode\**, *uuencode\**, and *wc*.

10178 **Unsatisfied Requirements**

10179 Sophisticated text formatting utilities, such as *troff* or *TeX*, are not included. Standards work in  
 10180 the area of SGML may satisfy this.

10181 **D.2.14 File Hierarchy Manipulation**

10182 The following utilities address user requirements in this area: *basename*, *cd*, *chgrp*, *chmod*, *chown*,  
 10183 *cksum*, *cp*, *dd*, *df\**, *diff*, *dirname*, *du\**, *find*, *ls*, *ln*, *mkdir*, *mkfifo*, *mv*, *patch\**, *pathchk*, *pax*, *pwd*, *rm*, *rmdir*,  
 10184 *test*, and *touch*.

10185 **Unsatisfied Requirements**

10186 Some graphical user interfaces offer more intuitive file manager components that allow file  
 10187 manipulation through the use of icons for novice users.

**10188 D.2.15 Locale Configuration**

10189 The standard utilities are affected by the various *LC\_* variables to achieve locale-dependent  
10190 operation: character classification, collation sequences, regular expressions and shell pattern  
10191 matching, date and time formats, numeric formatting, and monetary formatting. When the  
10192 POSIX2\_LOCALEDEF option is supported, applications can provide their own locale definition  
10193 files. The following utilities address user requirements in this area: *date*, *ed*, *ex\**, *find*, *grep*, *locale*,  
10194 *localedef*, *more\**, *sed*, *sh*, *sort*, *tr*, *uniq*, and *vi\**.

10195 The *iconv()*, *iconv\_close()*, and *iconv\_open()* functions are available to allow an application to  
10196 convert character data between supported character sets.

10197 The *genccat* utility and the *catopen()*, *catclose()*, and *catgets()* functions for message catalog  
10198 manipulation are available on XSI-conformant systems.

**10199 Unsatisfied Requirements**

10200 Some aspects of multi-byte character and state-encoded character encodings have not yet been  
10201 addressed. The C-language functions, such as *getopt()*, are generally limited to single-byte  
10202 characters. The effect of the *LC\_MESSAGES* variable on message formats is only suggested at  
10203 this time.

**10204 D.2.16 Inter-User Communication**

10205 The following utilities address user requirements in this area: *cksum*, *mailx\**, *mesg\**, *patch\**, *pax*,  
10206 *talk\**, *uudecode\**, *uuencode\**, *who\**, and *write\**.

10207 The historical UUCP utilities are included on XSI-conformant systems.

**10208 Unsatisfied Requirements**

10209 None.

**10210 D.2.17 System Environment**

10211 The following utilities address user requirements in this area: *chgrp*, *chmod*, *chown*, *df\**, *du\**, *env*,  
10212 *getconf*, *id*, *logger*, *logname*, *mesg\**, *newgrp\**, *ps\**, *stty*, *tput\**, *tty*, *umask*, *uname*, and *who\**.

10213 The *closelog()*, *openlog()*, *setlogmask()*, and *syslog()* functions provide System Logging facilities  
10214 on XSI-conformant systems; these are analogous to the *logger* utility.

**10215 Unsatisfied Requirements**

10216 None.

**10217 D.2.18 Printing**

10218 The following utilities address user requirements in this area: *pr* and *lp*.

10219 **Unsatisfied Requirements**

10220 There are no features to control the formatting or scheduling of the print jobs.

10221 **D.2.19 Software Development**

10222 The following utilities address user requirements in this area: *ar*, *asa*, *awk*, *c99*, *ctags\**, *fort77*,  
10223 *getconf*, *getopts*, *lex*, *localedef*, *make*, *nm\**, *od*, *patch\**, *pax*, *strings\**, *strip*, *time\**, and *yacc*.

10224 The *system()*, *popen()*, *pclose()*, *regcomp()*, *regexec()*, *regerror()*, *regfree()*, *fnmatch()*, *getopt()*,  
10225 *glob()*, *globfree()*, *wordexp()*, and *wordfree()* functions allow C-language programmers to access  
10226 some of the interfaces used by the utilities, such as argument processing, regular expressions,  
10227 and pattern matching.

10228 The SCCS source-code control system utilities are available on systems supporting the XSI  
10229 Development option.

10230 **Unsatisfied Requirements**

10231 There are no language-specific development tools related to languages other than C and  
10232 FORTRAN. The C tools are more complete and varied than the FORTRAN tools. There is no  
10233 data dictionary or other CASE-like development tools.

10234 **D.2.20 Future Growth**

10235 It is arguable whether or not all functionality to support applications is potentially within the  
10236 scope of IEEE Std. 1003.1-200x. As a simple matter of practicality, it cannot be. Areas such as  
10237 general networking, graphics, application domain-specific functionality, windowing, and so on,  
10238 should be in unique standards. As such, they are properly “Unsatisfied Requirements” in terms  
10239 of providing fully portable applications, but ones which are outside the scope of  
10240 IEEE Std. 1003.1-200x.

10241 **D.3 Profiling Considerations**

10242 This section offers guidance to writers of profiles on how the configurable options, limits, and  
10243 optional behavior of IEEE Std. 1003.1-200x should be cited in profiles. Profile writers should  
10244 consult the general guidance in POSIX.0 when writing POSIX Standardized Profiles.

10245 The information in this section is an inclusive list of features that should be considered by profile  
10246 writers. Further subsetting of IEEE Std. 1003.1-200x, including the specification of behavior  
10247 currently described as unspecified, undefined, implementation-defined, or with the verbs “may”  
10248 or “need not” violates the intent of the developers of IEEE Std. 1003.1-200x and the guidelines of  
10249 ISO/IEC TR 10000-1. A set of profiling option groups is described in the Base Definitions  
10250 volume of IEEE Std. 1003.1-200x, based on the IEEE Std. 1003.13-1998 options, with the addition  
10251 of a new profiling option group called `_POSIX_NETWORKING`.

**10252 D.3.1 Configuration Options**

10253 There are two set of options suggested by IEEE Std. 1003.1-200x: those for POSIX-conforming  
10254 systems and those for X/Open System Interface (XSI) conformance. The requirements for XSI  
10255 conformance are documented in the Base Definitions volume of IEEE Std. 1003.1-200x and not  
10256 discussed further here, as they superset the POSIX conformance requirements.

**10257 D.3.2 Configuration Options (Shell and Utilities)**

10258 There are three broad optional configurations for the Shell and Utilities volume of  
10259 IEEE Std. 1003.1-200x: basic execution system, development system, and user portability  
10260 interactive system. The options to support these, and other minor configuration options, are  
10261 listed in the Base Definitions volume of IEEE Std. 1003.1-200x, Chapter 2, Conformance. Profile  
10262 writers should consult the following list and the comments concerning user requirements  
10263 addressed by various components in Section D.2 (on page 3560).

**10264 POSIX2\_UPE**

10265 The system supports the User Portability Utilities option.

10266 This option is a requirement for a user portability interactive system. It is required  
10267 frequently except for those systems, such as embedded realtime or dedicated application  
10268 systems, that support little or no interactive time-sharing work by users or operators. XSI-  
10269 conformant systems support this option.

**10270 POSIX2\_SW\_DEV**

10271 The system supports the Software Development Utilities option.

10272 This option is required by many systems, even those in which actual software development  
10273 does not occur. The *make* utility, in particular, is required by many application software  
10274 packages as they are installed onto the system. If POSIX2\_C\_DEV is supported,  
10275 POSIX2\_SW\_DEV is almost a mandatory requirement because of *ar* and *make*.

**10276 POSIX2\_C\_BIND**

10277 The system supports the C-Language Bindings option.

10278 This option is required on some systems developing complex C applications or on any  
10279 system installing C applications in source form that require the functions in this option. The  
10280 *system()* and *popen()* functions, in particular, are widely used by applications; the others are  
10281 rather more specialized.

**10282 POSIX2\_C\_DEV**

10283 The system supports the C-Language Development Utilities option.

10284 This option is required by many systems, even those in which actual C-language software  
10285 development does not occur. The *c99* utility, in particular, is required by many application  
10286 software packages as they are installed onto the system. The *lex* and *yacc* utilities are used  
10287 less frequently.

**10288 POSIX2\_FORT\_DEV**

10289 The system supports the FORTRAN Development Utilities option

10290 As with C, this option is needed on any system developing or installing FORTRAN  
10291 applications in source form.

**10292 POSIX2\_FORT\_RUN**

10293 The system supports the FORTRAN Runtime Utilities option.

10294 This option is required for some FORTRAN applications that need the *asa* utility to convert  
10295 Hollerith printing statement output. It is unknown how frequently this occurs.

- 10296 POSIX2\_LOCALEDEF  
 10297 The system supports the creation of locales.
- 10298 This option is needed if applications require their own customized locale definitions to  
 10299 operate. It is presently unknown whether many applications are dependent on this.  
 10300 However, the option is virtually mandatory for systems in which internationalized  
 10301 applications are developed.
- 10302 XSI-conformant systems support this option.
- 10303 POSIX2\_PBS  
 10304 The system supports the Batch Environment option.
- 10305 POSIX2\_PBS\_ACCOUNTING  
 10306 The system supports the optional feature of accounting within the Batch Environment  
 10307 option. It will be required in servers that implement the optional feature of accounting.
- 10308 POSIX2\_PBS\_CHECKPOINT  
 10309 The systems supports the optional feature of checkpoint/restart within the Batch  
 10310 Environment option.
- 10311 POSIX2\_PBS\_LOCATE  
 10312 The system supports the optional feature of locating batch jobs within the Batch  
 10313 Environment option.
- 10314 POSIX2\_PBS\_MESSAGE  
 10315 The system supports the optional feature of sending messages to batch jobs within the  
 10316 Batch Environment option.
- 10317 POSIX2\_PBS\_TRACK  
 10318 The system supports the optional feature of tracking batch jobs within the Batch  
 10319 Environment option.
- 10320 POSIX2\_CHAR\_TERM  
 10321 The system supports at least one terminal type capable of all operations described in  
 10322 IEEE Std. 1003.1-200x.
- 10323 On systems with POSIX2\_UPE, this option is almost always required. It was developed  
 10324 solely to allow certain specialized vendors and user applications to bypass the requirement  
 10325 for general-purpose asynchronous terminal support. For example, an application and  
 10326 system that was suitable for block-mode terminals, such as IBM 3270s, would not need this  
 10327 option.
- 10328 XSI-conformant systems support this option.

### 10329 D.3.3 Configurable Limits

- 10330 Very few of the limits need to be increased for profiles. No profile can cite lower values.
- 10331 {POSIX2\_BC\_BASE\_MAX}  
 10332 {POSIX2\_BC\_DIM\_MAX}  
 10333 {POSIX2\_BC\_SCALE\_MAX}  
 10334 {POSIX2\_BC\_STRING\_MAX}
- 10335 No increase is anticipated for any of these *bc* values, except for very specialized applications  
 10336 involving huge numbers.
- 10337 {POSIX2\_COLL\_WEIGHTS\_MAX}
- 10338 Some natural languages with complex collation requirements require an increase from the  
 10339 default 2 to 4; no higher numbers are anticipated.

10340 {POSIX2\_EXPR\_NEST\_MAX}  
 10341 No increase is anticipated.

10342 {POSIX2\_LINE\_MAX}  
 10343 This number is much larger than most historical applications have been able to use. At some  
 10344 future time, applications may be rewritten to take advantage of even larger values.

10345 {POSIX2\_RE\_DUP\_MAX}  
 10346 No increase is anticipated.

10347 {POSIX2\_VERSION}  
 10348 This is actually not a limit, but a standard version stamp. Generally, a profile should specify  
 10349 Shell and Utilities volume of IEEE Std. 1003.1-200x, Chapter 2, Shell Command Language by  
 10350 name in the normative references section, not this value.

#### 10351 **D.3.4 Configuration Options (System Interfaces)**

10352 {NGROUPS\_MAX}  
 10353 A non-zero value indicates that the implementation supports supplementary groups.

10354 This option is needed where there is a large amount of shared use of files, but where a  
 10355 certain amount of protection is needed. Many profiles<sup>3</sup> are known to require this option; it  
 10356 should only be required if needed, but it should never be prohibited.

10357 \_POSIX\_ADVISORY\_INFO  
 10358 The system provides advisory information for file management.

10359 This option allows the application to specify advisory information that can be used to  
 10360 achieve better or even deterministic response time in file manager or input and output  
 10361 operations.

10362 \_POSIX\_ASYNCHRONOUS\_IO  
 10363 The system provides concurrent process execution and input and output transfers.

10364 This option was created to support historical systems that did not provide the feature. It  
 10365 should only be required if needed, but it should never be prohibited.

10366 \_POSIX\_BARRIERS  
 10367 The system supports barrier synchronization.

10368 This option was created to allow efficient synchronization of multiple parallel threads in  
 10369 multi-processor systems in which the operation is supported in part by the hardware  
 10370 architecture.

10371 \_POSIX\_CHOWN\_RESTRICTED  
 10372 The system restricts the right to “give away” files to other users.

10373 This option should be carefully investigated before it is required. Some applications expect  
 10374 that they can change the ownership of files in this way. It is provided where either security  
 10375 or system account requirements cause this ability to be a problem. It is also known to be  
 10376 specified in many profiles.

10377 \_\_\_\_\_  
 10378 3. There are no formally approved profiles of IEEE Std. 1003.1-200x at the time of publication; the reference here is to various  
 10379 profiles generated by private bodies or governments.

- 10380     \_POSIX\_CLOCK\_SELECTION  
10381         The system supports the Clock Selection option.
- 10382         This option allows applications to request a high resolution sleep in order to suspend a  
10383         thread during a relative time interval, or until an absolute time value, using the desired  
10384         clock. It also allows the application to select the clock used in a *pthread\_cond\_timedwait()*  
10385         function call.
- 10386     \_POSIX\_CPUTIME  
10387         The system supports the Process CPU-Time Clocks option.
- 10388         This option allows applications to use a new clock that measures the execution times of  
10389         processes or threads, and the possibility to create timers based upon these clocks, for  
10390         runtime detection (and treatment) of execution time overruns.
- 10391     \_POSIX\_FSYNC  
10392         The system supports file synchronization requests.
- 10393         This option was created to support historical systems that did not provide the feature.  
10394         Applications that are expecting guaranteed completion of their input and output operations  
10395         should require the *\_POSIX\_SYNC\_IO* option. This option should never be prohibited.
- 10396         XSI-conformant systems support this option.
- 10397     \_POSIX\_IPV6  
10398         The system supports facilities related to Internet Protocol Version 6 (IPv6).
- 10399         This option was created to allow systems to transition to IPv6.
- 10400     \_POSIX\_JOB\_CONTROL  
10401         Job control facilities are mandatory in IEEE Std. 1003.1-200x.
- 10402         The option was created primarily to support historical systems that did not provide the  
10403         feature. Many existing profiles now require it; it should only be required if needed, but it  
10404         should never be prohibited. Most applications that use it can run when it is not present,  
10405         although with a degraded level of user convenience.
- 10406     \_POSIX\_MAPPED\_FILES  
10407         The system supports the mapping of regular files into the process address space.
- 10408         XSI-conformant systems support this option.
- 10409         Both this option and the Shared Memory Objects option provide shared access to memory  
10410         objects in the process address space. The functions defined under this option provide the  
10411         functionality of existing practice for mapping regular files. This functionality was deemed  
10412         unnecessary, if not inappropriate, for embedded systems applications and, hence, is  
10413         provided under this option. It should only be required if needed, but it should never be  
10414         prohibited.
- 10415     \_POSIX\_MEMLOCK  
10416         The system supports the locking of the address space.
- 10417         This option was created to support historical systems that did not provide the feature. It  
10418         should only be required if needed, but it should never be prohibited.
- 10419     \_POSIX\_MEMLOCK\_RANGE  
10420         The system supports the locking of specific ranges of the address space.
- 10421         For applications that have well-defined sections that need to be locked and others that do  
10422         not, IEEE Std. 1003.1-200x supports an optional set of functions to lock or unlock a range of  
10423         process addresses. The following are two reasons for having a means to lock down a

- 10424 specific range:
- 10425 1. An asynchronous event handler function that must respond to external events in a  
10426 deterministic manner such that page faults cannot be tolerated
- 10427 2. An input/output “buffer” area that is the target for direct-to-process I/O, and the  
10428 overhead of implicit locking and unlocking for each I/O call cannot be tolerated
- 10429 It should only be required if needed, but it should never be prohibited.
- 10430 \_POSIX\_MEMORY\_PROTECTION  
10431 The system supports memory protection.
- 10432 XSI-conformant systems support this option.
- 10433 The provision of this option typically imposes additional hardware requirements. It should  
10434 never be prohibited.
- 10435 \_POSIX\_PRIORITIZED\_IO  
10436 The system provides prioritization for input and output operations.
- 10437 The use of this option may interfere with the ability of the system to optimize input and  
10438 output throughput. It should only be required if needed, but it should never be prohibited.
- 10439 \_POSIX\_MESSAGE\_PASSING  
10440 The system supports the passing of messages between processes.
- 10441 This option was created to support historical systems that did not provide the feature. The  
10442 functionality adds a high-performance XSI interprocess communication facility for local  
10443 communication. It should only be required if needed, but it should never be prohibited.
- 10444 \_POSIX\_MONOTONIC\_CLOCK  
10445 The system supports the Monotonic Clock option.
- 10446 This option allows realtime applications to rely on a monotonically increasing clock that  
10447 does not jump backwards, and whose value does not change except for the regular ticking  
10448 of the clock.
- 10449 \_POSIX\_PRIORITY\_SCHEDULING  
10450 The system provides priority-based process scheduling.
- 10451 Support of this option provides predictable scheduling behavior, allowing applications to  
10452 determine the order in which processes that are ready to run are granted access to a  
10453 processor. It should only be required if needed, but it should never be prohibited.
- 10454 \_POSIX\_REALTIME\_SIGNALS  
10455 The system provides prioritized, queued signals with associated data values.
- 10456 This option was created to support historical systems that did not provide the features. It  
10457 should only be required if needed, but it should never be prohibited.
- 10458 \_POSIX\_REGEX  
10459 Support for regular expression facilities are mandatory in IEEE Std. 1003.1-200x.
- 10460 \_POSIX\_SAVED\_IDS  
10461 Support for this feature is mandatory in IEEE Std. 1003.1-200x.
- 10462 Certain classes of applications rely on it for proper operation, and there is no alternative  
10463 short of giving the application root privileges on most implementations that did not provide  
10464 \_POSIX\_SAVED\_IDS.



- 10465 \_POSIX\_SEMAPHORES  
10466 The system provides counting semaphores.
- 10467 This option was created to support historical systems that did not provide the feature. It  
10468 should only be required if needed, but it should never be prohibited.
- 10469 \_POSIX\_SHARED\_MEMORY\_OBJECTS  
10470 The system supports the mapping of shared memory objects into the process address space.
- 10471 Both this option and the Memory Mapped Files option provide shared access to memory  
10472 objects in the process address space. The functions defined under this option provide the  
10473 functionality of existing practice for shared memory objects. This functionality was deemed  
10474 appropriate for embedded systems applications and, hence, is provided under this option. It  
10475 should only be required if needed, but it should never be prohibited.
- 10476 \_POSIX\_SHELL  
10477 Support for the *sh* utility command line interpreter is mandatory in IEEE Std. 1003.1-200x.
- 10478 \_POSIX\_SPAWN  
10479 The system supports the spawn option.
- 10480 This option provides applications with an efficient mechanism to spawn execution of a new  
10481 process.
- 10482 \_POSIX\_SPINLOCKS  
10483 The system supports spin locks.
- 10484 This option was created to support a simple and efficient synchronization mechanism for  
10485 threads executing in multi-processor systems.
- 10486 \_POSIX\_SPORADIC\_SERVER  
10487 The system supports the sporadic server scheduling policy.
- 10488 This option provides applications with a new scheduling policy for scheduling aperiodic  
10489 processes or threads in hard realtime applications.
- 10490 \_POSIX\_SYNCHRONIZED\_IO  
10491 The system supports guaranteed file synchronization.
- 10492 This option was created to support historical systems that did not provide the feature.  
10493 Applications that are expecting guaranteed completion of their input and output operations  
10494 should require this option, rather than the File Synchronization option. It should only be  
10495 required if needed, but it should never be prohibited.
- 10496 \_POSIX\_THREADS  
10497 The system supports multiple threads of control within a single process.
- 10498 This option was created to support historical systems that did not provide the feature.  
10499 Applications written assuming a multi-threaded environment would be expected to require  
10500 this option. It should only be required if needed, but it should never be prohibited.
- 10501 XSI-conformant systems support this option.
- 10502 \_POSIX\_THREAD\_ATTR\_STACKADDR  
10503 The system supports specification of the stack address for a created thread.
- 10504 Applications may take advantage of support of this option for performance benefits, but  
10505 dependence on this feature should be minimized. This option should never be prohibited.
- 10506 XSI-conformant systems support this option.

- 10507 \_POSIX\_THREAD\_ATTR\_STACKSIZE  
10508 The system supports specification of the stack size for a created thread.
- 10509 Applications may require this option in order to ensure proper execution, but such usage  
10510 limits portability and dependence on this feature should be minimized. It should only be  
10511 required if needed, but it should never be prohibited.
- 10512 XSI-conformant systems support this option.
- 10513 \_POSIX\_THREAD\_PRIORITY\_SCHEDULING  
10514 The system provides priority-based thread scheduling.
- 10515 Support of this option provides predictable scheduling behavior, allowing applications to  
10516 determine the order in which threads that are ready to run are granted access to a processor.  
10517 It should only be required if needed, but it should never be prohibited.
- 10518 \_POSIX\_THREAD\_PRIO\_INHERIT  
10519 The system provides mutual exclusion operations with priority inheritance.
- 10520 Support of this option provides predictable scheduling behavior, allowing applications to  
10521 determine the order in which threads that are ready to run are granted access to a processor.  
10522 It should only be required if needed, but it should never be prohibited.
- 10523 \_POSIX\_THREAD\_PRIO\_PROTECT  
10524 The system supports a priority ceiling emulation protocol for mutual exclusion operations.
- 10525 Support of this option provides predictable scheduling behavior, allowing applications to  
10526 determine the order in which threads that are ready to run are granted access to a processor.  
10527 It should only be required if needed, but it should never be prohibited.
- 10528 \_POSIX\_THREAD\_PROCESS\_SHARED  
10529 The system provides shared access among multiple processes to synchronization objects.
- 10530 This option was created to support historical systems that did not provide the feature. It  
10531 should only be required if needed, but it should never be prohibited.
- 10532 XSI-conformant systems support this option.
- 10533 \_POSIX\_THREAD\_SAFE\_FUNCTIONS  
10534 The system provides thread-safe versions of all of the POSIX.1 functions.
- 10535 This option is required if the Threads option is supported. This is a separate option because  
10536 thread-safe functions are useful in implementations providing other mechanisms for  
10537 concurrency. It should only be required if needed, but it should never be prohibited.
- 10538 XSI-conformant systems support this option.
- 10539 \_POSIX\_THREAD\_SPORADIC\_SERVER  
10540 The system supports the thread sporadic server scheduling policy.
- 10541 Support for this option provides applications with a new scheduling policy for scheduling  
10542 aperiodic threads in hard realtime applications.
- 10543 \_POSIX\_TIMEOUTS  
10544 The system provides timeouts for some blocking services.
- 10545 This option was created to provide a timeout capability to system services, thus allowing  
10546 applications to include better error detection, and recovery capabilities.
- 10547 \_POSIX\_TIMERS  
10548 The system provides higher resolution clocks with multiple timers per process.

10549 This option was created to support historical systems that did not provide the features. This  
 10550 option is appropriate for applications requiring higher resolution timestamps or needing to  
 10551 control the timing of multiple activities. It should only be required if needed, but it should  
 10552 never be prohibited.

10553 `_POSIX_TRACE`

10554 The system supports the trace option.

10555 This option was created to allow applications to perform tracing.

10556 `_POSIX_TRACE_EVENT_FILTER`

10557 The system supports the trace event filter option.

10558 This option is dependent on support of the Trace option.

10559 `_POSIX_TRACE_INHERIT`

10560 The system supports the trace inherit option.

10561 This option is dependent on support of the Trace option.

10562 `_POSIX_TRACE_LOG`

10563 The system supports the trace log option.

10564 This option is dependent on support of the Trace option.

10565 `_POSIX_TYPED_MEMORY_OBJECTS`

10566 The system supports typed memory objects.

10567 This option was created to allow realtime applications to access different kinds of physical  
 10568 memory, and allow processes in these applications to share portions of this memory.

### 10569 **D.3.5 Configurable Limits**

10570 In general, the configurable limits in the `<limits.h>` header defined in the Base Definitions  
 10571 volume of IEEE Std. 1003.1-200x have been set to minimal values; many applications or  
 10572 implementations may require larger values. No profile can cite lower values.

10573 `{AIO_LISTIO_MAX}`

10574 The current minimum is likely to be inadequate for most applications. It is expected that  
 10575 this value will be increased by profiles requiring support for list input and output  
 10576 operations.

10577 `{AIO_MAX}`

10578 The current minimum is likely to be inadequate for most applications. It is expected that  
 10579 this value will be increased by profiles requiring support for asynchronous input and  
 10580 output operations.

10581 `{AIO_PRIO_DELTA_MAX}`

10582 The functionality associated with this limit is needed only by sophisticated applications. It  
 10583 is not expected that this limit would need to be increased under a general-purpose profile.

10584 `{ARG_MAX}`

10585 The current minimum is likely to need to be increased for profiles, particularly as larger  
 10586 amounts of information are passed through the environment. Many implementations are  
 10587 believed to support larger values.

10588 `{CHILD_MAX}`

10589 The current minimum is suitable only for systems where a single user is not running  
 10590 applications in parallel. It is significantly too low for any system also requiring windows,  
 10591 and if `_POSIX_JOB_CONTROL` is specified, it should be raised.

- 10592 {CLOCKRES\_MIN}  
10593 It is expected that profiles will require a finer granularity clock, perhaps as fine as 1  $\mu$ s,  
10594 represented by a value of 1 000 for this limit.
- 10595 {DELAYTIMER\_MAX}  
10596 It is believed that most implementations will provide larger values.
- 10597 {LINK\_MAX}  
10598 For most applications and usage, the current minimum is adequate. Many implementations  
10599 have a much larger value, but this should not be used as a basis for raising the value unless  
10600 the applications to be used require it.
- 10601 {LOGIN\_NAME\_MAX}  
10602 This is not actually a limit, but an implementation parameter. No profile should impose a  
10603 requirement on this value.
- 10604 {MAX\_CANON}  
10605 For most purposes, the current minimum is adequate. Unless high-speed burst serial  
10606 devices are used, it should be left as is.
- 10607 {MAX\_INPUT}  
10608 See {MAX\_CANON}.
- 10609 {MQ\_OPEN\_MAX}  
10610 The current minimum should be adequate for most profiles.
- 10611 {MQ\_PRIO\_MAX}  
10612 The current minimum corresponds to the required number of process scheduling priorities.  
10613 Many realtime practitioners believe that the number of message priority levels ought to be  
10614 the same as the number of execution scheduling priorities.
- 10615 {NAME\_MAX}  
10616 Many implementations now support larger values, and many applications and users  
10617 assume that larger names can be used. Many existing profiles also specify a larger value.  
10618 Specifying this value will reduce the number of conforming implementations, although this  
10619 might not be a significant consideration over time. Values greater than 255 should not be  
10620 required.
- 10621 {NGROUPS\_MAX}  
10622 The value selected will typically be 8 or larger.
- 10623 {OPEN\_MAX}  
10624 The historically common value for this has been 20. Many implementations support larger  
10625 values. If applications that use larger values are anticipated, an appropriate value should be  
10626 specified.
- 10627 {PAGESIZE}  
10628 This is not actually a limit, but an implementation parameter. No profile should impose a  
10629 requirement on this value.
- 10630 {PATH\_MAX}  
10631 Historically, the minimum has been either 1024 or indefinite, depending on the  
10632 implementation. Few applications actually require values larger than 256, but some users  
10633 may create file hierarchies that must be accessed with longer paths. This value should only  
10634 be changed if there is a clear requirement.
- 10635 {PIPE\_BUF}  
10636 The current minimum is adequate for most applications. Historically, it has been larger. If  
10637 applications that write single transactions larger than this are anticipated, it should be

10638 increased. Applications that write lines of text larger than this probably do not need it  
10639 increased, as the text line is delimited by a newline.

10640 {POSIX\_VERSION}  
10641 This is actually not a limit, but a standard version stamp. Generally, a profile should specify  
10642 IEEE Std. 1003.1-200x by a name in the normative references section, not this value.

10643 {PTHREAD\_DESTRUCTOR\_ITERATIONS}  
10644 It is unlikely that applications will need larger values to avoid loss of memory resources.

10645 {PTHREAD\_KEYS\_MAX}  
10646 The current value should be adequate for most profiles.

10647 {PTHREAD\_STACK\_MIN}  
10648 This should not be treated as an actual limit, but as an implementation parameter. No  
10649 profile should impose a requirement on this value.

10650 {PTHREAD\_THREADS\_MAX}  
10651 It is believed that most implementations will provide larger values.

10652 {RTSIG\_MAX}  
10653 The current limit was chosen so that the set of POSIX.1 signal numbers can fit within a 32-  
10654 bit field. It is recognized that most existing implementations define many more signals than  
10655 are specified in POSIX.1 and, in fact, many implementations have already exceeded 32  
10656 signals (including the “null signal”). Support of {\_POSIX\_RTSIG\_MAX} additional signals  
10657 may push some implementations over the single 32-bit word line, but is unlikely to push  
10658 any implementations that are already over that line beyond the 64 signal line.

10659 {SEM\_NSEMS\_MAX}  
10660 The current value should be adequate for most profiles.

10661 {SEM\_VALUE\_MAX}  
10662 The current value should be adequate for most profiles.

10663 {SSIZE\_MAX}  
10664 This limit reflects fundamental hardware characteristics (the size of an integer), and should  
10665 not be specified unless it is clearly required. Extreme care should be taken to assure that  
10666 any value that might be specified does not unnecessarily eliminate implementations  
10667 because of accidents of hardware design.

10668 {STREAM\_MAX}  
10669 This limit is very closely related to {OPEN\_MAX}. It should never be larger than  
10670 {OPEN\_MAX}, but could reasonably be smaller for application areas where most files are  
10671 not accessed through *stdio*. Some implementations may limit {STREAM\_MAX} to 20 but  
10672 allow {OPEN\_MAX} to be considerably larger. Such implementations should be allowed for  
10673 if the applications permit.

10674 {TIMER\_MAX}  
10675 The current limit should be adequate for most profiles, but it may need to be larger for  
10676 applications with a large number of asynchronous operations.

10677 {TTY\_NAME\_MAX}  
10678 This is not actually a limit, but an implementation parameter. No profile should impose a  
10679 requirement on this value.

10680 {TZNAME\_MAX}  
10681 The minimum has been historically adequate, but if longer timezone names are anticipated  
10682 (particularly such values as UTC-1), this should be increased.

**10683 D.3.6 Optional Behavior**

10684 In IEEE Std. 1003.1-200x, there are no instances of the terms unspecified, undefined,  
10685 implementation-defined, or with the verbs “may” or “need not”, that the developers of  
10686 IEEE Std. 1003.1-200x anticipate or sanction as suitable for profile or test method citation. All of  
10687 these are merely warnings to portable applications to avoid certain areas that can vary from  
10688 system to system, and even over time on the same system. In many cases, these terms are used  
10689 explicitly to support extensions, but profiles should not anticipate and require such extensions;  
10690 future versions of IEEE Std. 1003.1-200x may do so.

# Index

1

|    |                                         |            |                                     |                  |
|----|-----------------------------------------|------------|-------------------------------------|------------------|
| 2  | /dev/tty .....                          | 3329       | _POSIX_THREAD_PRIO_INHERIT .....    | 3574             |
| 3  | /etc/passwd .....                       | 3342       | _POSIX_THREAD_PRIO_PROTECT .....    | 3574             |
| 4  | <pthread.h> .....                       | 3458       | _POSIX_THREAD_PROCESS_SHARED .....  | 3574             |
| 5  | _asm_builtin_atoi() .....               | 3383       | _POSIX_THREAD_SAFE_FUNCTIONS .....  | 3574             |
| 6  | _exit() .....                           | 3398, 3416 | _POSIX_THREAD_SPORADIC_SERVER ..... | 3574             |
| 7  | _Exit() .....                           | 3561       | _POSIX_TIMEOUTS .....               | 3574             |
| 8  | _exit() .....                           | 3561       | _POSIX_TIMERS .....                 | 3574             |
| 9  | _longjmp() .....                        | 3564       | _POSIX_TRACE .....                  | 3575             |
| 10 | _POSIX_ADVISORY_INFO .....              | 3570       | _POSIX_TRACE_EVENT_FILTER .....     | 3575             |
| 11 | _POSIX_ASYNCHRONOUS_IO .....            | 3570       | _POSIX_TRACE_INHERIT .....          | 3575             |
| 12 | _POSIX_BARRIERS .....                   | 3570       | _POSIX_TRACE_LOG .....              | 3575             |
| 13 | _POSIX_CHOWN_RESTRICTED .....           | 3570       | _POSIX_TYPED_MEMORY_OBJECTS .....   | 3575             |
| 14 | _POSIX_CLOCK_SELECTION .....            | 3571       | _POSIX_TZNAME_MAX .....             | 3360             |
| 15 | _POSIX_CPUTIME .....                    | 3571       | _SC_PAGESIZE .....                  | 3415, 3417       |
| 16 | _POSIX_C_SOURCE .....                   | 3384, 3387 | _setjmp() .....                     | 3564             |
| 17 | _POSIX_FSYNC .....                      | 3571       | __errno() .....                     | 3390             |
| 18 | _POSIX_IPV6 .....                       | 3571       | abort() .....                       | 3561             |
| 19 | _POSIX_JOB_CONTROL .....                | 3571, 3575 | access .....                        | 3562             |
| 20 | _POSIX_MAPPED_FILES .....               | 3571       | access() .....                      | 3342             |
| 21 | _POSIX_MEMLOCK .....                    | 3571       | active trace stream .....           | 3499             |
| 22 | _POSIX_MEMLOCK_RANGE .....              | 3571       | addressing .....                    | 3474             |
| 23 | _POSIX_MEMORY_PROTECTION .....          | 3572       | advisory information .....          | 3402             |
| 24 | _POSIX_MESSAGE_PASSING .....            | 3572       | aio_cancel() .....                  | 3410-3411        |
| 25 | _POSIX_MONOTONIC_CLOCK .....            | 3572       | aio_fsync() .....                   | 3395, 3409       |
| 26 | _POSIX_NETWORKING .....                 | 3567       | AIO_LISTIO_MAX .....                | 3575             |
| 27 | _POSIX_PRIORITIZED_IO .....             | 3572       | AIO_MAX .....                       | 3575             |
| 28 | _POSIX_PRIORITY_SCHEDULING .....        | 3572       | AIO_PRIO_DELTA_MAX .....            | 3575             |
| 29 | _POSIX_REALTIME_SIGNALS .....           | 3572       | aio_read() .....                    | 3411             |
| 30 | _POSIX_REGEX .....                      | 3572       | aio_suspend() .....                 | 3410, 3435       |
| 31 | _POSIX_RTSIG_MAX .....                  | 3392, 3577 | aio_write() .....                   | 3411             |
| 32 | _POSIX_SAVED_IDS .....                  | 3572       | alarm .....                         | 3561             |
| 33 | _POSIX_SEMAPHORES .....                 | 3573       | alarm() .....                       | 3400, 3434       |
| 34 | _POSIX_SHARED_MEMORY_OBJECTS .....      | 3573       | alias .....                         | 3564             |
| 35 | _POSIX_SHELL .....                      | 3573       | alias substitution .....            | 3524             |
| 36 | _POSIX_SOURCE .....                     | 3384       | AND lists .....                     | 3539             |
| 37 | _POSIX_SPAWN .....                      | 3573       | application instrumentation .....   | 3487             |
| 38 | _POSIX_SPINLOCKS .....                  | 3573       | appropriate privilege .....         | 3321             |
| 39 | _POSIX_SPORADIC_SERVER .....            | 3573       | appropriate privileges .....        | 3321, 3333       |
| 40 | _POSIX_SS_REPL_MAX .....                | 3431       | ar .....                            | 3567-3568        |
| 41 | _POSIX_SYNCHRONIZED_IO .....            | 3573       | ARG_MAX .....                       | 3330, 3515, 3575 |
| 42 | _POSIX_SYNC_IO .....                    | 3571       | arithmetic expansion .....          | 3530             |
| 43 | _POSIX_THREADS .....                    | 3573       | asa .....                           | 3565, 3567-3568  |
| 44 | _POSIX_THREAD_ATTR_STACKADDR .....      | 3573       | ASCII .....                         | 3332             |
| 45 | _POSIX_THREAD_ATTR_STACKSIZE .....      | 3574       | async-cancel safety .....           | 3471             |
| 46 | _POSIX_THREAD_PRIORITY_SCHEDULING ..... | 3574       | async-signal-safe .....             | 3399             |
| 47 | .....                                   | 3574       | asynchronous error .....            | 3475             |

|    |                                      |                                   |                                        |                        |
|----|--------------------------------------|-----------------------------------|----------------------------------------|------------------------|
| 48 | asynchronous I/O.....                | 3409, 3562                        | catclose().....                        | 3566                   |
| 49 | asynchronous lists.....              | 3538                              | catgets().....                         | 3566                   |
| 50 | at.....                              | 3565                              | catopen().....                         | 3566                   |
| 51 | atexit().....                        | 3471                              | cd.....                                | 3565                   |
| 52 | atoi().....                          | 3383                              | character.....                         | 3321                   |
| 53 | awk.....                             | 3565, 3567                        | character encoding.....                | 3350                   |
| 54 | background.....                      | 3326-3329, 3371                   | character set.....                     | 3350                   |
| 55 | backslash.....                       | 3522                              | character set description file.....    | 3350                   |
| 56 | barriers.....                        | 3451                              | character set, portable file name..... | 3332                   |
| 57 | barrier_wait().....                  | 3471                              | character, rationale.....              | 3321                   |
| 58 | basename.....                        | 3564-3565                         | CHARCLASS_NAME_MAX.....                | 3355                   |
| 59 | basic regular expression.....        | 3364                              | CHAR_MAX.....                          | 3357                   |
| 60 | batch.....                           | 3565                              | chgrp.....                             | 3520, 3565-3566        |
| 61 | general concepts.....                | 3549                              | CHILD_MAX.....                         | 3501, 3515, 3538, 3575 |
| 62 | batch environment.....               | 3546                              | chmod.....                             | 3520, 3562, 3565-3566  |
| 63 | option definitions.....              | 3547                              | chmod().....                           | 3334                   |
| 64 | batch environment utilities          |                                   | chown.....                             | 3520, 3562, 3565-3566  |
| 65 | common behavior.....                 | 3552                              | chown().....                           | 3334                   |
| 66 | batch services.....                  | 3551                              | chroot().....                          | 3332                   |
| 67 | batch systems                        |                                   | cksum.....                             | 3520-3521, 3565-3566   |
| 68 | historical implementations.....      | 3546                              | clock.....                             | 3431                   |
| 69 | history.....                         | 3546                              | clock tick.....                        | 3322                   |
| 70 | bc.....                              | 3564-3565, 3569                   | clock tick, rationale.....             | 3322                   |
| 71 | BC_BASE_MAX.....                     | 3515                              | CLOCKRES_MIN.....                      | 3576                   |
| 72 | BC_DIM_MAX.....                      | 3515                              | clocks.....                            | 3431                   |
| 73 | BC_SCALE_MAX.....                    | 3515                              | clock_getcpuclid().....                | 3437, 3439             |
| 74 | bg.....                              | 3565                              | clock_nanosleep().....                 | 3435-3436              |
| 75 | bounded response.....                | 3563                              | CLOCK_PROCESS_CPUTIME_ID.....          | 3437, 3439             |
| 76 | bracket expression                   |                                   | CLOCK_REALTIME.....                    | 3431-3434, 3436        |
| 77 | grammar.....                         | 3367                              | CLOCK_THREAD_CPUTIME_ID.....           | 3437, 3439             |
| 78 | BRE                                  |                                   | close.....                             | 3562                   |
| 79 | expression anchoring.....            | 3365                              | close().....                           | 3416                   |
| 80 | grammar lexical conventions.....     | 3367                              | closedir.....                          | 3562                   |
| 81 | matching a collating element.....    | 3364                              | closelog().....                        | 3566                   |
| 82 | matching a single character.....     | 3364                              | cmp.....                               | 3520-3521, 3565        |
| 83 | matching multiple characters.....    | 3364                              | COLL_WEIGHTS_MAX.....                  | 3515                   |
| 84 | ordinary character.....              | 3364                              | column position.....                   | 3322                   |
| 85 | periods.....                         | 3364                              | COLUMNS.....                           | 3360                   |
| 86 | precedence.....                      | 3365                              | comm.....                              | 3565                   |
| 87 | special character.....               | 3364                              | command.....                           | 3322, 3564             |
| 88 | BSD.....                             | 3323, 3326, 3328, 3331, 3370-3372 | command execution.....                 | 3536                   |
| 89 | .....                                | 3391-3394, 3397-3398, 3500        | command language.....                  | 3559, 3564             |
| 90 | C Shell.....                         | 3326, 3328                        | command search.....                    | 3536                   |
| 91 | C-language extensions.....           | 3559, 3564                        | command substitution.....              | 3529                   |
| 92 | c99.....                             | 3567                              | compilation environment.....           | 3384                   |
| 93 | cancelation cleanup handler.....     | 3470-3471                         | complex data manipulation.....         | 3559, 3565             |
| 94 | cancelation cleanup stack.....       | 3470                              | compound commands.....                 | 3539                   |
| 95 | canonical mode input processing..... | 3372                              | concurrent execution.....              | 3342                   |
| 96 | case.....                            | 3539                              | conditional construct                  |                        |
| 97 | case folding.....                    | 3342-3343                         | case.....                              | 3539                   |
| 98 | cat.....                             | 3520-3521                         | if.....                                | 3540                   |



## Index

- 99 configurable limits .....3569, 3575
- 100 configuration interrogation .....3558, 3561
- 101 configuration options .....3568
- 102 shell and utilities .....3568
- 103 system interfaces .....3570
- 104 conformance .....3313, 3317, 3319, 3321, 3344, 3500
- 105 conformance document .....3313
- 106 conformance document, rationale .....3313
- 107 conforming application .....3319, 3364, 3392
- 108 .....3518, 3520, 3522
- 109 conforming application, strictly ...3317, 3320, 3398
- 110 confstr .....3562
- 111 connection indication queue .....3475
- 112 control mode .....3374
- 113 controlling terminal .....3323, 3371, 3562
- 114 core .....3342
- 115 covert channel .....3344
- 116 cp .....3520, 3565
- 117 creat() .....3416, 3520
- 118 crontab .....3565
- 119 CSIZE .....3374
- 120 csplit .....3565
- 121 ctags .....3567
- 122 ctime() .....3563
- 123 cut .....3565
- 124 data access .....3558, 3562
- 125 data type .....3500
- 126 date .....3566
- 127 dd .....3520-3521, 3565
- 128 DELAYTIMER\_MAX .....3576
- 129 determinism .....3558
- 130 device number .....3323
- 131 device, logical .....3329
- 132 df .....3521, 3565-3566
- 133 diff .....3565
- 134 direct I/O .....3323
- 135 directory .....3323
- 136 directory device .....3368
- 137 directory entry .....3323
- 138 directory files .....3368
- 139 directory structure .....3368
- 140 directory, root .....3332
- 141 dirname .....3564-3565
- 142 display .....3323
- 143 dot .....3324
- 144 dot-dot .....3324, 3331, 3347
- 145 double-quotes .....3522
- 146 du .....3521, 3565-3566
- 147 dup .....3562
- 148 dup() .....3416
- 149 dup2 .....3562
- dup2() .....3416
- EBUSY .....3390, 3458
- ECANCELED .....3389
- echo .....3564
- ECHOE .....3374
- ECHOK .....3374
- ECHONL .....3374
- ed .....3565-3566
- EDOM .....3390
- EFAULT .....3388-3389
- effective user ID .....3342
- EFTYPE .....3389
- EILSEQ .....3390
- EINPROGRESS .....3410
- EINTR .....3389, 3391, 3400
- EINVAL .....3389
- ELOOP .....3389
- ENAMETOOLONG .....3389
- endgrent() .....3340
- endpwent() .....3340
- ENOMEM .....3389
- ENOSYS .....3389, 3413
- ENOTSUP .....3389
- ENOTTY .....3369, 3389-3390
- env .....3564, 3566
- environment access .....3558, 3562
- environment variable .....3359
- definition .....3359
- EPERM .....3467
- EPIPE .....3390
- Epoch .....3324, 3332, 3347, 3431
- ERANGE .....3390
- ERASE .....3372
- ERE .....3366
- alternation .....3366
- bracket expression .....3366
- expression anchoring .....3367
- grammar .....3367
- grammar lexical conventions .....3367
- matching a collating element .....3366
- matching a single character .....3366
- matching multiple characters .....3366
- ordinary character .....3366
- periods .....3366
- precedence .....3367
- special character .....3366
- EROFS .....3390
- errno .....3388
- per-thread .....3390
- error handling .....3543
- error numbers .....3388, 3391

|     |                             |                              |                                |                              |
|-----|-----------------------------|------------------------------|--------------------------------|------------------------------|
| 150 | errors                      | 3534                         | filtering of trace event types | 3496                         |
| 151 | escape character            | 3522                         | filtering trace event types    | 3496                         |
| 152 | ex                          | 3564-3566                    | find                           | 3565-3566                    |
| 153 | exec family                 | 3326, 3414, 3471, 3541, 3561 | flockfile()                    | 3391                         |
| 154 | exec()                      | 3493                         | fnmatch()                      | 3567                         |
| 155 | exec, family                | 3516                         | fold                           | 3565                         |
| 156 | executable script           | 3542                         | fopen                          | 3562                         |
| 157 | Execution Time Monitoring   | 3437                         | fopen()                        | 3325, 3520                   |
| 158 | execution time, measurement | 3344                         | for loop                       | 3539                         |
| 159 | exit status                 | 3534                         | foreground                     | 3326-3329, 3370-3371         |
| 160 | exit()                      | 3398, 3561                   | fork()                         | 3326, 3370, 3405, 3414, 3416 |
| 161 | EXIT_FAILURE                | 3563                         | .....                          | 3493, 3501-3502, 3561        |
| 162 | EXIT_SUCCESS                | 3563                         | fort77                         | 3567                         |
| 163 | expand                      | 3565                         | fpathconf                      | 3562                         |
| 164 | expr                        | 3564-3565                    | fpathconf()                    | 3561                         |
| 165 | EXPR_NEST_MAX               | 3515                         | fputc                          | 3562                         |
| 166 | extended regular expression | 3366                         | fread                          | 3562                         |
| 167 | extended security controls  | 3342                         | free()                         | 3399, 3470                   |
| 168 | false                       | 3564                         | fseek                          | 3562                         |
| 169 | fc                          | 3564                         | fseeko()                       | 3500                         |
| 170 | fchmod                      | 3562                         | fsetpos                        | 3562                         |
| 171 | fclose                      | 3562                         | fsetpos()                      | 3500                         |
| 172 | fcntl                       | 3562                         | fstat                          | 3561-3562                    |
| 173 | fcntl()                     | 3369, 3389, 3416, 3419, 3500 | fsync()                        | 3409                         |
| 174 | fcntl() locks               | 3473                         | truncate                       | 3562                         |
| 175 | fdopen()                    | 3416                         | truncate()                     | 3415-3416, 3418              |
| 176 | FD_CLOEXEC                  | 3419                         | ftw()                          | 3520                         |
| 177 | feature test macro          | 3384, 3500                   | function definition command    | 3540                         |
| 178 | fg                          | 3565                         | functions                      |                              |
| 179 | fgetc                       | 3562                         | implementation of              | 3383                         |
| 180 | fgetpos()                   | 3500                         | use of                         | 3383                         |
| 181 | field splitting             | 3532                         | fwrite                         | 3562                         |
| 182 | FIFO                        | 3324, 3331, 3425             | gencat                         | 3566                         |
| 183 | FIFO special file           | 3324                         | general terminal interface     | 3369                         |
| 184 | file                        | 3324                         | getc                           | 3562                         |
| 185 | file access permissions     | 3342                         | getc()                         | 3461                         |
| 186 | file classes                | 3324                         | getch                          | 3562                         |
| 187 | file descriptors            | 3400                         | getconf                        | 3514, 3561, 3566-3567        |
| 188 | file format notation        | 3349                         | getegid                        | 3561                         |
| 189 | file hierarchy              | 3342                         | geteuid                        | 3561                         |
| 190 | file hierarchy manipulation | 3559, 3565                   | getgid                         | 3561                         |
| 191 | file name                   | 3324                         | getgrent()                     | 3340                         |
| 192 | file names                  | 3342                         | getgrgid                       | 3561                         |
| 193 | file permissions            | 3342, 3371                   | getgrgid()                     | 3340, 3462                   |
| 194 | file system                 | 3324                         | getgrnam                       | 3561                         |
| 195 | file system, mounted        | 3330                         | getgrnam()                     | 3340, 3344, 3462             |
| 196 | file system, root           | 3332                         | getgroups()                    | 3334                         |
| 197 | file system, root of        | 3332                         | getlogin                       | 3561                         |
| 198 | file times update           | 3344                         | getopt()                       | 3376, 3566-3567              |
| 199 | file, passwd                | 3331                         | getopts                        | 3567                         |
| 200 | fileno()                    | 3330                         | getpgrp()                      | 3328                         |

## Index

|     |                                   |                  |                                         |                             |
|-----|-----------------------------------|------------------|-----------------------------------------|-----------------------------|
| 201 | getpid                            | 3561             | interfaces                              | 3474                        |
| 202 | getpid()                          | 3400             | internationalization variable           | 3359                        |
| 203 | getppid                           | 3561             | interprocess communication              | 3401                        |
| 204 | getpriority()                     | 3426             | invalid                                 |                             |
| 205 | getpwent()                        | 3340             | use in RE                               | 3363                        |
| 206 | getpwnam                          | 3561             | ioctl()                                 | 3369, 3390                  |
| 207 | getpwnam()                        | 3340, 3344, 3462 | IPC                                     | 3401                        |
| 208 | getpwuid                          | 3561             | ISO C standard                          | 3319, 3322, 3333, 3369      |
| 209 | getpwuid()                        | 3340, 3462       | 3383-3384, 3387, 3390, 3392, 3398, 3500 |                             |
| 210 | getrlimit()                       | 3521             | ISO/IEC 646: 1991 standard              | 3332                        |
| 211 | getrusage()                       | 3439             | ISTRIP                                  | 3373                        |
| 212 | getty                             | 3371             | job control                             | 3326-3329, 3331, 3370-3371  |
| 213 | getuid                            | 3561             | 3392, 3398, 3562                        |                             |
| 214 | getuid()                          | 3400, 3501       | job control, implementing applications  | 3328                        |
| 215 | gid_t                             | 3340             | job control, implementing shells        | 3326                        |
| 216 | glob()                            | 3567             | job control, implementing systems       | 3328                        |
| 217 | globfree()                        | 3567             | jobs                                    | 3565                        |
| 218 | gmtime()                          | 3333             | join                                    | 3565                        |
| 219 | grep                              | 3565-3566        | kernel                                  | 3329                        |
| 220 | group database                    | 3325             | kernel entity                           | 3464                        |
| 221 | group database access             | 3340             | kill                                    | 3565                        |
| 222 | group file                        | 3325             | kill()                                  | 3391-3393, 3396, 3398, 3501 |
| 223 | grouping commands                 | 3539             | last close                              | 3416                        |
| 224 | head                              | 3565             | lchown()                                | 3334                        |
| 225 | headers                           | 3378             | LC_COLLATE                              | 3355                        |
| 226 | here-document                     | 3534             | LC_CTYPE                                | 3354                        |
| 227 | historical implementations        | 3325             | LC_MESSAGES                             | 3358                        |
| 228 | hosted implementation             | 3325             | LC_MONETARY                             | 3357                        |
| 229 | ICANON                            | 3372, 3374       | LC_NUMERIC                              | 3357, 3377                  |
| 230 | iconv()                           | 3566             | LC_TIME                                 | 3357                        |
| 231 | iconv_close()                     | 3566             | legacy                                  | 3314                        |
| 232 | iconv_open()                      | 3566             | legacy, rationale                       | 3314                        |
| 233 | id                                | 3566             | lex                                     | 3567-3568                   |
| 234 | if                                | 3540             | library routine                         | 3329                        |
| 235 | implementation                    | 3325             | LINES                                   | 3360                        |
| 236 | implementation, historical        | 3325             | LINE_MAX                                | 3330, 3339, 3515            |
| 237 | implementation, hosted            | 3325             | link                                    | 3562                        |
| 238 | implementation, native            | 3330             | link()                                  | 3323, 3334                  |
| 239 | implementation, specific          | 3325             | LINK_MAX                                | 3515, 3576                  |
| 240 | implementation-defined            | 3313-3314        | lio_listio()                            | 3395, 3410                  |
| 241 | implementation-defined, rationale | 3313             | lists                                   | 3538                        |
| 242 | incomplete path name              | 3326             | ln                                      | 3520, 3565                  |
| 243 | input file descriptor             |                  | local mode                              | 3374                        |
| 244 | duplication                       | 3534             | locale                                  | 3353, 3566                  |
| 245 | input mode                        | 3373             | definition example                      | 3358                        |
| 246 | input processing                  | 3372             | definition grammar                      | 3358                        |
| 247 | canonical mode                    | 3372             | grammar                                 | 3358                        |
| 248 | non-canonical mode                | 3372             | lexical conventions                     | 3358                        |
| 249 | inter-user communication          | 3560, 3566       | locale configuration                    | 3560, 3566                  |
| 250 | interactive facilities            | 3559, 3564       | locale definition                       | 3353                        |
| 251 | interface characteristics         | 3370             | localedef                               | 3566-3567                   |

|     |                        |                              |                                     |                             |
|-----|------------------------|------------------------------|-------------------------------------|-----------------------------|
| 252 | localtime()            | 3333, 3461                   | mount point                         | 3330                        |
| 253 | logger                 | 3566                         | mount()                             | 3330                        |
| 254 | logical device         | 3329                         | mounted file system                 | 3330                        |
| 255 | LOGIN_NAME_MAX         | 3576                         | mprotect()                          | 3416                        |
| 256 | LOGNAME                | 3361                         | mq_open()                           | 3404                        |
| 257 | logname                | 3566                         | MQ_OPEN_MAX                         | 3576                        |
| 258 | longjmp()              | 3389, 3399, 3468, 3470, 3564 | MQ_PRIO_MAX                         | 3576                        |
| 259 | LONG_MAX               | 3375                         | mq_receive()                        | 3405                        |
| 260 | LONG_MIN               | 3375                         | mq_send()                           | 3405                        |
| 261 | lp                     | 3566                         | mq_timedreceive()                   | 3436                        |
| 262 | ls                     | 3419, 3521, 3565             | mq_timedsend()                      | 3436                        |
| 263 | lseek                  | 3562                         | msg*()                              | 3401                        |
| 264 | lseek()                | 3409-3410, 3416, 3460, 3500  | msgctl()                            | 3401                        |
| 265 | lseeko()               | 3500                         | msgget()                            | 3401                        |
| 266 | lstat                  | 3561-3562                    | msgrcv()                            | 3401                        |
| 267 | lstat()                | 3334, 3520                   | msgsnd()                            | 3401                        |
| 268 | lutime()               | 3334                         | msync()                             | 3416                        |
| 269 | mailx                  | 3564, 3566                   | multi-byte character                | 3324, 3372-3373             |
| 270 | make                   | 3567-3568                    | multiple tasks                      | 3559, 3565                  |
| 271 | malloc()               | 3399, 3419, 3421, 3449       | munmap()                            | 3415-3417, 3419, 3422       |
| 272 |                        | 3461-3462, 3470, 3472        | mutex                               | 3453                        |
| 273 | man                    | 3564                         | mutex attributes                    |                             |
| 274 | map                    | 3329                         | extended                            | 3456                        |
| 275 | mapped                 | 3329                         | mutex initialization                | 3467                        |
| 276 | margin code notation   | 3316                         | mv                                  | 3520, 3565                  |
| 277 | MAX_CANON              | 3372, 3515, 3576             | name space                          | 3384                        |
| 278 | MAX_INPUT              | 3515, 3576                   | name space pollution                | 3384-3385                   |
| 279 | may                    | 3313                         | NAME_MAX                            | 3515, 3576                  |
| 280 | may, rationale         | 3313                         | nanosleep()                         | 3432, 3434, 3436, 3563      |
| 281 | MCL_FUTURE             | 3413                         | native implementation               | 3330                        |
| 282 | MCL_INHERIT            | 3414                         | newgrp                              | 3566                        |
| 283 | memlockall()           | 3413                         | NGROUPS_MAX                         | 3333, 3515, 3570, 3576      |
| 284 | memory locking         | 3412                         | nice                                | 3565                        |
| 285 | memory management      | 3412, 3562                   | nice value                          | 3330                        |
| 286 | memory management unit | 3413                         | nice()                              | 3426                        |
| 287 | memory object          | 3329                         | nl_langinfo()                       | 3563                        |
| 288 | memory synchronization | 3344                         | nm                                  | 3567                        |
| 289 | memory-resident        | 3329                         | noclobber option                    | 3532                        |
| 290 | mesg                   | 3566                         | nohup                               | 3565                        |
| 291 | message passing        | 3403, 3562-3563              | non-canonical mode input processing | 3372                        |
| 292 | message queue          | 3403                         | non-printable                       | 3339                        |
| 293 | mkdir                  | 3562, 3565                   | NQS                                 | 3547                        |
| 294 | mkfifo                 | 3565                         | obsolescent                         | 3313                        |
| 295 | mkfifo()               | 3325                         | obsolescent, rationale              | 3313                        |
| 296 | mknod()                | 3325                         | od                                  | 3565, 3567                  |
| 297 | mktime()               | 3333                         | open                                | 3562                        |
| 298 | mmap()                 | 3415-3417, 3419              | open file description               | 3330                        |
| 299 | MMU                    | 3413                         | open file descriptors               | 3534                        |
| 300 | modem disconnect       | 3373                         | open()                              | 3325, 3371, 3416-3418, 3520 |
| 301 | monotonic clock        | 3435                         | opendir                             | 3562                        |
| 302 | more                   | 3564, 3566                   | openlog()                           | 3566                        |

## Index

|     |                                       |                  |                                       |                  |
|-----|---------------------------------------|------------------|---------------------------------------|------------------|
| 303 | OPEN_MAX.....                         | 3515, 3576-3577  | popen().....                          | 3564, 3567-3568  |
| 304 | option definitions.....               | 3547             | portability codes.....                | 3315             |
| 305 | optional behavior.....                | 3578             | portable character set.....           | 3350             |
| 306 | options.....                          | 3475             | portable file name character set..... | 3332             |
| 307 | OR lists.....                         | 3539             | positional parameters.....            | 3525             |
| 308 | orphaned process group.....           | 3331, 3398       | POSIX locale.....                     | 3353             |
| 309 | output device.....                    | 3368             | POSIX.1 symbols.....                  | 3384             |
| 310 | output file descriptor                |                  | POSIX.13.....                         | 3419             |
| 311 | duplication.....                      | 3534             | POSIX2_BC_BASE_MAX.....               | 3569             |
| 312 | output mode.....                      | 3374             | POSIX2_BC_DIM_MAX.....                | 3569             |
| 313 | output processing.....                | 3373             | POSIX2_BC_SCALE_MAX.....              | 3569             |
| 314 | overrun conditions.....               | 3499             | POSIX2_BC_STRING_MAX.....             | 3569             |
| 315 | overrun in dumping trace streams..... | 3499             | POSIX2_CHAR_TERM.....                 | 3569             |
| 316 | overrun in trace streams.....         | 3499             | POSIX2_COLL_WEIGHTS_MAX.....          | 3569             |
| 317 | page.....                             | 3331, 3415, 3419 | POSIX2_C_BIND.....                    | 3516, 3568       |
| 318 | PAGESIZE.....                         | 3415, 3459, 3576 | POSIX2_C_DEV.....                     | 3516, 3568       |
| 319 | parallel I/O.....                     | 3460             | POSIX2_EXPR_NEST_MAX.....             | 3570             |
| 320 | parameter expansion.....              | 3529             | POSIX2_FORT_DEV.....                  | 3516, 3568       |
| 321 | parameters.....                       | 3373, 3525       | POSIX2_FORT_RUN.....                  | 3516, 3568       |
| 322 | parameters, positional.....           | 3525             | POSIX2_LINE_MAX.....                  | 3570             |
| 323 | parameters, special.....              | 3525             | POSIX2_LOCALEDEF.....                 | 3516, 3566, 3569 |
| 324 | parent directory.....                 | 3331             | POSIX2_PBS.....                       | 3569             |
| 325 | passwd file.....                      | 3331             | POSIX2_PBS_ACCOUNTING.....            | 3569             |
| 326 | paste.....                            | 3565             | POSIX2_PBS_CHECKPOINT.....            | 3569             |
| 327 | patch.....                            | 3565-3567        | POSIX2_PBS_LOCATE.....                | 3569             |
| 328 | PATH.....                             | 3360             | POSIX2_PBS_MESSAGE.....               | 3569             |
| 329 | path name expansion.....              | 3532             | POSIX2_PBS_TRACK.....                 | 3569             |
| 330 | path name resolution.....             | 3346             | POSIX2_RE_DUP_MAX.....                | 3570             |
| 331 | path name, incomplete.....            | 3326             | POSIX2_SW_DEV.....                    | 3516, 3568       |
| 332 | pathchk.....                          | 3565             | POSIX2_SYMLINKS.....                  | 3516             |
| 333 | pathconf.....                         | 3562             | POSIX2_UPE.....                       | 3516, 3568-3569  |
| 334 | pathconf().....                       | 3325, 3514, 3561 | POSIX2_VERSION.....                   | 3570             |
| 335 | PATH_MAX.....                         | 3389, 3515, 3576 | POSIX_ALLOC_SIZE_MIN.....             | 3402             |
| 336 | pattern matching                      |                  | posix_fadvise().....                  | 3402             |
| 337 | multiple character.....               | 3544             | POSIX_FADV_DONTNEED.....              | 3402             |
| 338 | notation.....                         | 3544             | POSIX_FADV_NOREUSE.....               | 3402             |
| 339 | single character.....                 | 3544             | POSIX_FADV_RANDOM.....                | 3402             |
| 340 | patterns                              |                  | POSIX_FADV_SEQUENTIAL.....            | 3402             |
| 341 | file name expansion.....              | 3545             | POSIX_FADV_WILLNEED.....              | 3402             |
| 342 | pause.....                            | 3561             | posix_madvise().....                  | 3402             |
| 343 | pause().....                          | 3397, 3400       | POSIX_MADV_DONTNEED.....              | 3402             |
| 344 | pax.....                              | 3565-3567        | POSIX_MADV_RANDOM.....                | 3402             |
| 345 | pclose().....                         | 3567             | POSIX_MADV_SEQUENTIAL.....            | 3402             |
| 346 | Pending error.....                    | 3474             | POSIX_MADV_WILLNEED.....              | 3402             |
| 347 | per-thread errno.....                 | 3390             | posix_mem_offset().....               | 3419-3420        |
| 348 | performance enhancements.....         | 3558             | POSIX_REC_INCR_XFER_SIZE.....         | 3403             |
| 349 | PID_MAX.....                          | 3501             | POSIX_REC_MAX_XFER_SIZE.....          | 3403             |
| 350 | pipe.....                             | 3326, 3331, 3562 | POSIX_REC_MIN_XFER_SIZE.....          | 3403             |
| 351 | pipe().....                           | 3397             | POSIX_REC_XFER_ALIGN.....             | 3402             |
| 352 | pipelines.....                        | 3537             | posix_spawn().....                    | 3502             |
| 353 | PIPE_BUF.....                         | 3515, 3576       | posix_spawnp().....                   | 3502             |

|     |                                            |                        |                                                   |                                  |
|-----|--------------------------------------------|------------------------|---------------------------------------------------|----------------------------------|
| 354 | posix_trace_eventid_open()                 | 3494                   | pthread_mutex_timedlock()                         | 3436                             |
| 355 | POSIX_TRACE_LOOP                           | 3499                   | pthread_mutex_trylock()                           | 3456                             |
| 356 | posix_typed_mem_get_info()                 | 3419                   | pthread_mutex_unlock()                            | 3456                             |
| 357 | posix_typed_mem_open()                     | 3419                   | PTHREAD_PROCESS_PRIVATE                           | 3458                             |
| 358 | POSIX_VERSION                              | 3577                   | PTHREAD_PROCESS_SHARED                            | 3458                             |
| 359 | post-mortem filtering of trace event types | 3496                   | pthread_rwlockattr_destroy()                      | 3458                             |
| 360 | pr                                         | 3565-3566              | pthread_rwlockattr_getpshared()                   | 3458                             |
| 361 | pread()                                    | 3460                   | pthread_rwlockattr_init()                         | 3458                             |
| 362 | printf                                     | 3564-3565              | pthread_rwlockattr_setpshared()                   | 3458                             |
| 363 | printing                                   | 3560                   | pthread_rwlock_init()                             | 3458                             |
| 364 | privilege                                  | 3342                   | PTHREAD_RWLOCK_INITIALIZER                        | 3458                             |
| 365 | process group                              | 3370                   | pthread_rwlock_rdlock()                           | 3458                             |
| 366 | process group ID                           | 3326-3327, 3370        | pthread_rwlock_t                                  | 3458                             |
| 367 | process group lifetime                     | 3370                   | pthread_rwlock_tryrdlock()                        | 3458                             |
| 368 | process group, orphaned                    | 3331, 3398             | pthread_rwlock_trywrlock()                        | 3458                             |
| 369 | process groups, concepts in job control    | 3326                   | pthread_rwlock_unlock()                           | 3459, 3472                       |
| 370 | process ID reuse                           | 3347                   | pthread_rwlock_wrlock()                           | 3458                             |
| 371 | process ID, rationale                      | 3501                   | pthread_self()                                    | 3450                             |
| 372 | process management                         | 3558, 3561             | pthread_setconcurrency()                          | 3459                             |
| 373 | process scheduling                         | 3425, 3562             | pthread_setprio()                                 | 3467                             |
| 374 | profiling                                  | 3567                   | pthread_setschedparam()                           | 3467                             |
| 375 | programming manipulation                   | 3490                   | pthread_setspecific()                             | 3450                             |
| 376 | prompting                                  | 3527                   | pthread_spin_lock()                               | 3452, 3471                       |
| 377 | protocol families                          | 3474                   | pthread_spin_trylock()                            | 3452                             |
| 378 | protocols                                  | 3474                   | PTHREAD_STACK_MIN                                 | 3577                             |
| 379 | ps                                         | 3565-3566              | PTHREAD_THREADS_MAX                               | 3577                             |
| 380 | pthread                                    | 3497                   | putc                                              | 3562                             |
| 381 | pthread_attr_getguardsize()                | 3460                   | putc()                                            | 3461                             |
| 382 | pthread_attr_setguardsize()                | 3460                   | putchar                                           | 3562                             |
| 383 | PTHREAD_BARRIER_SERIAL_THREAD              | 3451                   | pwd                                               | 3565                             |
| 384 | pthread_barrier_wait()                     | 3451                   | pwrite()                                          | 3460                             |
| 385 | pthread_cond_init()                        | 3448                   | queuing of waiting threads                        | 3472                             |
| 386 | pthread_cond_timedwait()                   | 3391, 3435, 3457, 3571 | quote removal                                     | 3532                             |
| 387 | pthread_cond_wait()                        | 3391, 3407, 3457       | quoting                                           | 3522                             |
| 388 | pthread_create()                           | 3448-3449              | rand()                                            | 3470                             |
| 389 | PTHREAD_CREATE_DETACHED                    | 3469                   | RE                                                |                                  |
| 390 | PTHREAD_DESTRUCTOR_ITERATIONS              | 3577                   | grammar                                           | 3367                             |
| 391 | pthread_detach()                           | 3469                   | RE bracket expression                             | 3364                             |
| 392 | pthread_getconcurrency()                   | 3459                   | read                                              | 3562, 3564                       |
| 393 | pthread_getcpuclockid()                    | 3438-3439              | read lock                                         | 3458                             |
| 394 | pthread_join()                             | 3391, 3469             | read()                                            | 3326, 3371-3372, 3389, 3397-3398 |
| 395 | PTHREAD_KEYS_MAX                           | 3577                   | .....3400, 3409-3411, 3415-3416, 3460, 3470, 3501 |                                  |
| 396 | pthread_key_create()                       | 3450                   | read-write attributes                             | 3457                             |
| 397 | pthread_mutexattr_gettype()                | 3457                   | read-write locks                                  | 3457                             |
| 398 | pthread_mutexattr_settype()                | 3457                   | readdir                                           | 3562                             |
| 399 | PTHREAD_MUTEX_DEFAULT                      | 3456                   | reading an active trace stream                    | 3499                             |
| 400 | PTHREAD_MUTEX_ERRORCHECK                   | 3456                   | reading data                                      | 3372                             |
| 401 | pthread_mutex_init()                       | 3448                   | readlink                                          | 3562                             |
| 402 | pthread_mutex_lock()                       | 3391, 3456, 3470       | readlink()                                        | 3334                             |
| 403 | PTHREAD_MUTEX_NORMAL                       | 3456                   | realpath                                          | 3562                             |
| 404 | PTHREAD_MUTEX_RECURSIVE                    | 3456                   | realtime                                          | 3402                             |

## Index

|     |                                       |                             |                                   |                            |
|-----|---------------------------------------|-----------------------------|-----------------------------------|----------------------------|
| 405 | realtime signal delivery.....         | 3394                        | semget() .....                    | 3401                       |
| 406 | realtime signal generation .....      | 3394                        | semop() .....                     | 3401                       |
| 407 | realtime signals .....                | 3407                        | sem_init() .....                  | 3405                       |
| 408 | redirect input .....                  | 3533                        | SEM_NSEMS_MAX .....               | 3577                       |
| 409 | redirect output .....                 | 3533                        | sem_open() .....                  | 3405                       |
| 410 | redirection .....                     | 3532                        | sem_timedwait() .....             | 3436                       |
| 411 | regcomp() .....                       | 3567                        | sem_trywait() .....               | 3391, 3407                 |
| 412 | regerror() .....                      | 3567                        | SEM_VALUE_MAX .....               | 3577                       |
| 413 | regexexec() .....                     | 3567                        | sem_wait() .....                  | 3391, 3407                 |
| 414 | regfree() .....                       | 3567                        | sequential lists .....            | 3538                       |
| 415 | regular expression .....              | 3362                        | session .....                     | 3327, 3331, 3370           |
| 416 | definitions .....                     | 3362                        | set .....                         | 3527                       |
| 417 | general requirements .....            | 3363                        | setgid() .....                    | 3333                       |
| 418 | grammar .....                         | 3367                        | setgrent() .....                  | 3340                       |
| 419 | regular file .....                    | 3332                        | setjmp() .....                    | 3564                       |
| 420 | remove() .....                        | 3334                        | setlocale() .....                 | 3563                       |
| 421 | rename .....                          | 3562                        | setlogmask() .....                | 3566                       |
| 422 | rename() .....                        | 3334                        | setpgid() .....                   | 3326-3328, 3370            |
| 423 | renice .....                          | 3565                        | setpriority() .....               | 3426                       |
| 424 | replenishment period .....            | 3428                        | setpwent() .....                  | 3340                       |
| 425 | reserved words .....                  | 3525                        | setrlimit() .....                 | 3521                       |
| 426 | rewinddir .....                       | 3562                        | setsid() .....                    | 3370                       |
| 427 | RE_DUP_MAX .....                      | 3516                        | setuid() .....                    | 3333                       |
| 428 | rm .....                              | 3520, 3565                  | sh .....                          | 3566, 3573                 |
| 429 | rmdir .....                           | 3562, 3565                  | shall .....                       | 3313                       |
| 430 | rmdir() .....                         | 3334, 3390                  | shall, rationale .....            | 3313                       |
| 431 | root directory .....                  | 3332, 3347                  | shared memory .....               | 3417                       |
| 432 | root file system .....                | 3332                        | shell .....                       | 3326-3328                  |
| 433 | root of a file system .....           | 3332                        | SHELL .....                       | 3361                       |
| 434 | routing .....                         | 3474                        | shell .....                       | 3370-3371, 3392, 3397-3398 |
| 435 | RTSIG_MAX .....                       | 3577                        | shell commands .....              | 3535                       |
| 436 | samefile() .....                      | 3500                        | shell errors .....                | 3534                       |
| 437 | SA_NOCLDSTOP .....                    | 3327                        | shell execution environment ..... | 3525, 3543                 |
| 438 | SA_SIGINFO .....                      | 3396                        | shell grammar .....               | 3542                       |
| 439 | scheduling allocation domain .....    | 3465                        | lexical conventions .....         | 3543                       |
| 440 | scheduling contention scope .....     | 3465-3466                   | rules .....                       | 3543                       |
| 441 | scheduling documentation .....        | 3466                        | shell variables .....             | 3526                       |
| 442 | scheduling policy .....               | 3347                        | shell, job control .....          | 3326, 3392, 3398           |
| 443 | SCHED_FIFO .....                      | 3426-3427, 3465, 3472, 3563 | shm*() .....                      | 3401                       |
| 444 | SCHED_OTHER .....                     | 3426                        | shmctl() .....                    | 3401                       |
| 445 | SCHED_RR .....                        | 3426, 3465, 3472, 3563      | shmdt() .....                     | 3401                       |
| 446 | SCHED_SPORADIC .....                  | 3563                        | shm_open() .....                  | 3416-3419                  |
| 447 | seconds since the Epoch .....         | 3332-3333                   | shm_unlink() .....                | 3417-3419                  |
| 448 | security considerations .....         | 3321, 3324                  | should .....                      | 3313                       |
| 449 | .....                                 | 3328, 3340, 3342, 3371      | should, rationale .....           | 3313                       |
| 450 | security, monolithic privileges ..... | 3321                        | SIGABRT .....                     | 3338, 3391                 |
| 451 | sed .....                             | 3565-3566                   | sigaction() .....                 | 3394, 3396                 |
| 452 | sem*() .....                          | 3401                        | SIGBUS .....                      | 3338, 3391                 |
| 453 | semaphore .....                       | 3347                        | SIGCHLD .....                     | 3327, 3393, 3397           |
| 454 | semaphores .....                      | 3405, 3563                  | SIGCLD .....                      | 3397                       |
| 455 | semctl() .....                        | 3401                        | SIGCONT .....                     | 3327, 3394, 3397-3398      |

|     |                                     |                             |                                             |                                          |
|-----|-------------------------------------|-----------------------------|---------------------------------------------|------------------------------------------|
| 456 | SIGEMT .....                        | 3391                        | socket queue limit .....                    | 3474                                     |
| 457 | SIGEV_NONE .....                    | 3394                        | socket receive queue .....                  | 3474                                     |
| 458 | SIGEV_NOTIFY .....                  | 3394                        | socket types .....                          | 3474                                     |
| 459 | SIGEV_SIGNAL .....                  | 3394-3395                   | sockets .....                               | 3474                                     |
| 460 | SIGFPE .....                        | 3338, 3392, 3394, 3396      | IPv4 .....                                  | 3475                                     |
| 461 | SIGHUP .....                        | 3398                        | IPv6 .....                                  | 3475                                     |
| 462 | SIGILL .....                        | 3338, 3392                  | local UNIX connection .....                 | 3475                                     |
| 463 | SIGINT .....                        | 3328, 3463                  | software development .....                  | 3560, 3567                               |
| 464 | SIGIOT .....                        | 3391                        | sort .....                                  | 3565-3566                                |
| 465 | SIGKILL .....                       | 3391, 3394, 3398            | spawn example .....                         | 3502                                     |
| 466 | siglongjmp() .....                  | 3389, 3399, 3564            | spawn() .....                               | 3502, 3561                               |
| 467 | signal .....                        | 3333                        | special built-in .....                      | 3540                                     |
| 468 | signal acceptance .....             | 3393                        | special built-in utilities .....            | 3545                                     |
| 469 | signal actions .....                | 3397                        | special characters .....                    | 3373                                     |
| 470 | signal concepts .....               | 3391                        | special control character .....             | 3374                                     |
| 471 | signal delivery .....               | 3393                        | special parameters .....                    | 3525                                     |
| 472 | signal generation .....             | 3393                        | specific implementation .....               | 3325                                     |
| 473 | signal names .....                  | 3391                        | spin locks .....                            | 3452-3453                                |
| 474 | signal() .....                      | 3391, 3393                  | split .....                                 | 3565                                     |
| 475 | signals .....                       | 3475, 3543                  | sporadic server scheduling policy .....     | 3428                                     |
| 476 | SIGPIPE .....                       | 3338, 3390                  | SSIZE_MAX .....                             | 3501, 3577                               |
| 477 | sigprocmask() .....                 | 3393                        | SS_REPL_MAX .....                           | 3431                                     |
| 478 | SIGRTMAX .....                      | 3395-3396                   | standard I/O streams .....                  | 3400                                     |
| 479 | SIGRTMIN .....                      | 3395-3396                   | stat .....                                  | 3520, 3561-3562                          |
| 480 | SIGSEGV .....                       | 3338, 3392, 3396            | stat() .....                                | 3323, 3416, 3520                         |
| 481 | sigsetjmp() .....                   | 3564                        | state-dependent character encoding .....    | 3350                                     |
| 482 | sigset_t .....                      | 3391                        | statvfs() .....                             | 3521                                     |
| 483 | SIGSTOP .....                       | 3398                        | STREAMS .....                               | 3400                                     |
| 484 | sigsuspend() .....                  | 3397, 3400                  | STREAM_MAX .....                            | 3577                                     |
| 485 | SIGSYS .....                        | 3391                        | Strictly Conforming POSIX Application ..... | 3364                                     |
| 486 | SIGTERM .....                       | 3391                        | strings .....                               | 3567                                     |
| 487 | sigtimedwait() .....                | 3391, 3410, 3435            | strip .....                                 | 3567                                     |
| 488 | SIGTRAP .....                       | 3391                        | structures, additions to .....              | 3385                                     |
| 489 | SIGTSTP .....                       | 3328, 3398                  | stty .....                                  | 3360, 3566                               |
| 490 | SIGTTIN .....                       | 3328, 3371, 3398            | subshells .....                             | 3328                                     |
| 491 | SIGTTOU .....                       | 3327, 3371, 3398            | successfully completed .....                | 3339                                     |
| 492 | SIGUSR1 .....                       | 3391                        | sum .....                                   | 3520-3521                                |
| 493 | SIGUSR2 .....                       | 3391                        | superuser .....                             | 3321, 3333, 3342, 3527                   |
| 494 | sigwait() .....                     | 3391, 3470                  | supplementary group ID .....                | 3333                                     |
| 495 | sigwaitinfo() .....                 | 3391, 3410                  | supplementary groups .....                  | 3342                                     |
| 496 | sigwait_multiple() .....            | 3393                        | symbolic link .....                         | 3334                                     |
| 497 | SIG_DFL .....                       | 3393-3394, 3397             | symbols .....                               | 3384                                     |
| 498 | SIG_IGN .....                       | 3327, 3393-3394, 3397, 3399 | SYMLOOP_MAX .....                           | 3389                                     |
| 499 | simple commands .....               | 3535                        | synchronized I/O .....                      | 3410, 3562                               |
| 500 | single-quotes .....                 | 3522                        | data integrity completion .....             | 3339, 3410                               |
| 501 | SI_USER .....                       | 3396                        | file integrity completion .....             | 3339, 3410                               |
| 502 | sleep .....                         | 3561, 3564                  | synchronously-generated signal .....        | 3338                                     |
| 503 | sleep() .....                       | 3399-3400, 3563             | sysconf .....                               | 3562                                     |
| 504 | socket I/O mode .....               | 3474                        | sysconf() .....                             | 3325, 3413, 3415, 3417, 3463, 3514, 3561 |
| 505 | socket out-of-band data state ..... | 3474                        | syslog() .....                              | 3566                                     |
| 506 | socket owner .....                  | 3474                        | system call .....                           | 3339                                     |



## Index

|     |                                  |                                  |                                   |                             |
|-----|----------------------------------|----------------------------------|-----------------------------------|-----------------------------|
| 507 | system documentation.....        | 3314                             | timer_settime().....              | 3432-3433                   |
| 508 | system environment.....          | 3560, 3566                       | times().....                      | 3389, 3439, 3561            |
| 509 | System III.....                  | 3331, 3500                       | timestamp clock.....              | 3498                        |
| 510 | system interfaces.....           | 3502                             | time_t.....                       | 3333                        |
| 511 | system reboot.....               | 3339                             | token recognition.....            | 3524                        |
| 512 | System V.....                    | 3323, 3328-3329, 3392-3393, 3397 | TOSTOP.....                       | 3327                        |
| 513 | system().....                    | 3564, 3567-3568                  | touch.....                        | 3520, 3565                  |
| 514 | tabs.....                        | 3565                             | tput.....                         | 3566                        |
| 515 | tail.....                        | 3565                             | tr.....                           | 3565-3566                   |
| 516 | talk.....                        | 3564, 3566                       | trace analyzer.....               | 3488                        |
| 517 | tcgetattr().....                 | 3327                             | trace event type-filtering.....   | 3496                        |
| 518 | tcgetpgrp().....                 | 3328, 3370                       | trace event types.....            | 3496                        |
| 519 | tcsetattr().....                 | 3327, 3369                       | trace examples.....               | 3485                        |
| 520 | tcsetpgrp().....                 | 3327-3328                        | trace model.....                  | 3480                        |
| 521 | tee.....                         | 3564                             | trace operation control.....      | 3485                        |
| 522 | TERM.....                        | 3360                             | trace storage.....                | 3484                        |
| 523 | terminal access control.....     | 3371                             | trace stream attribute.....       | 3490                        |
| 524 | terminal device file.....        | 3370                             | trace stream states.....          | 3483                        |
| 525 | closing.....                     | 3373                             | tracing.....                      | 3475                        |
| 526 | terminal type.....               | 3368                             | tracing all processes.....        | 3484                        |
| 527 | termios structure.....           | 3373                             | tracing, detailed objectives..... | 3476                        |
| 528 | test.....                        | 3564-3565                        | triggering.....                   | 3498                        |
| 529 | TeX.....                         | 3565                             | troff.....                        | 3565                        |
| 530 | text file.....                   | 3339                             | trojan horse.....                 | 3321                        |
| 531 | thread.....                      | 3339                             | true.....                         | 3564                        |
| 532 | thread cancelability states..... | 3470                             | tty.....                          | 3566                        |
| 533 | thread cancelability type.....   | 3470                             | ttyname().....                    | 3461                        |
| 534 | thread cancelation.....          | 3468, 3470                       | TTY_NAME_MAX.....                 | 3577                        |
| 535 | thread cancelation points.....   | 3470                             | typed memory.....                 | 3419                        |
| 536 | thread concurrency level.....    | 3459                             | TZNAME_MAX.....                   | 3577                        |
| 537 | thread creation attributes.....  | 3447                             | tzset().....                      | 3563                        |
| 538 | thread ID.....                   | 3340, 3463                       | ualarm.....                       | 3561                        |
| 539 | thread interactions.....         | 3474                             | UID_MAX.....                      | 3501                        |
| 540 | thread mutex.....                | 3464                             | uid_t.....                        | 3340                        |
| 541 | thread read-write lock.....      | 3472                             | ULONG_MAX.....                    | 3514                        |
| 542 | thread scheduling.....           | 3464                             | umask.....                        | 3566                        |
| 543 | thread stack guard size.....     | 3459                             | umask().....                      | 3561                        |
| 544 | thread-safe.....                 | 3340                             | umount().....                     | 3330                        |
| 545 | thread-safe function.....        | 3340                             | unalias.....                      | 3564                        |
| 546 | thread-safety.....               | 3347, 3460                       | uname.....                        | 3566                        |
| 547 | thread-safety, rationale.....    | 3347                             | uname().....                      | 3561                        |
| 548 | thread-specific data.....        | 3449                             | unbounded priority inversion..... | 3467                        |
| 549 | threads.....                     | 3447                             | undefined.....                    | 3314                        |
| 550 | implementation models.....       | 3449                             | undefined, rationale.....         | 3314                        |
| 551 | tilde expansion.....             | 3528                             | unexpand.....                     | 3565                        |
| 552 | time.....                        | 3564, 3567                       | uniq.....                         | 3565-3566                   |
| 553 | time().....                      | 3389                             | unlink.....                       | 3562                        |
| 554 | timeouts.....                    | 3440                             | unlink().....                     | 3334, 3390, 3416-3417, 3419 |
| 555 | timers.....                      | 3431                             | unsafe functions.....             | 3399                        |
| 556 | TIMER_ABSTIME.....               | 3432-3433                        | unspecified.....                  | 3314                        |
| 557 | TIMER_MAX.....                   | 3577                             | unspecified, rationale.....       | 3314                        |

|     |                                 |                                   |
|-----|---------------------------------|-----------------------------------|
| 558 | until loop .....                | 3540                              |
| 559 | user database .....             | 3340                              |
| 560 | user database access .....      | 3340                              |
| 561 | user requirements .....         | 3557                              |
| 562 | usleep .....                    | 3561                              |
| 563 | utility .....                   | 3348                              |
| 564 | utility argument syntax .....   | 3375                              |
| 565 | utility conventions .....       | 3375                              |
| 566 | utility syntax guidelines ..... | 3376                              |
| 567 | utime .....                     | 3562                              |
| 568 | utime() .....                   | 3334                              |
| 569 | uucp .....                      | 3566                              |
| 570 | uudecode .....                  | 3565-3566                         |
| 571 | uuencode .....                  | 3565-3566                         |
| 572 | variable assignment .....       | 3348                              |
| 573 | variables .....                 | 3525                              |
| 574 | VEOF .....                      | 3374                              |
| 575 | VEOL .....                      | 3374                              |
| 576 | Version 7 .....                 | 3347, 3522                        |
| 577 | vhangup() .....                 | 3329                              |
| 578 | vi .....                        | 3564, 3566                        |
| 579 | virtual processor .....         | 3341                              |
| 580 | VMIN .....                      | 3374                              |
| 581 | VTIME .....                     | 3374                              |
| 582 | wait .....                      | 3564                              |
| 583 | wait() .....                    | 3389, 3392, 3397-3398, 3400, 3561 |
| 584 | waitpid() .....                 | 3327, 3331, 3398, 3501, 3561      |
| 585 | wc .....                        | 3565                              |
| 586 | WERASE .....                    | 3372                              |
| 587 | while loop .....                | 3540                              |
| 588 | who .....                       | 3566                              |
| 589 | wide-character codes .....      | 3350                              |
| 590 | word expansions .....           | 3528                              |
| 591 | wordexp() .....                 | 3567                              |
| 592 | wordfree() .....                | 3567                              |
| 593 | write .....                     | 3562, 3564, 3566                  |
| 594 | write lock .....                | 3458                              |
| 595 | write() .....                   | 3326-3327, 3371, 3389, 3397-3398  |
| 596 | .....                           | 3400, 3409-3411, 3415-3416, 3501  |
| 597 | writing data .....              | 3373                              |
| 598 | WUNTRACED .....                 | 3327                              |
| 599 | xargs .....                     | 3564                              |
| 600 | XSI .....                       | 3341                              |
| 601 | XSI IPC .....                   | 3401                              |
| 602 | XSI supported functions .....   | 3454                              |
| 603 | XSI threads extensions .....    | 3455                              |
| 604 | yacc .....                      | 3567-3568                         |